

Um guia para o motor de busca  
Java

# Luce EM AÇÃO ne

Otis Gospodnetic

Erik Hatcher  
PREFÁCIO Doug Cutting



MANNING

Equipe Fly-®

## Lucene in Action



# Lucene in Action

ERIK HATCHER  
OTIS GOSPODNETIC



MANNING  
Greenwich  
(74 ° W. longo.)

Para obter informações on-line e ordenação deste e outros livros de Manning, acesse  
[www.manning.com](http://www.manning.com). A editora oferece descontos neste livro quando ordenado em quantidade.

Para mais informações, favor contatar:

Departamento de Vendas Especiais  
Manning Publications Co.  
209 Bruce Park AvenueFax: (203) 661-9018  
Greenwich, CT 06830email: [orders@manning.com](mailto:orders@manning.com)

© 2005 por Manning Publications Co. Todos os direitos reservados.

Nenhuma parte desta publicação pode ser reproduzida, armazenada em um sistema de recuperação ou transmitida,  
de qualquer forma ou por meio eletrônico, mecânico, fotocópia, ou outro, sem  
permissão prévia por escrito do editor.

Muitas das designações usadas por fabricantes e vendedores para distinguir seus produtos  
são marcas registradas. Onde essas designações aparecem no livro, e Manning  
Publicações estava ciente de uma reivindicação da marca, as designações foram impressas em inicial  
caps ou todos os caps.

- ✉ Reconhecendo a importância de preservar o que foi escrito, é política de Manning  
ter os livros que publicam impresso em papel acid-free, e exercer nossos melhores esforços  
para esse fim.



ISBN 1-932394-28-1

Impresso nos Estados Unidos da América  
1 2 3 4 5 6 7 8 9 10 - VHG - 08 07 06 05 04

Para Ethan, Jakob, e Carole

-E.H.

Para a comunidade Lucene, chichimichi e Saviotlama

-O.G.



# conteúdos breve

---

PART Lucene CORE 1 .....	
1 · Conheça Lucene 3	
2 · Indexação 28	
3 · Adicionando pesquisa para a sua aplicação 68	
4 · Análise de 102	
5 · Técnicas avançadas de pesquisa 149	
6 · Estendendo pesquisa 194	
PART 2 APLICADA Lucene .....	221
7 · Análise de formatos de documentos comuns 223	
8 · Ferramentas e extensões 267	
9 · Lucene portas 312	
10 · Estudos de caso 325	



# conteúdo

---

prefácio xvii  
prefácio xix  
reconhecimentos xxii  
xxv sobre este livro

---

PART 1 CORE Lucene ..... 1

---

## 1

### Conheça Lucene 3

1,1 Evolução da organização da informação e acesso 4

1,2 Compreensão Lucene 6

    O que Lucene é de? 7 Que Lucene pode fazer por você?

    História do Lucene 9 Quem usa Lucene 10 portas Lucene: Perl,  
    Python, C + +, .NET, Ruby 10

1,3 Indexação e busca 10

    O que é indexação, e por que é importante? 10

    O que está procurando? 11

1,4 Lucene em ação: um exemplo de aplicativo 11

    Criação de um índice 12 Pesquisando um índice 15

- 1,5 Compreender as classes básicas de indexação 18
  - DirectoryIndexWriter 19 19
  - Field documento 20 20
  - Analyzer 19
- 1,6 Entendendo o núcleo busca classes 22
  - IndexSearcher Term 23 23
  - TermQuery 24 Hits 24
  - Consulta 23
- 1,7 Revisão de produtos de busca alternativo 24
  - IR 24 bibliotecas de indexação e busca aplicações 26
  - Recursos on-line 27
- 1,8 Resumo 27

## 2

### Indexação 28

- 2,1 Compreender o processo de indexação 29
  - Conversão para texto 29
  - Análise de 30
  - Índice de escrever 31
- 2,2 Basic operações de índice 31
  - Adicionar documentos a um índice 31
  - Remover documentos a partir de um índice 33
  - Recuperando 36
  - Documentos Atualização em um índice 36
- 2,3 Documentos impulsionar e Campos 38
- 2,4 Indexação datas 39
- 2,5 40 números de indexação
- 2,6 Os campos de indexação usada para classificar 41
- 2,7 Controlar o processo de indexação 42
  - Ajuste de desempenho de indexação 42
  - In-memória de indexação: RAMDirectory 48
  - Limitando tamanhos de campo: 54 MaxFieldLength
- 2,8 Otimizando um índice 56
- 2,9 Simultaneidade, o thread de segurança, e problemas de bloqueio 59
  - Regras de concorrência 59
  - Tópico segurança-60
  - Índice de travamento 62
  - índice desativar o bloqueio 66
- 2,10 Depuração indexação 66
- 2,11 Resumo 67

## 3

### Adicionando pesquisa para a sua aplicação 68

#### 3,1 Implementação de um recurso de busca simples 69

À procura de um termo específico <sup>70</sup>Análise de um usuário entrou expressão: QueryParser 72 consulta

#### 3,2 Usando IndexSearcher 75

Trabalhando com Hits 76 Folheando Hits 77  
Índices de leitura na memória 77

#### 3,3 Compreensão de pontuação Lucene 78

Lucene, você tem um monte de 'splainin' para fazer!

#### 3,4 Consultas criar programaticamente <sup>80</sup> 81

Pesquisando por prazo: Pesquisando TermQuery 82 dentro de uma faixa:

Pesquisando RangeQuery 83 em uma string: PrefixQuery 84

Consultas combinando: BooleanQuery 85 Pesquisando por frase:

Pesquisando PhraseQuery 87 por curinga: WildcardQuery 90

Pesquisar por termos semelhantes: FuzzyQuery 92

#### 3,5 Análise de expressões de consulta: 93 QueryParser

Operadores Query.toString 94 Boolean Agrupamento 94 95

Campo de seleção de 95 Faixa de buscas 96 consultas Frase 98

Curinga e prefixo consultas 99 consultas 99 Impulsionar Fuzzy

Para consultas 99 QueryParse ou não QueryParse? 100

#### 3,6 Resumo 100

## 4

### Análise de 102

#### 4,1 Usando analisadores de 104

Indexação análise análise QueryParser 105 106

Análise versus análise: quando um analisador não é apropriado 107

#### 4,2 Analisando o analisador de 107

O que é um token? 108 <sup>109</sup> TokenStreams sem censura

Visualizando analisadores <sup>112</sup> fim de filtragem pode ser importante 116

#### 4,3 Usando o built-in analisadores 119

StopAnalyzer 119 • StandardAnalyzer 120

#### 4,4 Lidar com campos de palavra-chave 121

Analisador de palavra-chave alternativo

#### 4,5 "Só como" consulta <sup>125</sup> 125

4,6	Sinônimos, apelidos e palavras que significa que o 128 mesmo
	Visualizando posições token 134
4,7	Decorrentes de análise 136
	Buracos deixando 136 Colocá-lo em conjunto 137
	Lote do furo de problemas 138
4,8	Idioma questões de análise 140
	Unicode e codificações 140 Analisando non-Inglês
	Analisando 141 Línguas idiomas asiáticos 142
	Zajian 145
4,9	Nutch análise 145
4,10	Resumo 147

## 5

### Técnicas avançadas de pesquisa 149

5,1	Classificação dos resultados da pesquisa 150
	Usando uma espécie 150 classificação por relevância classificação pelo índice 152
	Classificando ordem 153 por um campo de 154 Reverter ordem de classificação 154
	Ordenar por vários campos 155 Selecionando um tipo de campo de classificação 156
5,2	Usando Phrases Prefix Query 157 para classificar 157 efeito de desempenho de 158
5,3	Consultando em vários campos de uma só vez 159
5,4	Consultas Span: nova jóia Lucene está escondido 161
	Construção de bloco de spanning, SpanTermQuery 163 Encontrar abrange, no início de um campo de 165 Spans perto de um 166 Excluindo outra sobreposição vão desde jogos 168 Em todo o globo SpanQuery 169 e 170 QueryParser
5,5	Filtragem uma pesquisa 171
	Usando DateFilter 171 Usando QueryFilter 173
	Filtros de segurança 174 A alternativa QueryFilter 176
	Cache resultados de filtro 177 Além dos filtros built-in 177
5,6	Pesquisando em vários índices Lucene 178
	Usando MultiSearcher pesquisar usando 178 Multithreaded ParallelMultiSearcher 180

- 5,7 Aproveitando vetores prazo 185
  - Livros como este 186 Qual categoria? 189
- 5,8 Resumo 193

## 6

### Estendendo pesquisa 194

- 6,1 Usando um método de classificação personalizada 195
  - Acessando valores utilizados na classificação custom 200
- 6,2 Desenvolvimento de um HitCollector custom 201
  - Sobre BookLinkCollector 202 • Usando BookLinkCollector 202
- 6,3 Estendendo QueryParser 203
  - Comportamento QueryParser personalização de 203 Proibir fuzzy e consultas curinga 204 Handling campo numérico gama-205 consultas Permitindo que ordenou consultas frase 208
- 6,4 Usando um filtro personalizado 209
  - Usando uma query filtrada 212
- 6,5 Testes de desempenho 213
  - Testando a velocidade de um teste de carga pesquisa 213 217 QueryParser novamente! 218 Morais de testes de desempenho 220
- 6,6 Resumo 220

PART 2 APPLIED LUCENE ..... 221

---

## 7

### Análise de formatos de documentos comuns 223

- 7,1 Manipulação de rich-text documentos 224
  - Criação de um interface comum DocumentHandler 225
- 7,2 Indexação XML 226
  - Análise e indexação usando SAX 227 • Análise e indexação usando Digester 230
- 7,3 Indexar um documento PDF 235
  - Extrair texto e indexação usando PDFBox 236
  - Built-in Lucene suporte 239
- 7,4 Indexação de um documento HTML 241
  - Obter os dados de origem HTML 242 • Usando JTidy 242
  - Usando NekoHTML 245

- 7,5 Indexação de um documento do Word Microsoft 248
  - Utilizando POI 249 Usando TextMining.org 's API 250
- 7,6 Indexação de um documento RTF 252
- 7,7 Indexar um documento de texto simples-253
- 7,8 Criação de um quadro manipulação de documentos-254
  - FileHandler interface ExtensionFileHandler 255 257
  - Aplicação FileIndexer 260 Usando FileIndexer 262
  - FileIndexer desvantagens, e como estender o quadro 263
- 7,9 Outras ferramentas de extração de texto 264
  - Gerenciamento de documentos sistemas e serviços 264
- 7,10 Resumo 265

## 8

### Ferramentas e extensões 267

- 8,1 Jogando no Sandbox do Lucene 268
- 8,2 Interagir com um índice de 269
  - luci: a interface de linha de comando 269 Luke: o Índice de Lucene Toolbox LIMO 271: Lucene Índice Monitor 279
- 8,3 Analisadores, tokenizers e TokenFilters, oh meu 282
  - SnowballAnalyzer 283 • Obtenção do Sandbox analisadores 284
- 8,4 Java Development com Ant e Lucene 284
  - Usando a tarefa <index> 285 Criando um documento personalizado Instalação manipulador de 286 290
- 8,5 JavaScript browser utilitários 290
  - JavaScript construção de consulta e validação 291 Escaping caracteres especiais 292 Usando JavaScript apoio 292
- 8,6 Sinônimos de WordNet 292
  - Construindo o sinônimo índice de 294 sinônimos para Amarrar WordNet um analisador de 296 Calling em 297 Lucene
- 8,7 Destacando termos da consulta 300
  - Destacando com CSS 301 • Hits destacando 303
- 8,8 Filtros encadeamento 304
- 8,9 Armazenar um índice em Berkeley DB 307
  - Codificação para 308 DbDirectoryInstalação de 309 DbDirectory

- 8,10 Construção do 309 Sandbox
  - Check it out 310 • Formiga na 310 Sandbox
- 8,11 Resumo 311

## 9

### Lucene portas 312

- 9,1 Relação portas 'para Lucene 313
- 9,2 CLucene 314
  - Plataformas suportadas 314 API compatibilidade 314
  - Supporte a Unicode Performance 316 317 317 usuários
- 9,3 dotLucene 317
  - API compatibilidade compatibilidade 317 Index 318
  - Performance 318 usuários 318
- 9,4 Plucene 318
  - API compatibilidade compatibilidade 319 Index 320
  - Performance 320 usuários 320
- 9,5 Lupy 320
  - API compatibilidade compatibilidade Índice 320 322
  - Performance 322 usuários 322
- 9,6 PyLucene 322
  - Compatibilidade API compatibilidade Índice 323 323
  - Performance 323 usuários 323
- 9,7 Resumo 324

## 10

### Estudos de caso 325

- 10,1 Nutch: "O NPR dos motores de busca" 326
  - Mais em profundidade 327
  - características Nutch 328
- 10,2 Usando o Lucene em jGuru 329
  - Léxicos tópico e categorização de documentos do banco de dados da pesquisa 330
  - estrutura de 331 campos de índice 332 de indexação e de conteúdo
  - preparação 333 Consultas 335 JGuruMultiSearcher 339
  - Diversos 340
- 10,3 Usando o Lucene em SearchBlox 341
  - Por que escolher o Lucene? 341 SearchBlox arquitectura 342
  - Resultados de pesquisa 343 343 suporte de idiomas
  - Relatórios Resumo Engine 344 344

10,4	Inteligência competitiva com o Lucene em XM-XtraMind de InformationMinder™ 344
	A arquitetura do sistema 347• Como Lucene tem nos ajudado a 350
10,5	Alias-i: a variação ortográfica com Lucene 351
	Alias-i aplicação arquitetura variação Orthographic 352 354
	O modelo de canal ruidoso de correção ortográfica 355 O vetor
	modelo de comparação de ortografia variação 356 A Lucene subpalavra
	analizador de Precisão 357, eficiência e outros aplicativos 360
	Misturando referências contexto 360 361
10,6	Pesquisar Artful em Michaels.com 361
	Indexação de conteúdo de conteúdo Searching 362 367
	Pesquisa estatísticas 370 Resumo 371
10,7	Eu amo Lucene: TheServerSide 371
	Capacidade de busca de construção mais 371 de alto nível
	infra-estrutura de 373 Construindo o índice de 374 Pesquisando a
	índice de configuração 377: um lugar para todos governar 379
	Web tier: TheSeeeeeeeeeerverSide? 383 Resumo 385
10,8	Conclusão 385

Apêndice A: Instalando Lucene 387

Apêndice B: Lucene formato de índice 393

apêndice C: Recursos 408

índice 415

# **prefácio**

---

Lucene começou como um projeto de auto-serviço. No final de 1997, meu trabalho incerto, eu procurou algo meu para o mercado. Java foi a programação nova quente língua, e eu precisava de uma desculpa para aprender. Eu já sabia como escrever software de busca, e pensei que eu poderia preencher um nicho de escrever software de pesquisa em Java. Então eu escrevi Lucene.

Alguns anos mais tarde, em 2000, eu percebi que eu não gostava de coisas de mercado. Eu tinha nenhum interesse na negociação de licenças e contratos, e eu não queria contratar pessoas e construir uma empresa. Eu gostava de software de gravação, e não vendê-lo. Então, joguei Lucene acima em SourceForge, para ver se de fonte aberta pode deixe-me continuar a fazer o que eu gostei.

A poucas pessoas começaram a usar Lucene imediatamente. Cerca de um ano depois, em 2001, pessoal da Apache Lucene oferecido a adotar. O número de mensagens diárias sobre as listas de discussão Lucene cresceu de forma constante. Contribuições de código começou a pingar dentro

A maioria eram adições em torno das bordas Lucene: Eu ainda era o ativo somente desenvolvedor que totalmente grokked seu núcleo. Ainda assim, Lucene foi no caminho para tornar-ção de um projeto real de colaboração.

Agora, em 2004, Lucene tem um pool de desenvolvedores ativos com profundas entende-  
sultados de seu núcleo. Não estou mais envolvido na maioria desenvolvimento dia-a-dia; sub-  
acréscimos substanciais e melhorias são feitas regularmente por esta equipe forte.

Através dos anos, Lucene foi traduzido para várias outras programações de linguagens de programação, incluindo C++, C#, Perl e Python. Na Java original,

e nestas outras encarnações, Lucene é usado muito mais amplamente do que eu jamais teria sonhado. Ela busca forças em diversas aplicações como a discussão grupos de empresas Fortune 100, trackers bug comerciais, pesquisa de e-mail sup-necidas pelo Microsoft, e um motor de busca da web que as escalas de bilhões de páginas. Quando,

em eventos da indústria, sou apresentado a alguém como "o cara Lucene," com mais freqüência pessoas que não me diga como eles usaram Lucene em um projeto. Eu ainda acho que eu tenho só

ouvi falar de uma pequena fração de todas as aplicações Lucene.

Lucene é muito mais usado do que jamais teria sido se eu tivesse tentado para vendê-lo. Os desenvolvedores de aplicativos parecem preferir open source. Em vez de ter que contatar o suporte técnico quando têm um problema (e depois esperar por uma resposta, esperando que eles foram corretamente entendido), eles podem freqüentemente basta olhar para o código fonte para diagnosticar os seus problemas. Se isso não bastasse, o apoio do livre fornecidas pelos seus pares nas listas de discussão é melhor do que apoiar mais comercial. A funcionamento do projeto open-source como o Lucene faz desenvolvedores de aplicações mais eficiente e produtiva.

Lucene, através de código aberto, tornou-se algo muito maior do que eu jamais imaginou que seria. Eu defini-lo ir, mas levou a esforços combinados da Lucene comunidade para fazê-lo prosperar.

Então, o que vem por aí para Lucene? Eu não posso te dizer. Armado com este livro, você está agora um membro da comunidade Lucene, e cabe a você tomar Lucene para novos lugares. Bon voyage!

DOUG CORTE  
Criador do Lucene e Nutch

# **prefácio**

---

De Erik Hatcher

Eu fui intrigado com a busca e indexação dos primeiros dias do Internet. Tenho boas recordações (circa 1991), de gestão de uma lista de e-mail usando majordomo, MUSH (Shell Mail Usuário), e um punhado de Perl, awk, e shell scripts. Eu implementei uma interface web CGI para permitir aos utilizadores pesquisar a lista arquivos e perfis de outros usuários usando truques grep debaixo das cobertas. Então veio Yahoo, AltaVista, Excite e, todos os que visitei regularmente.

Depois do meu primeiro filho, Jakob, nasceu, meu arquivo de fotos digital começou crescendo rapidamente. Fiquei intrigado com a idéia de desenvolver um sistema para gerenciar as fotos para que eu pudesse anexar meta-dados a cada imagem, tais como palavras-chave e data de tomada, e, é claro, localizar as imagens facilmente em qualquer dimensão que escolheu. No final de 1990, eu prototyped uma abordagem baseada em sistema de arquivos usando

Tecnologias Microsoft, incluindo Microsoft Index Server, Active Server Pages, e um componente COM terceiros para manipulação de imagens. Na época, o meu provedor profissional foi consumido com essas mesmas tecnologias. Eu era capaz de calçada juntos uma aplicação atraente em um par de dias de reposição em tempo hacking.

Minha vida profissional mudou em direção tecnologias Java, e meu computação vida consistia em menos Microsoft Windows. Em um esforço para reimplementar meu arquivo de fotos pessoais e motor de pesquisa em tecnologias Java em um operating sistema independente de forma, me deparei com Lucene. Lucene facilidade de uso do bem

excedeu as minhas expectativas, eu tinha experimentado vários outros de código aberto bibliotecas e ferramentas que eram muito mais simples conceitualmente ainda muito mais complexo de usar.

Em 2001, Steve Loughran e comecei a escrever Java Development com Ant (Manning). Levamos a idéia de uma aplicação do motor de busca de imagem e generalizou como um motor de busca de documentos. Este exemplo de aplicação é usada em todo o Ant livro e pode ser personalizado como um motor de busca de imagem. O empate para Ant vem não apenas de um simples processo de construção compilar e pacote, mas também de um personalizado tarefa Ant, <index>. Criamos índices que os arquivos durante o processo de construção usando Lucene. Esta tarefa Ant agora vive em Sandbox Lucene e é descrito no seção 8.4 deste livro.

Esta tarefa Ant está em uso para o meu sistema de produção de blogs personalizados, que eu chamo BlogScene (<http://www.blogscene.org/erik>). Eu corro um processo de construção Ant, depois de criar uma entrada de blog, que cria índices novas entradas e carrega-los para o meu servidor. Meu servidor de blog consiste em um servlet, alguns modelos de velocidade, e um índice Lucene, permitindo consultas ricos, mesmo syndication de consultas. Comparado a outros blogging sistemas, BlogScene é muito inferior em recursos e finesse, mas o texto completo recursos de pesquisa são muito poderosos.

Agora estou trabalhando com o grupo de Pesquisa Aplicada em Patacriticism na Universidade da Virgínia (<http://www.patacriticism.org>), onde eu estou colocando o meu texto analysis, indexação e busca especialização para o teste e alongamento minha mente com discussões sobre como a física quântica diz respeito à literatura. "Os poetas são os UNACKnowledged engenheiros do mundo."

#### Da Otis Gospodnetic

Meu interesse e paixão pela recuperação de informação e gestão começou a during meus anos de estudante no Middlebury College. Naquele tempo, eu descobri uma imensa fonte de informações conhecido como internet. Embora a Web ainda estava em sua infância, a necessidade de longo prazo para a recolha, análise, indexação e pesquisa era evidente. Fiquei obcecado com a criação de repositórios de informações puxado a partir da Web, começou a escrever crawlers, e sonhava em maneiras de pesquisar o col- selecionada informações. Eu vi de pesquisa como a aplicação de um assassino em grande parte inexplorado território. Com que, no fundo da minha mente, comecei o primeiro em minha série de projetos que compartilham um denominador comum: coleta e busca de informações.

Em 1995, seu colega Marshall Levin e criei WebPh, um open-source programa usado para coletar e recuperar informações de contato pessoal. Em essência, era um livro simples telefone eletrônico com uma interface web (CGI), um dos o primeiro de seu tipo na época. (Na verdade, ele foi citado como um exemplo de arte prévia em um processo judicial no final de 1990!) Universidades e instituições governamentais em torno

o mundo têm sido os adotantes principal deste programa, e muitos ainda estão usá-lo. Em 1997, armado com minha experiência WebPh, eu passou a criar Populus, um popular páginas em branco no tempo. Mesmo que a tecnologia (semelhante ao de WebPh) era rudimentar, Populus realizado o seu peso e foi um-comparativa ble corresponder aos jogadores grandes como WhoWhere, Bigfoot, e Infospace.

Depois de dois projetos que incidiu sobre as informações de contato pessoal, que era hora de explorar novos territórios. Comecei minha próxima aventura, Infojump, que envolveu abate informações de alta qualidade a partir de boletins on-line, revistas, jornais e revistas. Além do meu próprio software, que consistia em grandes conjuntos de Perl módulos e scripts, Infojump utilizou um web crawler chamado Webinator e uma completa produto de busca de texto chamado Texit. O serviço prestado pelo Infojump em 1998 foi muito semelhante ao de hoje FindArticles.com.

Embora WebPh, Populus, e Infojump servido seus propósitos e foram totalmente funcional, todos eles tinham limitações técnicas. A peça que falta em cada um dos eles era uma biblioteca de recuperação de informação poderosa que permitiria texto completo Buscas apoiado por índices invertidos. Em vez de tentar reinventar a roda, eu começou a procurar uma solução que eu suspeitava estava lá fora. No início de 2000, eu encontrados Lucene, a peça que faltava eu estava procurando, e eu caí no amor com ele.

Entrei para o projeto Lucene no início, quando ele ainda morava no SourceForge e, mais tarde, na Fundação Apache Software Lucene quando migraram para lá em 2002. Minha devoção a Lucene decorre do fato de ser um componente essencial de muitas idéias que tinha fila na minha mente ao longo dos anos. Uma dessas idéias foi Simpy, meu lat- est projeto de estimativa. Simpy é uma característica-rico do serviço web que permite pessoais tag usuários, informação do índice, pesquisar, compartilhar e encontrada online. Ele faz uso pesado de Lucene, com milhares de seus índices, e é alimentado por Nutch, outro projeto de Doug Corte (ver capítulo 10). Minha participação ativa no projeto Lucene resultou em uma oferta de Manning para co-autor Lucene in Action com Erik Hatcher.

Lucene in Action é a fonte mais abrangente de informações sobre Lucene. As informações contidas nos próximos 10 capítulos abrange todos os conhecimento que você precisa para criar aplicações sofisticadas construído em cima do Lucene. É o resultado de um processo de colaboração muito bom e ágil, muito parecida com a dentro da comunidade Lucene. Lucene e Lucene in Action exemplificam o que pessoas podem alcançar quando têm interesses semelhantes, a disposição de ser flexível, eo desejo de contribuir para o pool de conhecimentos global, apesar do fato de que eles ainda têm de se encontrar pessoalmente.

# reconhecimentos

---

Em primeiro lugar, agradecer aos nossos cônjuges, Carole (Erik) e Margaret (Otis), para suportar a autoria deste livro. Sem o seu apoio, este livro nunca se materializaram. Erik graças seus dois filhos, Ethan e Jakob, por sua paciência e compreensão quando o pai trabalhou neste livro ao invés de brincar com eles.

Estamos sinceramente e humildemente grato a Corte Doug. Sem Doug generosidade para com o mundo, não haveria Lucene. Sem o outro Committers Lucene, Lucene teria características muito menos, mais bugs, e uma tempo muito mais difícil prosperando com a adoção crescente de Lucene. Muitos graças a todos os committers incluindo Peter Carlson, Tal Dayan, Scott Ganyo, Eugene Gluzberg, Brian Goetz, Christoph Goller, Mark Harwood, Tim Jones, Daniel Naber, Andrew C. Oliver, Dmitry Serebrennikov, Kelvin Tan, e Matt Tucker. Da mesma forma, agradecemos a todos aqueles que contribuíram o caso estudos que aparecem no capítulo 10: Dion Almaer, Michael Cafarella, Bob Carpenter, Karsten Konrad, Terence Parr, Robert Selvaraj, Ralf Steinbach, Paredes Holger Stenzhorn, e Craig.

Nossos agradecimentos ao pessoal em Manning, incluindo Marjan BACE, Lianna WLAsuik, Karen Tegtmeyer, Susannah Pfalzer, Mary Piergies, Leslie Haimes, David Roberson, Lee Fitzpatrick, Ann Navarro, Clay Andres, Tiffany Taylor, Denis Dalinnik, e Susan Forsyth.

Manning arredondado um grande conjunto de revisores, a quem agradecemos para melhorar nossos rascunhos para o que você agora lê. Os revisores incluem Doug Warren, Scott Ganyo, Bill Fly, Oliver Zeigermann, Jack Hagan, Michael Oliver, Brian Goetz, Ryan Cox, John D. Mitchell, e Norman Richards. Terry Steichen desde feedback informal, ajudando a esclarecer alguns pontos ásperos. Extra-especiais agradecimentos para Brian Goetz para sua edição técnica.

#### Erik Hatcher

Eu, pessoalmente, agradecer a Otis por seus esforços com este livro. Embora ainda temos de atender

em pessoa, a Otis foi uma alegria para trabalhar. Ele e eu temos dado bem e chegaram a acordo sobre a estrutura eo conteúdo deste livro por toda parte.

Graças ao Java Java em Charlottesville, Virgínia, para me manter com fio e sem fio, graças, também, para Greenberry para ficar abertos até mais tarde do que Java e Java mantendo-me longe de problemas por não ter acesso à Internet (update: agora eles têm wi-fi, para o desespero da minha produtividade).

As pessoas que eu me cerquei com enriquecer a minha vida mais do que qualquer coisa. David Smith tem sido um mentor ao longo da vida, e seu brilho continua a deslumbrar mim, ele me deu muita comida para o pensamento sobre visualização Lucene (A maioria dos quais eu ainda estou lutando para compreender plenamente, e peço desculpa que não

torná-lo este manuscrito). Jay Zimmerman eo Fluff Não, sym-Just Stuff possum circuito foram drasticamente influente para mim. O NFJS regular alto-falantes, incluindo Dave Thomas, Halloway Stuart, James Duncan Davidson, Jason Hunter, Ted Neward, Ben Galbraith, Glenn Vanderburg, Venkat Subramanian, Paredes Craig, e Bruce Tate têm sido uma grande fonte de apoio e amizade. Rick Hightower e Nick Lesiecki merecem menção especial, eles ambos foram fundamentais para me empurrando para além dos limites da minha técnica e habilidades de comunicação. Palavras fazem pouco para expressar o entusiasmo incansável e encorajamento Mike Clark tem me dado todo escrito Lucene in Action.

Tecnicamente, Mike contribuiu com a JUnitPerf testes de desempenho-exemplos, mas sua energia, ambição e amizade eram muito mais fundamental.

Eu estendo a gratidão a soluções Darden para trabalhar comigo através do meu tirando livro e agenda de viagens e permitindo-me a manter um baixo stress do dia a tempo parcial emprego. A Darden colega de trabalho, Dave Engler, desde que o Swing esqueleto CellPhone aplicação que tenho demonstrado nas sessões NFJS e JavaOne e que é incluída na secção 8.6.3; graças, Dave! Outros colegas de trabalho Darden, Andrew Shan-não e Nick Skriloff, deu-nos uma visão Verity, uma solução competitiva para usar Lucene. Amy Moore forneceram uma visão gráfica. Meu grande amigo Davie Murray pacientemente criado figura 4.4, suportando vários pedidos de revisão. Daniel Steinberg

é um amigo pessoal e mentor, e ele me permitiu ar idéias Lucene como artigos em [java.net](#). Simon Galbraith, um grande amigo e agora um guru de busca, e eu tinha idéias divertidas saltando ao redor em busca de e-mail.

**Otis Gospodnetic**

Escrita Lucene in Action Foi um esforço grande para mim, não só por causa do técnico conteúdo que ele contém, mas também porque eu tinha que encaixar em lado com um dia de trabalho de tempo integral, projetos de estimação e, claro, minha vida pessoal. Alguém precisa descobrir como para estender dias para pelo menos 48 horas. Trabalhar com Erik era um prazer: Seu ágil desenvolvimento de competências são impressionantes, sua flexibilidade e admirável compaixão.

Eu odeio reconhecimentos brega, mas eu realmente não posso agradecer o suficiente Margaret

por ser tão solidário e paciente comigo. Devo-lhe um suprimento vitalício de chá e arroz. Meus pais Sanja e Vito abriu meus olhos no início da minha infância por mostrando-me como grande parte do mundo que podiam, e que fez um mundo de diferenças de referência. Eles também foram os que sugeriu que eu escrever meu primeiro livro, que eliminado o medo do livro escrito no início da minha vida.

Agradeço também a John Stewart eo resto da Geração Wireless, Inc., meu empregador, por ser paciente comigo ao longo do ano passado. Se você comprar uma cópia do livro, eu vou agradecer a você também!

# sobre este livro

---

Lucene in Action fornece detalhes, melhores práticas, advertências, dicas e truques para usando o melhor open-source Java motor de busca disponíveis.

Este livro assume que o leitor está familiarizado com programação Java básico. Lucene em si é um único arquivo Java Archive (JAR) e integra-se mais simples Java programa console stand-alone, bem como a empresa mais sofisticados aplicação.

## Roteiro

---

Nós organizamos uma parte deste livro para cobrir o núcleo Lucene Application Programming Interface (API) na ordem que você é provável encontrar-lo como integrar Lucene em suas aplicações:

- No capítulo 1, você encontra Lucene. Nós introduzimos algumas informações básicas recuperação de terminologia, e notamos competição principal do Lucene. Com a perder tempo, nós imediatamente construir indexação simples e pesquisar aplicações que você pode colocar direito de usar ou adaptar para o seu necessidades. Este aplicativo de exemplo abre a porta para explorar o resto das capacidades do Lucene.
- Capítulo 2 familiariza-lo com as operações básicas de indexação Lucene. Nós descrever os vários tipos de campo e técnicas para os números de indexação

e datas. Ajustando o processo de indexação, otimização de um índice, e como lidar com fio de segurança são cobertos.

- Capítulo 3 leva você através de pesquisa básica, incluindo detalhes de como Lucene classifica os documentos com base em uma consulta. Discute-se a fundamental tipos de consulta e também como elas podem ser criadas através de humanos entrou expressões de consulta.
- Capítulo 4 mergulha profundamente no coração da magia do Lucene indexação, o anal-Ysis processo. Nós cobrimos os blocos de construção, incluindo tokens analisador, token cérregos, e filtros de token. Cada um dos analisadores de built-in recebe a sua quota de atenção e detalhe. Construímos vários analisadores personalizados, apresentando syn-injeção onym e metaphone (como soundex) de substituição. Análise de não-Inglês línguas é dada atenção, com exemplos específicos de analyzing texto chinês.
- Capítulo 5 pega onde o capítulo busca parou, com a análise agora em mente. Nós cobrimos várias características busca avançada, incluindo a separação, filtragem, e aproveitando vetores prazo. Os tipos de consulta avançadas tornam sua aparência, incluindo a espetacular SpanQuery da família. Finalmente, nós cobrir o apoio do Lucene built-in para consulta vários índices, mesmo em paralelo e remotamente.
- Capítulo 6 vai muito além da busca avançada, mostrando-lhe como ampliar os recursos em busca do Lucene. Você vai aprender a personalizar resultados da pesquisa de classificação, estender a análise de expressão de consulta, implementar hit col- da selecção de, e ajustar o desempenho da consulta. Ufa!

Parte 2 vai além de built-in do Lucene instalações e mostra o que pode ser feito ao redor e acima Lucene:

- No capítulo 7, criamos uma estrutura reutilizável e extensível para análise documentos em Word, HTML, XML, PDF e outros formatos.
- Capítulo 8 inclui uma miscelânea de extensões e ferramentas em torno Lucene. Descrevemos visualizar várias índice Lucene e ferramentas de desenvolvimento, bem como muitos brinquedos interessantes na Sandbox do Lucene. Destacando termos de pesquisa é uma extensão Sandbox de tal forma que é provável que você precisa, juntamente com outros goodies como a construção de um índice a partir de um processo de construção Ant, usando noncore.
- Capítulo 9 demonstra os portos de Lucene para várias línguas, como C++, C#, Perl e Python.

- Capítulo 10 traz todos os detalhes técnicos de volta Lucene em foco com muitos estudos de caso maravilhosa contribuiu por aqueles que construíram juros ing, aplicações rápidas e escaláveis com Lucene em seu núcleo.

## **Quem deve ler este livro?**

---

Desenvolvedores que precisam de recursos poderosos de busca embutido em suas aplicações deve ler este livro. Lucene in Action também é indicado para desenvolvedores que estão curioso sobre Lucene ou indexação e técnicas de pesquisa, mas que não podem ter uma necessidade imediata de usá-lo. Adicionando Lucene know-how à sua caixa de ferramentas é val-

capazes para futuros projetos de pesquisa-é um tema quente e continuará a ser no futuro.

Este livro utiliza principalmente a versão Java do Lucene (do Apache Jakarta), ea maioria dos exemplos de código usam a linguagem Java. Os leitores familiarizados com Java vai ser em casa. Expertise Java será útil, no entanto, Lucene foi portado para uma série de outras línguas, incluindo C + +, C #, Python, e Perl. Os conceitos, técnicas e até mesmo a própria API são comparáveis entre as versões da linguagem Java e outros de Lucene.

## **Exemplos de código**

---

O código fonte para este livro está disponível no site de Manning em <http://www.manning.com/hatcher2>. Instruções para usar este código são fornecidos no Arquivo README incluído com o pacote de código-fonte.

A maioria dos códigos mostrados neste livro foi escrito por nós e está incluído no pacote de código-fonte. Algum código (particularmente o código do estudo de caso) não é fornecidas em nosso pacote de código-fonte; os trechos de código mostrado lá são de propriedade

pelos contribuintes e são doados como é. Em alguns casos, incluímos um pequeno trecho de código a partir de codebase Lucene, que está licenciado sob a Apache Software License (<http://www.apache.org/licenses/LICENSE-2.0>).

Exemplos de código não incluem pacotes e instruções de importação, para conservar espaço; referem-se ao código-fonte real para esses detalhes.

## **Por JUnit?**

Acreditamos exemplos de código em livros deve ser top-notch qualidade e do mundo real aplicável. O típico "Olá mundo" exemplos freqüentemente insultar nossa inteligência e geralmente fazem pouco para ajudar os leitores a ver como realmente se adaptam ao seu ambiente.

Nós tomamos uma abordagem única para os exemplos de código Lucene in Action. Muitos dos nossos exemplos são casos de teste atual JUnit (<http://www.junit.org>). JUnit,

o facto de Java framework de testes de unidade, facilmente permite que o código de afirmar que a-par

pressuposto cular funciona como esperado de uma forma repetível. Automatizando JUnit casos de teste através de uma IDE ou Ant permite que um passo (ou nenhuma passos com contínua

integração) de confiança. Optamos por utilizar JUnit neste livro, porque nós usá-lo diariamente em nossos outros projetos e quero que você veja como nós código realmente. Teste

Driven Development (TDD) é uma prática de desenvolvimento é altamente defendem.

Se você não estiver familiarizado com JUnit, por favor leia o seguinte primer. Nós também sugiro que você leia Teste de unidade pragmática em Java com JUnit por Dave Thomas e Andy Hunt, seguido por Manning JUnit em Ação por Vincent Massol e Ted Husted.

### Primer JUnit

Esta seção é uma introdução rápida e reconhecidamente incompleta para JUnit. Vamos fornecer as bases necessárias para compreender nossos exemplos código. Primeiro, o nosso teste JUnit

casos ampliar junit.framework.TestCase e muitos estendê-lo indiretamente por meio de nosso costume LiaTestCase classe base. Nossas classes de teste de concreto aderir a uma nomeação

convenção: se o sufixo nome da classe com Teste. Por exemplo, nosso QueryParser testes estão em QueryParserTest.java.

Corredores JUnit executar automaticamente todos os métodos com a assinatura público void testXXX (), Onde XXX é um nome arbitrário, mas significativa. JUnit de teste métodos deve ser conciso e claro, mantendo bom design de software em mente (Como não repetir-se, criando funcionalidade reutilizáveis, e assim por diante).

### Afirmações

JUnit é construído em torno de um conjunto de afirmar declarações, libertando-o para testes de código claramente

e deixando a estrutura JUnit lidar com suposições falhou e relatar a detalhes. Os mais utilizados afirmar afirmação é assertEquals, Há uma número de variantes sobrecarregada do assertEquals assinatura do método para várias tipos de dados. Um método de ensaio exemplo parecido com este:

```
testExample public void () {  
    SomeObject obj = new someObject ();  
    assertEquals (10, obj.someMethod ());  
}
```

O afirmar métodos de lançar uma exceção de tempo de execução se o valor esperado (10, neste

exemplo) não é igual ao valor real (o resultado da chamada someMethod em obj, Em neste exemplo). Além de assertEquals, Existem vários outros afirmar métodos para conveniência. Nós também usamos assertTrue (expressão), assertFalse (expressão), e assertNull (expressão) declarações. Estes testar se a expressão é verdadeiro, falso e nulo, respectivamente.

O afirmar declarações têm sobrecarregado assinaturas que levem um adicional `Corda` parâmetro como o primeiro argumento. Este `Corda` argumento é utilizado exclusivamente para fins de relatório, dando ao desenvolvedor mais informações quando um teste falhar. Nós utilizar este `Corda` argumento de mensagem para ser mais descritivo (ou às vezes cômica).

Codificando as nossas suposições e expectativas em casos de teste JUnit neste manner, nós nos livramos da complexidade de grandes sistemas que construímos e pode focar em detalhes menos de cada vez. Com uma massa crítica de casos de teste no lugar, podemos permanecermos confiantes e ágil. Essa confiança vem de saber que a mudança código, como algoritmos de otimização, não vai quebrar outras partes do sistema, porque se assim fosse, o nosso conjunto de testes automatizados iria deixar-nos saber muito antes de o código feito isso para produção. Agilidade vem de ser capaz de manter a base de código limpo através de refactoring. Refatoração é a arte (ou é uma ciência?) De mudar o estrutura interna do código para que ele acomoda exigências evoluindo sem afetar a interface externa de um sistema.

### JUnit em contexto

Vamos dar o que temos dito até agora sobre JUnit e enquadrá-lo dentro do contexto de este livro. Casos de teste JUnit em última análise, se estendem desde `junit.framework.TestCase`, e métodos de ensaio têm a `testXXX public void ()` assinatura. Um dos nossos testes casos (do capítulo 3) é mostrado aqui:

```
BasicSearchingTest public class LiaTestCase {  
  
    testTerm public void () throws Exception {  
        IndexSearcher searcher = new IndexSearcher (diretório);  
        Termo t = novo Termo ("sujeito", "ant");  
        Query = new TermQuery (t);  
        Hits hits = searcher.search (query);  
        assertEquals ("JDwA", 1, hits.length());  
  
        t = novo Termo ("subject", "junit");  
        hits = searcher.search (novo TermQuery (t));  
        assertEquals (2, hits.length());  
  
        searcher.close ();  
    }  
}
```

LiaTestCase estende  
junit.framework.  
TestCase  
diretório vem  
de LiaTestCase

Um hit esperado para  
busca de "formiga"

Dois hits esperado  
para "junit"

É claro, vamos explicar a API Lucene utilizados neste caso de teste mais tarde. Aqui nós vamos focalizar nos detalhes JUnit. Uma variável usada em `testTerm,diretório`, Não está definido nesta classe. JUnit oferece um gancho de inicialização que é executado antes de cada método de ensaio, o gancho é um método com o `setUp public void ()` assinatura. Nosso `LiaTestCase` classe base implementa `setUp` desta maneira:

```
public abstract class LiaTestCase estende TestCase {  
    private String indexDir = System.getProperty ("index.dir");  
    diretório protegido Directory;  
  
    setUp protected void () throws Exception {  
        = diretório FSDirectory.getDirectory (indexDir, false);  
    }  
}
```

Se o nosso primeiro afirmar `em testTerm` falhar, nós vemos uma exceção como este:

```
junit.framework.AssertionFailedError: JDwA espera: <1> mas foi: <0>  
    em lia.searching.BasicSearchin Tes  
    ⇒testTerm (BasicSearchingTes .java 20)
```

Esta falha indica nossos dados de teste é diferente do que esperamos.

### Testes Lucene

A maioria dos testes neste teste livre Lucene em si. Na prática, isso é realístico? Não é a idéia de escrever casos de teste que prova o nosso próprio código, e não as bibliotecas-los

eus? Há uma interessante viravolta Test Driven Development utilizado para o aprendizado uma API: Test Driven Learning. É extremamente útil para escrever testes diretamente a um nova API, a fim de aprender como ele funciona eo que você pode esperar dele. Isto é precisamente o que temos feito na maioria de nossos exemplos de código, de forma que testes são testes

Lucene em si. Não jogue esses testes de aprendizagem de distância, no entanto. Mantê-los em torno de para garantir as suas expectativas da API são verdadeiras quando você atualizar para uma nova versão

versão da API, e refazer-los quando a mudança inevitável API é feita.

Em alguns casos, usamos objetos mock para fins de teste. São objetos mock usados como sondas enviadas para a lógica do negócio real para afirmar que o negócio lógica está funcionando corretamente. Por exemplo, no capítulo 4, temos um `SynonymEngine` interface (ver secção 4.6). A lógica do negócio real que utiliza essa interface é um analisador. Quando queremos testar o analisador de si, é irrelevante o tipo de `SynonymEngine` é usado, mas queremos usar aquele que tem bem definidas e prevêem comportamento capaz. Nós criamos uma `MockSynonymEngine`, Permitindo-nos de forma confiável e pre-

dictably testar o nosso analisador. Objetos Mock ajudar a simplificar casos de teste de tal forma que eles testam

apenas uma faceta de um sistema único de cada vez ao invés de ter entrelaçadas dependências que levam à complexidade na solução de problemas que realmente deu errado quando

um teste falhar. Um efeito legal de usar objetos mock vem do design mudanças que nos leva a, tais como separação de interesses e concepção de interfaces usando, em vez direta de implementações concretas.

### Nossos dados de teste

A maioria do nosso livro gira em torno de um conjunto comum de dados de exemplo para fornecer consistência e evitar ter que grok um conjunto inteiramente novo de dados para cada seção. Este exemplo consiste em dados detalhes do livro. A Tabela 1 mostra os dados para que você pode fazer referência a ela e fazer o sentido dos nossos exemplos.

Tabela 1 Dados da amostra utilizada ao longo deste livro

Título / Autor	Categoria	Assunto
A Arte Moderna da Educação Rudolf Steiner	/ Educação / pedagogia	filosofia da educação psicologia prática Waldorf
Segredos Imperial da Saúde e Longevidade Bob Falhas	/ Saúde / alternativas / chinesas	dieta medicina chinesa qi saúde ervas gong
Tao Te Ching 道德經 Stephen Mitchell	/ Filosofia / Leste	taoism
Gödel, Escher, Bach: Eternal Golden Braid um Douglas Hofstadter	/ Tecnologia / computadores / ai	número de inteligência artificial teoria musical matemática
Mindstorms Seymour Papert	/ Tecnologia / programação de computadores / educação	// crianças computadores poderosos idéias educação LOGO
Java Development com Ant Erik Hatcher, Steve Loughran	/ Tecnologia / computadores / programação	apache jakarta ant ferramenta de construção java junit desenvolvimento
JUnit em Ação Vincent Massol, Ted Husted	/ Tecnologia / computadores / programação	unidade junit testes mock objetos
Lucene in Action Otis Gospodnetic, Erik Hatcher	/ Tecnologia / computadores / programação	lucene pesquisa
Extreme Programming Explained Kent Beck	/ Tecnologia / programação de computadores / metodologia	// extremas de programação ágil desenvolvimento orientado a testes metodologia
Tapeçaria em Ação Howard Lewis Ship-	/ Tecnologia / computadores / programação	tapeçaria interface de usuário web componentes
The Pragmatic Programmer Dave Thomas, Andy Hunt	/ Tecnologia / computadores / programação	metodologia ágil pragmática ferramentas de desenvolvimento

Os dados, além dos campos mostrados na tabela, inclui campos para ISBN, URL, e publicação mês. Os campos para a categoria e assunto são os nossos subjetividade própria valores tivo, mas a outras informações é objetivamente factual sobre os livros.

## Convenções de código e downloads

---

Código-fonte em listas ou de texto está em um fonte de largura fixa para separá-lo texto comum. Nomes de métodos Java, dentro do texto, geralmente não incluem a completa assinatura do método.

A fim de acomodar o espaço de página disponíveis, o código foi formatado com uma largura limitada, incluindo marcadores de continuação de linha se for o caso.

Nós não incluem declarações de importação e raramente se referem a classe totalmente qualificado

nomes-isso fica no caminho e ocupa valioso espaço. Referem-se a Java Lucene docs para esta informação. Todos os IDEs decentes têm um excelente suporte para automaticamente

adição de declarações de importação; Erik blissfully códigos sem saber totalmente qualificado classnames usando IntelliJ IDEA, e Otis faz o mesmo com XEmacs. Adicione o Lucene JAR ao classpath do seu projeto, e está tudo pronto. Também no classpath questão (que é um incômodo notório), assumimos que o JAR Lucene e qualquer outros JARs necessários estão disponíveis no classpath e não mostrá-lo explicitamente.

Nós criamos um monte de exemplos para este livro que estão disponíveis gratuitamente para você.

Um arquivo. Zip de todo o código está disponível em site de Manning para a web Lucene em Ação: <http://www.manning.com/hatcher2>. Instruções detalhadas sobre a execução do código de exemplo são fornecidos no diretório principal do arquivo expandido como um Arquivo README.

## Autor online

---

A compra de Lucene in Action inclui acesso gratuito a uma corrida privada web forum por Manning Publications, onde você pode discutir o livro com os autores e outros leitores. Para acessar o fórum e subscrevê-lo, aponte seu navegador para <http://www.manning.com/hatcher2>. Esta página fornece informações sobre como entrar no forum uma vez que são registrados, que tipo de ajuda está disponível, e os regras de conduta do Fórum.

## Sobre os autores

---

Erik Hatcher códigos, escreve e fala sobre temas técnicos que ele acha divertido e desafiador. Ele tem escrito software para um número de diversas indústrias usando diversas tecnologias e linguagens. Erik co-autoria Java Development com Ant (Manning, 2002) com Steve Loughran, um livro que recebeu admira-aclamação da indústria ful. Desde o lançamento do primeiro livro de Erik, ele tem falado em numerosos locais, incluindo o Fluff Não, Just Stuff simpósio circuito, JavaOne,

Convenção O'Reilly Open Source, o Open Source Content Management Conferências, reuniões e muitos Java User Group. Como Apache Software Foundation membro dação, ele é um contribuinte ativo e committer no Apache várias projetos, incluindo Lucene, Ant, e Tapestry. Erik trabalha atualmente na Universidade de Humanidades Virginia "departamento de apoio Pesquisa Aplicada em PATA-s criticism. Ele vive em Charlottesville, Virginia, com sua bela esposa, Carole, e dois filhos surpreendente, Ethan e Jakob.

Otis Gospodnetic tem sido um colaborador ativo Lucene por quatro anos e mantém a jGuru Lucene FAQ. Ele é um Engenheiro de Software no Wireless Generation. Além disso, uma empresa que desenvolve soluções tecnológicas para avaliações educacionais de alunos e professores. Em seu tempo livre, ele desenvolve Simpy, um Personal Web serviço que utiliza o Lucene, que ele criou para fora de sua paixão pelo conhecimento, recuperação de informação e gestão. Anteriores publicações técnicas incluem vários artigos sobre o Lucene, publicado pela O'Reilly Network e IBM desenvolvendo Works. Otis também escreveu Escolher e ser escolhido: Perseguiendo Educação na América, um guia para estrangeiros que desejam estudar nos Estados Unidos, é baseado em sua própria experiência. Otis é da Croácia e vive atualmente em New York City.

## Sobre o título

---

Ao combinar introduções, visões gerais e exemplos de como fazer, o Em Ação livros são projetados para ajudar a aprendizagem e lembrar. De acordo com a pesquisa em ciência cognitiva, a coisas que as pessoas se lembrar são coisas que descobrir durante auto-motivado exploração.

Embora ninguém em Manning é um cientista cognitivo, estamos convencidos de que para aprendendo a se tornar permanente que deve passar por estágios de exploração, reprodução, e, curiosamente, re-dizer do que está sendo aprendido. As pessoas entendem e lembre-se coisas novas, o que significa dizer que dominá-las, só depois ativamente explorá-las. Os seres humanos aprendem em ação. Uma parte essencial de um Em Ação guia é que é exemplo-driven. Que incentiva o leitor a experimentar as coisas, para brincar com novo código, e explorar novas idéias.

Há uma outra, mais mundana razão, para o título deste livro: os nossos leitores estão ocupados. Eles usam os livros para fazer um trabalho ou resolver um problema. Eles precisam de livros que permitir-lhes saltar e saltar para fora facilmente e aprender exatamente o que eles querem apenas quando eles querem. Eles precisam de livros que a ajuda-los em ação. Os livros nesta série são projetados para esses leitores.

## Sobre a ilustração da capa

A figura na capa da *Lucene in Action* é "Um habitante da costa da Síria." A ilustração é tomado de uma coleção de fantasias do Império Otomano publicado em 1 de janeiro de 1802, por William Miller de Old Bond Street, Londres. O título da página está faltando da coleção e temos sido incapazes de segui-lo até à data. Tabela do livro de conteúdo identifica os números em Inglês e francês, e cada ilustração traz os nomes dos dois artistas que trabalharam no que, ambos os quais, sem dúvida, se surpreender ao descobrir sua arte enfeitando a frente capa de um livro de programação de computador ... 200 anos depois.

A coleção foi comprada por um editor de Manning em uma pulga antiquário market na "Garagem" na West 26th Street em Manhattan. O vendedor era um Americano com sede em Ankara, Turquia, ea transação ocorreu exatamente como ele foi arrumando seu stand para o dia. O editor de Manning não tinha em sua pessoa a quantidade substancial de dinheiro que era necessário para a compra e um crédito cartão e verifique se ambos os educadamente recusou.

Com o vendedor a voar de volta para Ancara naquela noite a situação estava ficando sem esperança. Qual foi a solução? Ele acabou por ser nada mais do que um antigo acordo verbal moda selado com um aperto de mão. O vendedor simplesmente propôs que o dinheiro ser transferido para ele por fio eo editor saiu com as informações do vendedor do banco em um pedaço de papel e da carteira de imagens debaixo do braço. Escusado será dizer, que os fundos transferidos no dia seguinte, e continuamos agradecidos e impressionados com a confiança desta pessoa desconhecida em um dos nós. Ele lembra algo que poderia ter acontecido há muito tempo.

As fotos da coleção Otomano, como as outras ilustrações que aparecem em nosso cobre, trazer à vida a riqueza e variedade de costumes vestido de dois séculos atrás. Eles lembram a sensação de isolamento e distância do que período e de cada período histórico outros senão o nosso próprio presente hipercinético.

Códigos de vestimenta mudaram desde então e da diversidade por região, tão rica em o tempo, desapareceu. Agora é difícil dizer o habitante de um continente permanente de outra. Talvez, tentandovê-lo de forma otimista, temos trocou uma cultura diversidade cultural e visual para uma vida mais variada pessoal. Ou uma mais variada e vida intelectual e técnico interessante.

Nós da Manning celebrar a inventividade, a iniciativa, e, sim, a diversão de o negócio de computadores com o livro de cobre com base na rica diversidade da regional vida de dois séculos atrás, trouxe de volta à vida com as fotos desta coleção.

# Parte 1

## Core Lucene

T

ele primeiro semestre deste livro cobre out-of-the-box out (errr ... do JAR) Lucene. Você vai "Meet Lucene" com uma visão geral e desenvolver um completo indexação e busca de aplicação. Cada capítulo sucessivas sistematicamente mergulha em áreas específicas. "Indexação" dados e documentos e, posteriormente, "Searching" para eles são os primeiros passos para usar Lucene. Retornando a um glossed-over processo de indexação, "Análise", irá preencher a sua compreensão de o que acontece com o texto indexado com Lucene. Pesquisar é onde Lucene realmente brilha: Esta seção conclui com "Advanced procurando" técnicas usando apenas os recursos built-in, e "pesquisa Estendendo" showcasing Lucene extensibilidade para fins de costume.



# Conheça Lucene

## Este capítulo aborda

- Compreensão Lucene
- Usando a API de indexação básica
- Trabalhando com a API de busca
- Considerando-se produtos alternativos

Um dos principais fatores por trás da popularidade Lucene eo sucesso é a sua simplicidade. A exposição cuidadosa da sua indexação e busca API é um sinal de bem-desenhado software. Conseqüentemente, você não precisa de um conhecimento aprofundado sobre como

Indexação Lucene de informação e recuperação de trabalho para começar a usá-lo.

Além disso, API simples do Lucene exige que você aprender a usar apenas um punhado de suas classes.

Neste capítulo, vamos mostrar-lhe como realizar indexação e busca básica com Lucene com ready-to-use exemplos de código. Em seguida, uma breve introdução todas as elementos essenciais que você precisa saber para ambos os processos. Nós também fornecemos comentários breves de competir Java / não-Java, livre, e produtos comerciais.

## 1.1 Evolução da organização da informação e acesso

---

, A fim de dar sentido à complexidade percebida do mundo, os seres humanos têm categorizações inventado, classificações, gêneros, espécies, e outros tipos de hierárquica esquemas organizacionais. O sistema decimal Dewey para categorizar itens de uma coleção de biblioteca é um exemplo clássico de uma categorização hierárquica esquema. A explosão da Internet e repositórios de dados eletrônico tem trouxe grandes quantidades de informação ao nosso alcance. Algumas empresas, como como Yahoo, têm feito organização e classificação de dados on-line os seus negócios. Com o tempo, no entanto, a quantidade de dados disponíveis tornou-se tão vasta que precisávamos alternativo, formas mais dinâmicas de encontrar informações. Embora nós pode classificar os dados, de arrasto através de centenas ou milhares de categorias e sub-categorias de dados já não é um método eficiente para encontrar informações.

A necessidade de localizar rapidamente informações no mar de dados não se limita à Internet reino desktop-computadores podem armazenar dados cada vez mais. Mudança diretórios e expandindo e recolhendo hierarquias de pastas não é uma forma eficaz forma de acesso a documentos armazenados. Além disso, já não usam computadores apenas por suas habilidades de computação primas: Eles também servem como players multimídia e Dispositivos de armazenamento de mídia. Os usos para os computadores exigem a capacidade de rapidamente encontrar um pedaço específico de dados, o que é mais, nós precisamos fazer media, tais como ricos imagens, vídeo e arquivos de áudio em vários formatos, fácil de localizar.

Com esta abundância de informação, e com o tempo sendo um dos mais pre-commodities preciosos para a maioria das pessoas, precisamos ser capazes de fazer flexível, free-form, consultas ad-hoc que pode rapidamente atravessam as fronteiras rígidas e categoria encontrar exatamente o que estamos procurando, mas exige o mínimo de esforço possível.

Para ilustrar a omnipresença de busca através da Internet e da mesa-superior, figura 1.1 mostra uma pesquisa para lucene no Google. O número inclui um contexto



Figura 1.1 Convergência da Internet com o Google e pesquisar o navegador web.

menu que nos permite usar o Google para pesquisar o texto destacado. A Figura 1.2 mostra o Apple Mac OS X Finder (o equivalente a Explorer da Microsoft em Windows) eo recurso de busca embutido no canto superior direito. O Mac OS X música player, iTunes, também tem capacidades de busca, como mostrado na figura 1.3.

Funcionalidade de pesquisa está em toda parte! Todos os principais sistemas operacionais têm embedded busca. A inovação mais recente é o recurso Spotlight (<http://www.apple.com/macosx/tiger/spotlighttech.html>) anunciado por Steve Jobs na

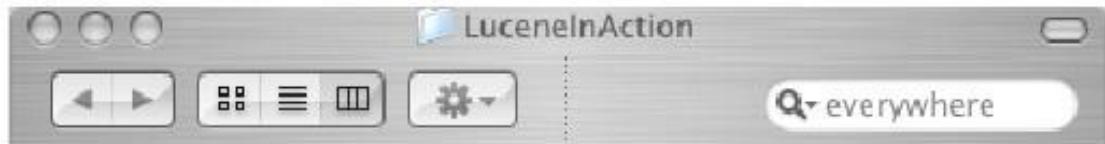


Figura 1.2 Mac OS X Finder com sua capacidade de procura incorporadas.



Figura 1.3 iTunes da Apple intuitivamente incorpora a funcionalidade de pesquisa.

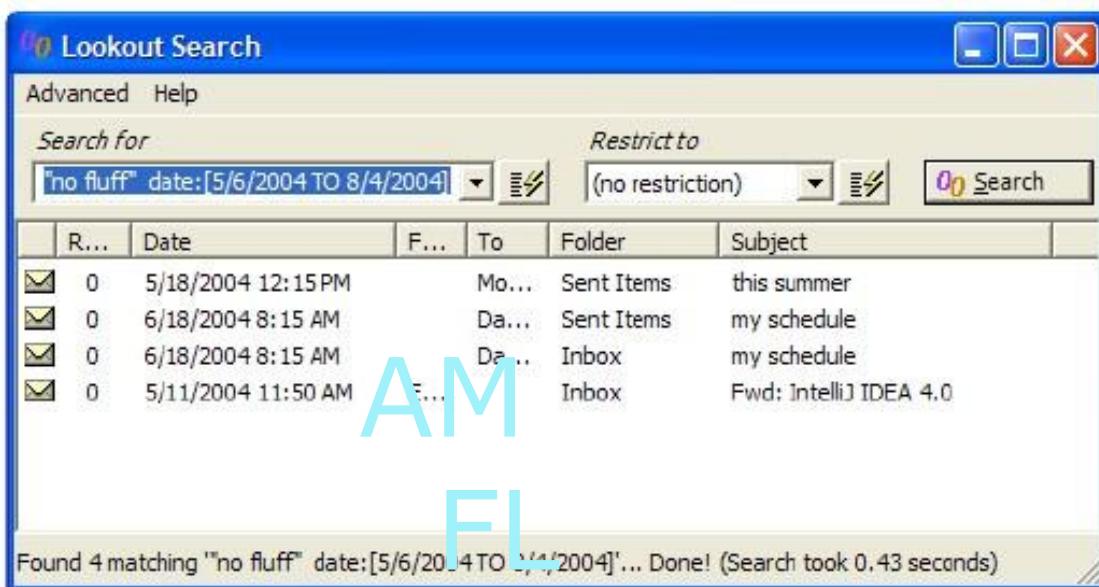


Figura 1.4 Microsoft recém adquirida do produto Lookout, usando por baixo Lucene.Net.

próxima versão do Mac OS X (apelidado Tiger); ele integra a indexação e buscação em todos os tipos de arquivo, incluindo metadados ricos específicas para cada tipo de arquivo, tais como e-mails, contatos e mais.<sup>1</sup>

Google tem ido IPO. A Microsoft lançou uma versão beta de sua busca MSN motor; em uma nota potencialmente relacionados, a Microsoft adquiriu Lookout, um produto alavancando a porta Lucene.Net de Lucene para indexar e Microsoft busca Outlook e-mail e pastas pessoais (como mostrado na figura 1.4). Yahoo! comprado Overdrive e é aumentado suas capacidades de pesquisa personalizado.

Para entender o que Lucene papel desempenha na pesquisa, vamos começar com o básico e aprender sobre o que Lucene é e como ele pode ajudá-lo com as suas necessidades de pesquisa.

## Compreendendo 1.2 Lucene

Diferentes pessoas estão lutando contra a sobrecarga de informações, utilizando-problema mesmo

diferentes abordagens. Alguns têm vindo a trabalhar em novas interfaces de usuário, alguns em agentes inteligentes, e outros no desenvolvimento de ferramentas de busca mais sofisticadas, como

Lucene. Antes de entrar em ação com amostras de código mais adiante neste capítulo, vamos dar-lhe uma imagem de alto nível do que Lucene é, o que não é, e como ele veio a ser.

<sup>1</sup> Erik admite a seu gosto de todas as coisas Apple.

### 1.2.1 O Lucene é

Lucene é uma alta performance, escalável Information Retrieval biblioteca (IR). Ele permite que adicionar indexação e busca recursos para suas aplicações. Lucene é uma madura, livre, projeto de código aberto implementado em Java, é um membro da Apache Jakarta popular família de projetos, licenciado sob a Apache liberal Licença de software. Como tal, Lucene é atualmente, e tem sido por alguns anos, a gratuito mais popular biblioteca Java IR.

**NOTA** Ao longo do livro, usaremos o termo recuperação da informação (IR) para descrever as ferramentas de busca como o Lucene. As pessoas geralmente se referem a bibliotecas como IR pesquisa motores, mas você não deve confundir bibliotecas IR com motores de busca da web.

Como você vai descobrir logo, Lucene fornece uma API núcleo simples, mas poderosa que requer uma compreensão mínima de indexação de texto completo e pesquisa. Você precisa aprender sobre apenas um punhado de suas classes, a fim de começar a integrar em Lucene um aplicativo. Porque Lucene é uma biblioteca Java, não faz suposições sobre o que os índices e pesquisas, o que lhe dá uma vantagem sobre uma série de outras aplicações de busca.

Novas pessoas para Lucene muitas vezes confundem-lo para um aplicativo de ready-to-use como um arquivo de pesquisa-programa, um web crawler, ou um motor de busca web site. Não é isso que Lucene é: Lucene é uma biblioteca de software, um kit de ferramentas, se você vai, e não um full-featured aplicação de pesquisa. Preocupa-se com a indexação de texto e pesquisa, e faz as coisas muito bem. Lucene permite que o seu negócio de aplicativos com regras de negócio específica para o seu domínio do problema, escondendo a complexidade de indexação e em busca de implementação de um simples de usar API. Você pode pensar em Lucene como uma camada que as aplicações se sentar em cima de, como mostrado na figura 1.5.

Um número de aplicações full-featured de pesquisa foram construídas em cima de Lucene. Se você está procurando algo pré-construídos ou um quadro para rastreamento, manuseamento de documentos, e busca, consulte o Wiki Lucene "powered by" página (<Http://wiki.apache.org/jakarta-lucene/PoweredBy>) para muitas opções: Zilverline, SearchBlox, Nutch, LARM, e jSearch, para citar alguns. Estudos de caso de ambos os Nutch e SearchBlox estão incluídas no capítulo 10.

### 1.2.2 O Lucene pode fazer por você

Lucene permite adicionar capacidades de indexação e busca para seus aplicativos (Estas funções são descritos na seção 1.3). Lucene pode indexar e fazer busca poder todos os dados que podem ser convertidos para um formato textual. Como você pode ver na figura 1.5,

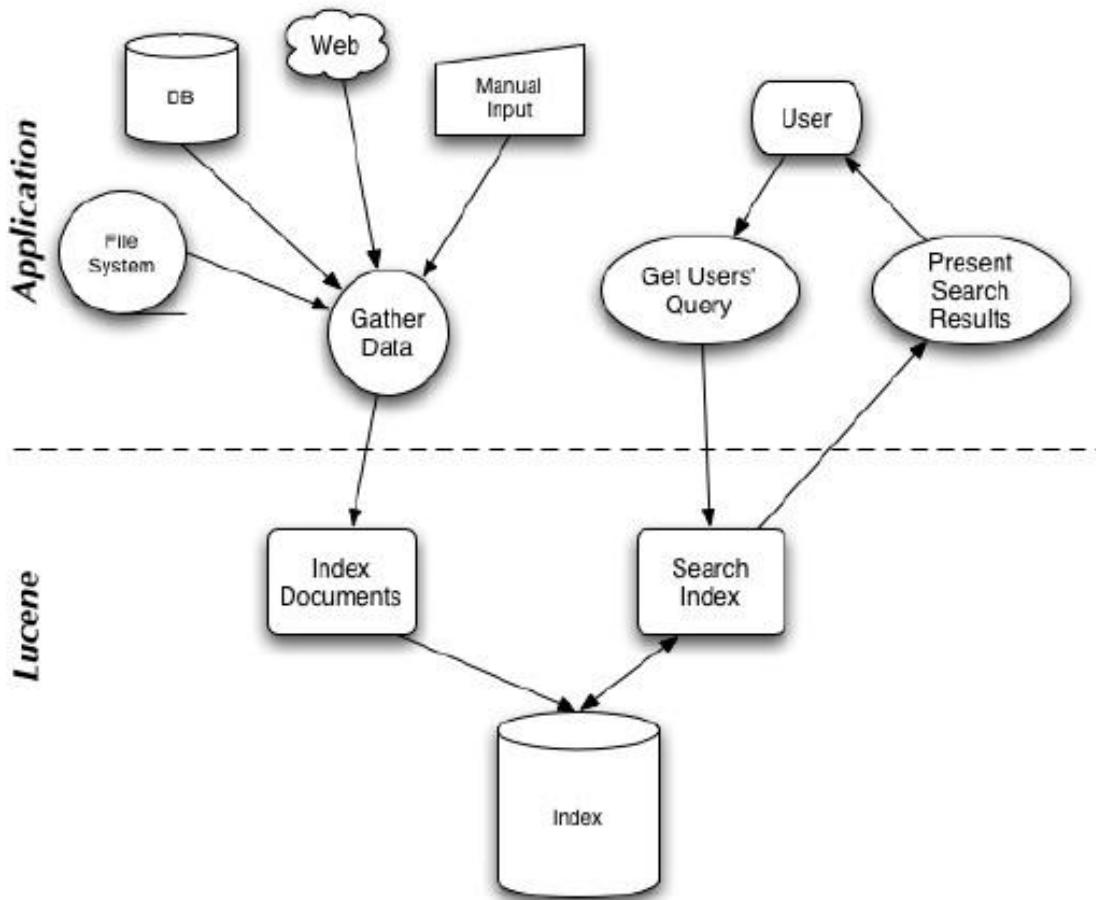


Figura 1.5 A integração de aplicações típicas com Lucene

Lucene não se preocupa com a origem dos dados, seu formato, ou até mesmo sua linguagem, contanto que você possa converter para texto. Isto significa que você pode usar Lucene para indexar e

pesquisa de dados armazenados em arquivos: páginas web em servidores web remotos, documentos armazenados em sistemas de arquivos local, arquivos de texto simples, documentos do Microsoft Word, HTML ou PDF arquivos, ou qualquer outro formato a partir do qual você pode extrair informação textual.

Da mesma forma, com a ajuda do Lucene você pode indexar dados armazenados em seus bancos de dados, dando seus usuários capacidades de pesquisa de texto completo que muitos bancos de dados não fornecem.

Uma vez que você integrar Lucene, os usuários de seus aplicativos podem fazer pesquisas, como + + George Rice-comer pudim de ,Apple-pie + Tiger,animal: Macaco e alimentos: bananaE assim por diante. Com Lucene, você pode indexar e mensagens de e-mail de busca, mailing-list arquivos, chats de mensagens instantâneas, páginas da Wiki ... a lista é longa.

### 1.2.3 História do Lucene

Lucene foi originalmente escrito por Doug Cutting,<sup>2</sup> que foi inicialmente disponível para download de sua casa no site da SourceForge. Ele se juntou ao Apache Software Foundation de Jacarta, de produtos de alta qualidade em código aberto Java em Setembro de 2001. Com cada versão, desde então, o projeto tem tido aumento de visibilidade, atrair mais usuários e desenvolvedores. A partir de julho de 2004, Lucene versão 1.4 foi lançado, com uma versão de correção de bugs 1.4.2 no início de outubro. A Tabela 1.1 mostra a história do lançamento.

Tabela 1.1 Lucene história de liberação

Versão	Data de lançamento	Marcos
0,01	Março 2000	Liberação primeira fonte aberto (SourceForge)
1,0	Outubro 2000	
1.01b	Julho 2001	Última SourceForge liberação
1,2	Junho 2002	Apache Jakarta primeira liberação
1,3	Dezembro 2003	Formato composto índice, melhorias QueryParser, remoto pesquisa, token posicionamento, marcando API extensível
1,4	Julho 2004	Triagem, consultas span, vetores prazo
1.4.1	Agosto 2004	Correção de bug para classificar o desempenho
1.4.2	Outubro 2004	IndexSearcher otimização e misc. correções
1.4.3	Inverno 2004	Misc. correções

#### NOTA

Criador do Lucene, Doug Cutting, tem importantes teóricos e práticos experiência na área de RI. Ele publicou uma série de pesquisas artigos sobre vários temas IR e já trabalhou para empresas como a Excite, Apple, e Grand Central. Mais recentemente, preocupado com a diminuição do número de motores de busca da web e um monopólio potencial em aquele reino, ele criou Nutch, a Web open-source World-Wide primeiro motor de busca (<http://www.nutch.org>), é projetado para lidar com o rastreamento, indexação e pesquisa de vários bilhões de páginas web atualizada frequentemente. Não surpreendentemente, Lucene está no centro de Nutch; seção 10.1 inclui uma estudo de caso de como Nutch aproveita Lucene.

<sup>2</sup> Lucene é o nome do meio é a esposa de Doug, é também o primeiro nome de sua avó materna é.

Corte Doug continua a ser a principal força por trás Lucene, mas as mentes mais brilhantes aderiram ao projeto desde mover Lucene, sob o guarda-chuva Jakarta Apache. No momento da redação deste texto, a equipe do Lucene núcleo inclui cerca de meia dúzia ativos desenvolvedores, dois dos quais são autores deste livro. Além do oficial desenvolvedores de projetos, Lucene tem uma comunidade de usuários bastante grande e ativa técnicos que freqüentemente contribui patches, correções de bugs e novos recursos.

#### 1.2.4 Quem usa o Lucene

Quem não? Além das entidades mencionadas na Powered by Lucene página na Wiki Lucene, uma série de outros grandes, well-known, multinacionais-organizações internacionais estão usando o Lucene. Ele fornece recursos de pesquisa para o Eclipse IDE, a Enciclopédia Britannica CD-ROM/DVD, FedEx, da Clínica Mayo, Hewlett-Packard, New Scientist revista, Epifania, do MIT OpenCourseWare e DSpace, plataforma Akamai EdgeComputing, e assim por diante. Seu nome estará em esta lista em breve, também.

#### 1.2.5 portas Lucene: Perl, Python, C + +, NET, Ruby.

Uma maneira de julgar o sucesso do software de código aberto é pelo número de vezes ele foi portado para outras linguagens de programação. Usando este Lucene, métrica é um sucesso! Embora o Lucene original é escrito em Java, como esta escrito Lucene foi portado para Perl, Python, C + + e NET., E algumas bases tem sido feito para portá-lo para Ruby. Esta é uma excelente notícia para os desenvolvedores que precisam para acessar os índices Lucene de aplicações escritas em diferentes idiomas. Você pode aprender mais sobre algumas dessas portas no capítulo 9.

### 1,3 de indexação e busca

---

No coração de todos os motores de busca é o conceito de indexação: processamento do dados originais em uma pesquisa altamente eficiente de referência cruzada para facilitar a busca rápida. Vamos dar uma olhada rápida de alto nível, tanto de indexação e pesquisar processos.

#### 1.3.1 O que é indexação, e por que é importante?

Suponha que você precisava para procurar um número grande de arquivos, e você queria ser capaz para encontrar arquivos que continham uma determinada palavra ou uma frase. Como você faria para escrever um programa para fazer isso? Uma abordagem ingênuo seria seqüencialmente scan cada arquivo para a dada palavra ou frase. Esta abordagem tem uma série de falhas, o mais óbvio das quais é que ele não escala a conjuntos de arquivo maior ou casos em que os arquivos

são muito grandes. Aqui é onde entra em indexação: Para pesquisar grandes quantidades de texto rapidamente, você deve primeiro índice que texto e convertê-lo em um formato que vai deixá-lo busca-lo rapidamente, eliminando o processo de digitalização lenta seqüencial. Este converso Comissão processo é chamado indexação, e sua saída é chamado de índice.

Você pode pensar de um índice como uma estrutura de dados que permite o acesso rápido aleatório para palavras armazenadas dentro dele. O conceito por trás dele é análogo a um índice no final de um livro, que lhe permite localizar rapidamente páginas que falam sobre determinados temas. No caso do Lucene, um índice é uma estrutura especialmente projetada de dados, normalmente armazenados no sistema de arquivos como um conjunto de arquivos de índice. Nós cobrimos a estrutura dos arquivos de índice em detalhes no Apêndice B, mas por agora basta pensar um índice Lucene como uma ferramenta

### 1.3.2 O que está procurando?

permite pesquisa de palavra rápida. Pesquisa é o processo de procurar palavras em um índice para localizar documentos em que elas aparecem. A qualidade de uma pesquisa é geralmente descrito utilizando precisão e recordar métricas. Medidas lembrar quão bem o sistema de busca encontra relevantes documentos, enquanto que as medidas de precisão quão bem o sistema filtra o irrelevante documentos. No entanto, você deve considerar uma série de outros fatores, quando pensação sobre a pesquisa. Já mencionamos a velocidade ea capacidade de rapidamente busca de grandes quantidades de texto. Suporte para consultas individuais e MultiTerm, frase consultas, curingas ranking resultado e classificação são também importantes, como é um amistoso sintaxe para entrar nessas consultas. Biblioteca Lucene é um poderoso software oferece uma número de recursos de pesquisa, sinos e assobios, tantos que tivemos de spread nossa cobertura de busca mais três capítulos (capítulos 3, 5 e 6).

## 1,4 Lucene em ação: um aplicativo de exemplo

---

Vamos ver o Lucene em ação. Para fazer isso, lembre-se o problema da indexação e busca arquivos ing, que descrevemos no ponto 1.3.1. Além disso, suponha que você precise arquivos de índice e busca armazenados em uma árvore de diretórios, e não apenas em um único diretório. Para mostrar-lhe indexação Lucene e capacidade de procura, nós vamos usar um par de comando aplicações de linha: `Indexador` e `Searcher`. Primeiro vamos índice de uma árvore de diretórios conter- arquivos de texto ing, então vamos procurar o índice criado.

Estas aplicações exemplo irá familiarizá-lo com a API do Lucene, sua facilidade de uso, e seu poder. As listagens de código são completos, prontos para usar linha de comando programas. Se a indexação de arquivos / pesquisa é o problema que você precisa para resolver, então você pode copiar as listagens de código e ajustá-los para atender às suas necessidades. Nos capítulos que se seguem, vamos descrever cada aspecto do uso do Lucene com muito mais detalhes.

Antes de podermos busca com Lucene, precisamos construir um índice, por isso começamos com nosso Indexador aplicação.

#### 1.4.1 Criar um índice

Nesta seção, você verá uma única classe chamada `Indexador` e seus quatro métodos estáticos; juntos, eles recursivamente percorrer diretórios de arquivos do sistema e indexar todos os arquivos com um

.Txt. Quando `Indexador` conclui a execução deixa para trás uma Lucene índice para o seu irmão, `Searcher` (Apresentado na seção 1.4.2).

Nós não esperamos que você esteja familiarizado com as classes Lucene poucos e métodos usado neste exemplo Que nós explicar-lhes logo. Após a listagem de código anotado, vamos mostrar-lhe como usar `IndexadorE`, se ele o ajuda a aprender `Indexador` é usado antes você ver como ele é codificado, vá diretamente para a discussão de uso que se segue o código.

#### Usando indexador de arquivos de texto índice

Listagem 1.1 mostra o `Indexador` de linha de comando do programa. Ela recebe dois argumentos:

- Um caminho para um diretório onde armazenar o índice Lucene
- Um caminho para um diretório que contém os arquivos que deseja indexar

Listagem 1.1 Indexador: atravessa um sistema de arquivos e índices. txt

```

/*
 * Este código foi escrito originalmente para
 * Erik Lucene intro java.net artigo
 */
Indexador public class {

    public static void main (String [] args) throws Exception {
        if (args.length! = 2) {
            throw new Exception ("Uso: java" + Indexer.class.getName ()
                + "<index Dir> <data dir>");
        }
        indexDir (args [0]);
        Arquivo = new File datadir (args [1]);
        longo start = new Date () getTime ();
        int numIndexed = índice (indexDir, datadir);
        final longo = new Date () getTime ();

        System.out.println ("Indexação" + numIndexed + "arquivos tomou"
            + (Final - inicial) + "ms");
    }

    / / Abre um índice de arquivo e começar de passagem de diretório
    índice public static int (File indexDir, Arquivo datadir)
        throws IOException {

```

```

if (dataDir.exists () || ! dataDir.isDirectory ()) {
    throw IOException novo (datadir
        + "Não existe ou não é um diretório");
}

Escritor IndexWriter IndexWriter = new (indexDir,
    nova StandardAnalyzer (), true);
writer.setUseCompoundFile (false);

indexDirectory (escritor, datadir);

int numIndexed writer.docCount = ();
writer.optimize ();
writer.Close ();
    Fechar
    retorno numIndexed;
    índice
}

/ / Método recursivo que chama a si mesmo quando encontra um diretório
indexDirectory private static void (escritor IndexWriter, File dir)
throws IOException {

    File [] files = dir.listFiles ();

    for (int i = 0; i < files.length; i++) {
        File f = arquivos [i];
        if (f.isDirectory ()) {
            indexDirectory (escritor, f); Recurso
        } Else if (f.getName ().EndsWith ("CTxt")) {
            indexFile (escritor, f);
        }
    }
}

/ / Método para realmente indexar um arquivo usando o Lucene
indexFile private static void (escritor IndexWriter, f File)
throws IOException {

    if (f.isHidden () || ! f.exists () || ! f.canRead ()) {
        retorno;
    }

    System.out.println ("Indexação" + f.getCanonicalPath ());

    Documento doc = new Document ();
    doc.add (Field.Text ("conteúdo", new FileReader (f)));
        Arquivo f
        conteúdo
    doc.add (Field.Keyword ("filename", f.getCanonicalPath ())); Índice
        nome do arquivowriter.addDocument (doc); Adicionar documento f Lucene
    } para índice}

```

b Criar Lucene índice

Índice. Txt apenas os arquivos

d

Curiosamente, a maior parte do código executa recursiva de passagem de diretório (C).

Apenas a criação e fechamento das `IndexWriter` (b) E quatro linhas no

`indexFile` método (d,e,f) De `Indexador` envolvem o Lucene-API efetivamente seis linhas de código.

Este exemplo intencionalmente concentra-se em arquivos de texto com. Extensões txt para manter

coisas simples ao mesmo tempo demonstrando o uso do Lucene e poder. No capítulo 7, vamos mostrar-lhe como lidar com arquivos não-texto, e vamos desenvolver um ready-to-use pequenos quadros

trabalho capaz de análise e indexação de documentos em vários formatos comuns.

`Indexador` de execução

Na linha de comando, corremos `Indexador` contra um diretório local de trabalho incluindo o código-fonte do próprio Lucene. Nós instruído `Indexador` para arquivos de índice em o / Lucene diretório e armazenar o índice Lucene na construir / index diretório:

```
% Lia.meetlucene.Indexer java build / index / lucene

Lucene indexação / / build / teste / TestDoc / test.txt
Indexação / lucene/build/test/TestDoc/test2.txt
Lucene indexação / / build.txt
Lucene indexação / / changes.txt
Indexação / lucene / LICENSE.txt
Indexação / lucene / README.txt
Indexação / lucene / src / jsp / README.txt
Indexação / lucene / src / test / org / apache / lucene / análise / ru /
⇒ stemsUnicode.txt
Indexação / lucene/src/test/org/apache/lucene/analysis/ru/test1251.txt
Indexação / lucene/src/test/org/apache/lucene/analysis/ru/testKOI8.txt
Indexação / lucene / src / test / org / apache / lucene / análise / ru /
⇒ testUnicode.txt
Indexação / lucene / src / test / org / apache / lucene / análise / ru /
⇒ wordsUnicode.txt
Lucene indexação / / todo.txt
Indexação 13 imagens levou 2.205 milissegundos
```

`Indexador` imprime os nomes de arquivos que os índices, assim você pode ver que ele indexa apenas os arquivos com a extensão txt..

**NOTA** Se você estiver executando este aplicativo em um shell de comando da plataforma Windows,  
você precisa ajustar separadores de linha de comando do diretório e caminho.  
A linha de comando do Windows é `java build / index c: \ lucene`.

Quando ele completar indexação, `Indexador` imprime o número de arquivos que ele indexados e o tempo que levou para fazê-lo. Porque o tempo reportado inclui tanto o arquivo diretório traversal e indexação, você não deve considerá-la uma medida de performance oficial.

No nosso exemplo, cada um dos arquivos indexados era pequena, mas cerca de dois segundos para  
puxando um índice de arquivos de texto é razoavelmente impressionante.

Velocidade de indexação é uma preocupação, e vamos ver isso no capítulo 2. Mas, geralmente, pesquisa é de importância ainda maior.

#### 1.4.2 Pesquisando um índice

Pesquisando no Lucene é tão rápido e simples como a indexação, o poder desta função-alidade é surpreendente, como os capítulos 3 e 5 irá mostrar-lhe. Por agora, vamos olhar para Searcher, Um programa de linha de comando que vamos usar para pesquisar o índice criado por Indexador. (Tenha em mente que o nosso Searcher serve o propósito de demonstrar o uso da API do Lucene pesquisa. Seu aplicativo de pesquisa também pode assumir a forma de uma aplicação web ou desktop com uma interface gráfica, um EJB, e assim por diante.)

Na seção anterior, indexamos um diretório de arquivos de texto. O índice, neste exemplo, reside em um diretório de seu próprio sistema de arquivos. Nós instruído Indexador para criar um índice Lucene em um diretório build / índice, em relação à direção-história de que nós invocada Indexador. Como você viu na listagem 1.1, este índice contém os arquivos indexados e seus caminhos absolutos. Agora precisamos usar Lucene para pesquisar

que o índice a fim de encontrar arquivos que contenham uma parte específica do texto. Por exemplo,  
podemos querer encontrar todos os arquivos que contêm a palavra-chave Java ou lucene, ou podemos  
quero encontrar arquivos que contenham a frase "Sistema requisitos".

#### Usando Searcher para implementar uma busca

O Searcher programa complementa Indexador e fornece linha de comando  
pesquisar capacidade. Listagem 1.2 mostra Searcher na sua totalidade. São precisos dois  
de linha de comando argumentos:

- O caminho para o índice criado com Indexador
- A consulta para usar para pesquisar o índice

Listagem 1.2 Pesquisa: Buscas um índice Lucene para uma consulta passada como um argumento

```
/ **  
 * Este código foi escrito originalmente para  
 * Erik Lucene intro java.net artigo  
 * /  
 public class {Searcher  
  
    public static void main (String [] args) throws Exception {  
        if (args.length! = 2) {  
            throw new Exception ("Uso: java" + Searcher.class.getName ()  
                + "<query> Dir <index>");  
    }  
}
```

```

Arquivo = new File indexDir (args [0]);
Q = String args [1];String de consulta

if (indexDir.exists () !! | indexDir.isDirectory ()) {
    throw new Exception (indexDir +
        "Não existe ou não é um diretório.");
}

pesquisa (indexDir, q);
}

pesquisa public static void (File indexDir, String q)
throws Exception {
    Directory fsDir = FSDirectory.getDirectory (indexDir, false);
    IndexSearcher is = new IndexSearcher (fsDir);Índice Open
    Query = QueryParser.parse (q, "conteúdo",
        nova StandardAnalyzer ());
    longo start = new Date () getTime ();
    Hits hits = is.search (query);Índice de pesquisa
    final longo = new Date () getTime ()
    System.err.println ("found" hits.length + () +
        "Documento (s) (em" + (final - inicial) +
        "Milissegundos) que consulta combinados '" +
        q + "':");

    for (int i = 0; i < hits.length (); i++) {
        Documento doc = hits.doc (i);
        System.out.println (doc.get ("filename"));
    }
}

```

Diretório índice  
criado por indexador

b

c Analisar consulta

d Escreva pesquisa  
stats

e Recuperar documento correspondente  
Mostrar  
nome do arquivo

Searcher, Como seu Indexador irmão, tem apenas algumas linhas de código de lidar com Lucene. Um par de coisas especiais ocorrem no pesquisa método,

- b Usamos Lucene IndexSearcher e FSDirectory classes para abrir o nosso índice de busca.
- c Usamos QueryParser analisar uma consulta legível em Lucene Pergunta classe.
- d Pesquisando hits retorna na forma de um Hits objeto.
- e Note que o Hits objeto contém apenas referências aos documentos subjacentes.
- e Em outras palavras, em vez de ser carregado imediatamente após a pesquisa, as partidas são carregado a partir do índice de um preguiçoso moda-somente quando solicitado com o hits. doc (int) chamada.

### Searcher execução

Vamos correr Searcher e encontrar alguns documentos em nosso índice usando a consulta "Lucene":

```
% Java lia.meetlucene.Searcher build / index 'lucene'

Encontrados seis documento (s) (em 66 milisegundos) que combinava
=> consulta 'lucene':
/ Lucene / README.txt
/ Lucene / src / jsp / README.txt
/ Lucene / build.txt
/ Lucene / todo.txt
/ Lucene / LICENSE.txt
/ Lucene / changes.txt
```

A saída mostra que seis dos 13 documentos que indexados com Indexador conter a palavra lucene e que a pesquisa tomou um magro 66 milissegundos. Porque Indexador caminhos armazena arquivos "absoluta no índice, Searcher pode imprimi-los. É interessante notar que armazenar o caminho do arquivo como um campo foi a nossa decisão e aprofundou, neste caso, mas a partir da perspectiva do Lucene é arbitrária meta-dados anexado de documentos indexados.

Claro, você pode usar consultas mais sofisticadas, como "Lucene e Doug ' ou "Lucene E NÃO lenta ' ou '+ Lucene + livro "E assim por diante. Capítulos 3, 5 e 6 cobrir todos os aspectos diferentes de pesquisa, incluindo a sintaxe de consulta do Lucene.

### Usando o utilitário xargs

O Searcher classe é uma demonstração simplista das características do Lucene pesquisa. Como tal, dumps corresponde apenas para a saída padrão. No entanto, Searcher tem mais uma truque na manga. Imagine que você precisa para encontrar arquivos que contenham uma certa chave palavra ou frase, e então você deseja processar os arquivos correspondentes de alguma forma. Para manter as coisas simples, vamos imaginar que você deseja listar cada arquivo que combinem usando o ls Comando UNIX, talvez para ver o tamanho do arquivo, os bits de permissão, ou proprietário. Por ter caminhos documento correspondente escrita sem adornos para a saída padrão, e ter a saída estatística escrita para o erro padrão, você pode usar o nifty UNIX Java utilitário para processar os arquivos correspondentes, como mostrado aqui:

```
=> "Lucene E NÃO lenta '| xargs ls-l
```

```
Encontrados seis documento (s) (em 131 milissegundos) que
=> combinado consulta 'lucene E NÃO lenta':
-Rw-r - r - 1 erik staff4215 10 de setembro 21:51 / lucene / build.txt
-Rw-r - r - 1 erik pessoal 17.889 28 de dezembro 10:53 / lucene / changes.txt
-Rw-r - r - 1 erik staff2670 4 de novembro de 2001 / lucene / LICENSE.txt
```

```
-Rw-r - r - 1 erik    pessoal    683 04 de novembro 2001 / lucene / README.txt
-Rw-r - r - 1 erik    pessoal    370 26 de janeiro 2002 / lucene / src / jsp /
=> README.txt
-Rw-r - r - 1 erik    pessoal    943 18 de setembro 21:27 / lucene / todo.txt
```

Neste exemplo, optamos a consulta Boolean "Lucene E NÃO lenta", Que encontra todos os arquivos que contenham a palavra lucene e não contêm a palavra lento. Esta consulta levou 131 milissegundos e encontrou 6 arquivos correspondentes. Nós encanada SearcherOutput's para

O `xargs` comando, que por sua vez, usou o `ls -l` comando para listar cada partida- arquivo ing. De forma semelhante, os arquivos pareados pode ser copiado, concatenadas, enviado, ou jogados a norma output.<sup>3</sup>

Nosso exemplo de indexação e busca demonstrar aplicações Lucene em um muito de sua glória. Seu uso da API é simples e discreto. A maior parte do código (e isso se aplica a todas as aplicações interagindo com Lucene) é encanamento relativos à a finalidade nos negócios neste caso, Indexador'S rastreador sistema de arquivos que procura arquivos de texto e Searcher'S que imprime código de filenames combinado com base em uma consulta para na saída padrão. Mas não deixe que este fato, ou a concisão dos exemplos, tentá-lo a complacência: Há muita coisa acontecendo nos bastidores do Lucene, e nós temos utilizado bastante algumas das melhores práticas que vêm com a experiência. A efivamente Lucene alavancagem, é importante entender mais sobre como funciona e como estendê-lo quando necessário. O restante deste livro é dedicado a dar-lhe essas peças que faltam.

## 1,5 Compreender as classes de indexação do núcleo

Como você viu em nosso `Indexador` classe, você precisa as seguintes classes para realizar a procedimento mais simples de indexação:

- `IndexWriter`
- `Diretório`
- `Analizador`
- `Documento`
- `Campo`
- 

O que se segue é um breve resumo dessas classes, para lhe dar uma idéia aproximada sobre seu papel na Lucene. Vamos usar essas classes ao longo deste livro.

---

<sup>3</sup> Neal Stephenson detalhes esse processo muito bem em "In the Beginning Was a linha de comando": <http://www.cryptonomicon.com / beginning.html>.

### 1.5.1 IndexWriter

`IndexWriter` é o componente central do processo de indexação. Esta classe cria um novo índice e adiciona documentos a um índice existente. Você pode pensar em `Índice Escritor` como um objeto que lhe dá acesso de gravação para o índice, mas não permitem que você leia ou busca-lo. Apesar do nome, `IndexWriter` não é a única classe que é usado para modificar um índice; seção 2.2 descreve como usar a API Lucene para modificar um índice.

### 1.5.2 Directory

O `Diretório` classe representa a localização de um índice do Lucene. É um resumo classe que permite que suas subclasses (dois dos quais estão incluídos no Lucene) para armazenar o índice como entenderem. Em nossa `Indexador` exemplo, usamos um caminho para um sistema de arquivos real diretório para obter uma instância de `Diretório`, Que passamos a `IndexWriter`'S construtor. `IndexWriter` e depois usou uma do concreto `Diretório` implementações, `FSDirectory`, E criou o nosso índice em um diretório no sistema de arquivos.

Em seus aplicativos, você provavelmente será armazenar um índice Lucene em um disco. Para fazer isso, use `FSDirectory`, Um `Diretório` subclasse que mantém uma lista de arquivos real no sistema de arquivos, como fizemos em `Indexador`.

A implementação de outras `Diretório` é uma classe chamada `RAMDirectory`. Embora ele expõe uma interface idêntica à do `FSDirectory`, `RAMDirectory` mantém todos os seus dados na memória. Esta aplicação é, portanto, útil para os pequenos índices que podem ser totalmente carregado na memória e podem ser destruídos após o término de um aplicativo. Porque todos os dados são mantidos na memória de acesso rápido e não em um ritmo mais lento disco rígido, `RAMDirectory` é adequado para situações onde você precisa

acesso muito rápido para o índice, se durante a indexação ou pesquisa. Para exemplo, os desenvolvedores do Lucene fazem uso extensivo de `RAMDirectory` em todas as suas

testes de unidade: Quando um teste é executado, uma rápida índice na memória é criado ou pesquisado, e

quando um teste é concluído, o índice é automaticamente destruído, não deixando resíduos no disco. Claro, a diferença de desempenho entre `RAMDirectory` e `FSDirectory` é menos visível quando Lucene é usado em sistemas operacionais que cache arquivos na memória. Você vai ver os dois `Diretório` implementações usado no código-snippet animais de estimativa neste livro.

### 1.5.3 Analyzer

Antes do texto é indexado, é passado através de um `Analisador`. O `Analisador`, Especificado no `IndexWriter` construtor, é responsável pela extração de tokens de texto a ser indexados e eliminando o resto. Se o conteúdo a ser indexado não é texto puro, que deve primeiro ser convertido para ele, como mostrado na figura 2.1. Capítulo 7 mostra como

extraír o texto das mais comuns formatos de rich-media documento. Analisador é uma classe abstrata, mas Lucene vem com várias implementações do mesmo. Alguns dos os a lidar com saltos parar as palavras (Palavras usadas com freqüência que não ajudam a distin-

guir um documento da outra, como um, um, a, em, e por diante); algum acordo com conversão de símbolos para letras minúsculas, de modo que as pesquisas não são case-sensitive;

e assim por diante. Analisadores são uma parte importante do Lucene e pode ser usado por muito

mais do que a entrada simples de filtragem. Para um desenvolvedor integrar Lucene em um aplicação, a escolha do analisador (s) é um elemento crítico do projeto do aplicativo.

Você vai aprender muito mais sobre eles no capítulo 4.

#### 1.5.4 Documento

A `Documento` representa uma coleção de campos. Você pode pensar nisso como um virtual documento, um bloco de dados, tais como uma página web, uma mensagem de e-mail, ou um arquivo de texto

que você quer fazer recuperáveis em um momento posterior. Campos de um documento representam

o documento ou meta-dados associados a esse documento. A fonte original (Como um registro de banco de dados, um documento do Word, um capítulo de um livro, e assim por diante)

dos dados do documento é irrelevante para Lucene. A meta-dados como autor, título, assunto, data modificada, e assim por diante, são indexados e armazenados separadamente como campos de

um `Documento`. NOTA Quando nos referimos a um documento neste livro, queremos dizer um Microsoft Word, RTF, PDF, ou outro tipo de documento; não estamos falando do Lucene `Documento` classe. Note a distinção no caso e tipo de letra.

Lucene trata apenas de texto. Núcleo do Lucene não se lidar com qualquer coisa, mas `java.lang.String` e `java.io.Reader`. Embora vários tipos de documentos podem ser indexados e fez `searchable`, processá-los não é tão simples como pró-processamento de conteúdo puramente textual que pode ser facilmente convertida em uma `Corda` ou `Leitor`

Tipo de Java. Você aprenderá mais sobre a manipulação de documentos não-texto no capítulo 7.

Em nossa `Indexador`, Nós estamos preocupados com arquivos de texto de indexação. Assim, para cada arquivo de texto

encontramos, criamos uma nova instância do `Documento` classe, preenchê-lo com `Campos` (Descrito a seguir), e acrescentar que `Documento` para o índice, de forma eficaz o arquivo de indexação.

#### 1.5.5 Campo

Cada `Documento` em um índice contém um ou mais campos com nome, consubstanciado num classe chamada `Campo`. Cada campo corresponde a um pedaço de dados que seja consultado contra ou recuperado do índice durante a pesquisa.

Lucene oferece quatro tipos diferentes de campos a partir do qual você pode escolher:

- Palavra chave T-Isn't analisadas, mas é indexado e armazenado no índice verbatim. Este tipo é apropriado para campos cujo valor original deve ser preservada em sua totalidade, como URLs, os caminhos do sistema de arquivos, datas, nomes pessoais, Social Números de segurança, números de telefone, e assim por diante. Por exemplo, usamos o caminho do sistema de arquivo no Indexador (Listagem 1.1) como um Palavra chave de campo.
- UnIndexed Não é nem analisado, nem indexados, mas seu valor é armazenado no índice como é. Este tipo é apropriado para os campos que você precisa para mostrar com resultados da pesquisa (tais como uma chave de URL ou banco de dados primário), mas cujos valores você nunca vai procurar diretamente. Como o valor original de um campo deste tipo é armazenado no índice, este tipo não é adequado para o armazenamento de campos com grande quantidade de texto que não precisa ser recuperada na sua forma original, tais como corpos de páginas web, ou qualquer outro tipo de documento de texto.
- Unstored O oposto de UnIndexed. Este tipo de campo é analisada e indexados, mas não é armazenado no índice. É apropriado para a indexação de um grande campo que não precisa ser recuperada na sua forma original, tais como corpos de páginas web, ou qualquer outro tipo de documento de texto.
- Texto É analisado, e é indexada. Isto implica que os campos deste tipo podem ser pesquisados contra, mas ser cauteloso sobre o tamanho do campo. Se os dados indexado é uma Corda, Também é armazenada, mas se os dados (como em nosso Indexador exemplo) é de um Leitor, Não é armazenada. Isso é muitas vezes fonte de confusão, de modo tomar nota desta diferença ao usar Field.Text.

Todos os campos consistem de um nome e um par de valores. Que tipo de campo você deve usar depende de como você deseja usar o campo e seus valores. Estritamente falando, Lucene tem uma única Campo Tipo: Os campos são distinguidos uns dos outros com base sobre suas características. Alguns são analisados, mas os outros não são, alguns são indexadas, enquanto outros são armazenadas textualmente, e assim por diante.

A Tabela 1.2 fornece um resumo das características do campo diferente, mostrando-lhe como os campos são criados, juntamente com exemplos de uso comum.

Tabela 1.2 Uma visão geral dos tipos de campos diferentes, as suas características, e seu uso

Método de campo / tipo	Analisado	Indexados	Armazenadas	Exemplo de uso
Field.Keyword (String, String)		✓	✓	Telefone e Segurança Social números, URLs, nomes de pessoas Datas
Field.Keyword (Date, String)				
Field.UnIndexed (String, String)			✓	Tipo de documento (PDF, HTML, e assim on), se não for usado como um critério de pesquisa

continua na página seguinte

Tabela 1.2 Uma visão geral dos tipos de campos diferentes, as suas características, e seu uso (Continuação)

Método de campo / tipo	Analisado	Indexados	Armazenadas	Exemplo de uso
Field.UnStored (String, String)	✓	✓		Títulos de documentos e conteúdo
Field.Text (String, String)	✓	✓	✓	Títulos de documentos e conteúdo
Field.Text (String, Reader)	✓	✓		Títulos de documentos e conteúdo

Repare que todos os tipos de campos podem ser construídos com dois `Strings` que representam a nome do campo e seu valor. Além disso, uma `Palavra chave` campo pode ser passado tanto um `Corda` e um `Data` objeto, eo `Texto` campo aceita um `Leitor` objeto, além de o `Corda`. Em todos os casos, o valor é convertido para um `Leitor` antes da indexação; esses existem métodos adicionais para fornecer uma API amigável.

## NOTA

Note a distinção entre `Field.Text (String, String)` e De campo. De texto (`String, Reader`). O `Corda` variante lojas os dados de campo, enquanto o `Leitor` variante não. Para indexar uma `Corda`, Mas não armazená-la, use `Field.UnStored (String, String)`.

Finalmente, `UnStored` e `Texto` campos podem ser usados para criar vetores prazo (Um avançado tópico, tratados na secção 5.7). Para instruir Lucene para criar vetores termo para um dado `UnStored` ou `Texto` campo, você pode usar `Field.UnStored (String, String, true)`, `Field.Text (String, String, true)` Ou `Field.Text (String, Reader, true)`. Você vai aplicar este punhado de classes mais frequentemente quando se utiliza o Lucene para índice ing. Para implementar a funcionalidade de pesquisa básica, você precisa estar familiarizado com um conjunto tão pequeno e simples de classes de busca Lucene.

## 1.6 Compreender as classes em busca core

A interface de pesquisa básica que fornece Lucene é tão simples como o para indexação. Apenas algumas classes são necessárias para executar a operação de pesquisa básica:

- `IndexSearcher`
- `Prazo`
- `Pergunta`
- `TermQuery`
- `Hits`

As seguintes seções fornecem uma breve introdução a essas classes. Nós vamos expandir sobre essas explicações nos capítulos que se seguem, antes de mergulharmos mais tópicos avançados.

### 1.6.1 IndexSearcher

`IndexSearcher` é o que busca `IndexWriter` é a indexação: o elo central de o índice que expõe vários métodos de pesquisa. Você pode pensar em `IndexSearcher` como uma classe que abre um índice em um modo somente leitura. Ele oferece uma série de métodos, alguns dos quais são implementados em sua classe pai abstrato `Searcher`; mais simples tem uma única `Pergunta` objeto como um parâmetro e retorna um `Hits` objeto. Um uso típico deste método parecido com este:

```
IndexSearcher is = new IndexSearcher (   
    FSDirectory.getDirectory ("tmp // index", false));  
Query q = new TermQuery (Termo de novo ("conteúdo", "lucene"));  
Hits hits = is.search (q);
```

Nós cobrimos os detalhes da `IndexSearcher` no capítulo 3, juntamente com a mais avançada informações nos capítulos 5 e 6.

### 1.6.2 Prazo

A `Prazo` é a unidade básica para a pesquisa. Semelhante ao `Campo` objeto, ele consiste em um par de elementos string: o nome do campo eo valor desse campo. Nota que `Prazo` objetos também estão envolvidos no processo de indexação. No entanto, eles são criados por `internals Lucene`, então você normalmente não precisa pensar sobre eles, enquanto indexação. Durante a pesquisa, você pode construir `Prazo` objetos e usá-los juntamente com `TermQuery`:

```
Query q = new TermQuery (novo Prazo ("Conteúdo", "lucene"));  
Hits hits = is.search (q);
```

Este código instrui Lucene para encontrar todos os documentos que contenham a palavra `lucene` em um campo chamado `conteúdo`. Porque o `TermQuery` objeto é derivado do abstrato classe pai `Pergunta`, Você pode usar o `Pergunta` tipo no lado esquerdo da instrução.

### 1.6.3 Consulta

Lucene vem com uma série de concreto `Pergunta` subclasses. Até agora neste capítulo mencionamos apenas Lucene o mais básico `Pergunta:TermQuery`. Outros `Pergunta` tipos são `BooleanQuery`, `PhraseQuery`, `PrefixQuery`, `PhrasePrefixQuery`, `RangeQuery`, `FilteredQuery` E `SpanQuery`. Todos estes são cobertos no capítulo 3. `Pergunta` é o comum, classe pai abstrato. Ele contém métodos utilitários diversos, a maioria da interessante do que é `setBoost (float)`, Descrito na seção 3.5.9.

#### 1.6.4 TermQuery

TermQuery é o tipo mais básico de consulta suportado pelo Lucene, e é um dos o primitivo tipos de consulta. É usado para documentos correspondentes que contêm campos com valores específicos, como você viu no últimos parágrafos.

#### 1.6.5 Hits

O Hits classe é um container simples de ponteiros para classificação de resultados de pesquisa-  
docu-  
mentos que correspondem a uma determinada consulta. Por motivos de desempenho, Hits  
casos não  
carga a partir do índice de todos os documentos que correspondem a uma consulta, mas apenas  
uma pequena parte

### 1.7 Análise de produtos de busca alternativo

---

Antes de selecionar sua biblioteca Lucene como IR de escolha, você pode querer rever outras soluções no mesmo domínio. Fizemos algumas pesquisas sobre alternativas protos que você pode querer considerar e avaliar, esta seção resume os nossos descobertas. Nós grupo desses produtos em duas categorias principais:

- Recuperação de bibliotecas informações
- Indexação e busca aplicações

O primeiro grupo é menor, que consiste de indexação de texto completo e bibliotecas em busca semelhante ao Lucene. Os produtos deste grupo permitem que você incorporá-las na sua aplicação, conforme mostrado anteriormente na figura 1.5.

O grupo, segundo maior é composta de ready-to-use indexação e busca software. Este software é normalmente concebidos para indexar e pesquisar um determinado tipo de dados, tais como páginas web, e é menos flexível do que software no primeiro grupo. No entanto, alguns destes produtos também expor sua API de nível inferior, assim você pode às vezes usá-los como bibliotecas IR também.

#### 1.7.1 bibliotecas IR

Em nossa pesquisa para este capítulo, encontramos dois IR bibliotecas e-Egothor Xapian-que oferecem um conjunto comparável de características e destinam-se a aproximadamente o mesmo público: os desenvolvedores. Encontramos também MG4J, que não é uma biblioteca de IR, mas é antes um conjunto de ferramentas úteis para a construção de uma biblioteca de IR; pensamos desenvolvedores trabalho Egothor IR deveria saber sobre isso. Aqui estão as nossas opiniões de todos os três produtos. A indexação de texto completo e procurar Java biblioteca, Egothor usa algoritmos core que são muito semelhantes aos utilizados pelo Lucene. Tem sido a existência de várias

anos e tem um desenvolvedor pequeno mas ativo e comunidade de usuários. O desenvolvimento de chumbo

oper é desenvolvedor Checa Galambos Leo, um estudante de PhD com uma acadêmica sólida de fundo em matéria de IR. Às vezes ele participa de usuário e Lucene desenvolvedor discussões mailing list.

Egothor suporta um modelo estendido Boolean, o que lhe permite funcionar como tanto o modelo puro Boolean eo modelo vetorial. Você pode ajustar qual o modelo de usar através de um parâmetro de consulta em tempo simples. Este software apresenta uma série de diferentes tipos de consulta, suporta sintaxe de pesquisa similar, e permite que vários segmentos consulta, que pode vir a calhar se você está trabalhando em um computador multi-CPU ou pesquisar índices remoto.

A distribuição vem com vários Egothor ready-to-use aplicações, tais como um web crawler chamado Capek, um indexador de arquivos com uma interface gráfica Swing, e muito mais. Ele também parsers fornece para vários formatos de texto rico documento, como PDF e Documentos do Microsoft Word. Como tal, Egothor e Capek são comparáveis aos Lucene / Nutch combinação, e indexador de arquivos Egothor e analisadores documento são semelhantes para a análise de documentos e pequenos indexação quadro apresentado em capítulo 7 deste livro.

Livre, open source e liberado sob uma licença BSD-like, o projeto Egothor é comparável ao Lucene, na maioria dos aspectos. Se você ainda tem que escolher um texto completo indexação e busca de biblioteca, você pode querer avaliar Egothor além de Lucene. Página Egothor de casa está em <http://www.egothor.org/>; como do presente escrito, é possui um demo de seu web crawler e funcionalidade de pesquisa.

### Xapian

Xapian é uma biblioteca de recuperação de informação probabilística escrito em C ++ e lançado sob a GPL. Este projeto (ou melhor, os seus antecessores) tem um interessante História: A empresa que desenvolveu e propriedade passou por mais da metade uma dúzia de aquisições, mudanças de nome, muda de foco, e tal.

Xapian é desenvolvido ativamente software. É atualmente na versão 0.8.3, mas tem uma longa história por trás dele e é baseado em décadas de experiência na área de RI. Sua site web, <http://www.xapian.org/>, mostra que tem um rico conjunto de recursos, bem como Lucene. Ele suporta uma ampla gama de consultas e tem um analisador de consulta que suporta humano-friendly sintaxe de pesquisa; derivações com base em Snowball Dr. Martin Porter projeto; parsers para vários tipos de documento rico; ligações para Perl, Python, PHP e (em breve) Java; em busca de índice remoto;; E assim por diante,

Além de fornecer uma biblioteca de IR, Xapian vem com uma pesquisa do site web aplicativo chamado Omega, que você pode fazer o download separadamente.

## MG4J

Embora MG4J (Managing Gigabytes para Java) não é uma biblioteca de IR como Lucene, Egothor e Xapian, acreditamos que cada engenheiro de software a leitura deste livro deve estar ciente disso porque fornece suporte de baixo nível para a construção de Java IR bibliotecas. MG4J o nome de um livro IR popular, Gerenciando Gigabytes: Compressing e indexação de documentos e imagens, escrito por Ian H. Witten, Alistair Moffat, e Timothy C. Bell. Depois de coletar grandes quantidades de dados web com seus distribuídos, web crawler tolerante a falhas chamado UbiCrawler, seus autores necessário software capaz de analisar os dados coletados; de que precisam, MG4J nasceu.

A biblioteca fornece classes otimizado para manipulação de I / O índice invertido compressão, e muito mais. A home page do projeto está em <http://mg4j.dsi.unimi.it/>, o biblioteca é livre, open source, disponibilizado sob LGPL, e atualmente na versão 0.8.2.

### 1.7.2 Indexação e aplicações em busca

O outro grupo de software disponíveis, tanto gratuitos e comerciais, é montado em produtos pré-embalados. Esse tipo de software geralmente não expõe muito de sua API e não requer que você construa um aplicativo personalizado em cima dela. A maior parte deste software expõe um mecanismo que permite controlar um conjunto limitado de parâmetros mas não o suficiente para usar o software de uma maneira que é drasticamente diferente do seu assumido uso. (Para ser justo, há notáveis exceções a esta regra.)

Como tal, não podemos comparar este software para Lucene diretamente. No entanto, alguns dos estes produtos podem ser suficientes para suas necessidades e deixá-lo começar executando rapidamente, mesmo que Lucene ou alguma biblioteca IR outros acaba por ser uma escolha melhor no longo prazo. Aqui está uma pequena lista de diversos produtos populares nesta categoria:

- SWISH, SWISH-E, e SWISH + <http://homepage.mac.com/pauljlucas/software/swish/>, <http://swish-e.org/>
- Vislumbrar e WebGlimpse-<http://webglimpse.net/>
- Namazu-<http://www.namazu.org/>
- ht-<http://www.htdig.org/>
- Colheita e colheita-NG-Http: <http://www.sourceforge.net/projects/harvest/>, <http://webharvest.sourceforge.net/> ng /
- Microsoft Index Server-<http://www.microsoft.com/NTServer/techresources/webserv/IndxServ.asp>
- Verity-<http://www.verity.com/>

### 1.7.3 Recursos on-line

As seções anteriores fornecem apenas um breve panorama dos produtos relacionados. Seus recursos federais vai ajudar você a encontrar outras bibliotecas IR e produtos além daqueles mencionamos:

- DMOZ-No DMOZ Open Directory Project (ODP), você encontrará [http://dmoz.org / Computadores / Software / Information\\_Retrieval /](http://dmoz.org / Computadores / Software / Information_Retrieval /) e todos os seus subcategorias muito informativo.
- Google-Eembora Google Directory é baseado em dados do Open Directory, as duas listas não são diferentes. Assim, você também deve visitar [http://directory.google.com.br / Top / Computadores / Software / Information\\_Retrieval /](http://directory.google.com.br / Top / Computadores / Software / Information_Retrieval /).
- Searchtools-Há é um site web dedicado à busca de ferramentas em <http://www.searchtools.com />. Este site não é sempre atualizado, mas tem sido em torno de por anos e é bastante abrangente. Software é classificado pela operação sistema de linguagem de programação, licenças, e assim por diante. Se você está interessado apenas em software de busca escrito em Java, visite <http://www.searchtools.com/tools / ferramentas-java.html>.

Nós fornecemos comentários positivos de algumas alternativas para o Lucene, mas estamos confi-dente que a sua lição de casa requisito vai levar você a Lucene como a melhor escolha!

## 1,8 Resumo

---

Neste capítulo, você ganhou algum conhecimento básico Lucene. Você já sabe que Lucene é uma biblioteca de recuperação de informação, e não um pronto-a-uso do produto, e que ele certamente não é um web crawler, como novas pessoas às vezes pensam Lucene. Você também aprendeu um pouco sobre como chegou a ser Lucene e sobre as pessoas-chave ea organização por trás dele.

No espírito de Manning em Ação livros, que rapidamente chegou ao ponto de mostran-ing-lhe duas aplicações standalone, Indexador e Searcher, Que são capazes de arquivos de texto indexação e busca armazenados em um sistema de arquivos. Em seguida, brevemente

descrito cada uma das classes Lucene utilizados nestas duas aplicações. Finalmente, nós apresentou os resultados de nossa pesquisa para alguns produtos semelhante ao Lucene.

Pesquisa está em toda parte, e as chances são de que, se você estiver lendo este livro, você está inter-  
 interessadas na busca de ser uma parte integrante de suas aplicações. Dependendo de suas necessidades,

integração Lucene pode ser trivial, ou pode envolver considerações de arquitetura

Temos organizado próximos dois capítulos, como fizemos neste capítulo. O primeiro coisa que precisamos fazer é o índice de alguns documentos; discutimos esse processo em detalhes no capítulo 2.

# Indexação



## Este capítulo aborda

- Executando operações de índice básico
- Impulsionar Documents and Fields durante a indexação
- Indexação datas, números e Campos para uso em Classificando resultados de pesquisa
- Utilizando parâmetros que afetam a indexação do Lucene desempenho e consumo de recursos
- Otimizando índices
- Simultaneidade entendimento, multithreading e bloqueio de questões no contexto da indexação

Então você deseja pesquisar os arquivos armazenados em seu disco rígido, ou talvez procurar o seu e-mail, páginas web, ou até mesmo dados armazenados em um banco de dados. Lucene pode ajudá-lo a fazer isso. Como nunca, antes que você pode pesquisar alguma coisa, você tem que indexá-lo, e é isso que você vai aprender a fazer neste capítulo.

No capítulo 1, você viu um exemplo simples de indexação. Este capítulo vai mais longe e ensina-lo sobre atualizações de índice, parâmetros que você pode usar para ajustar o índice tuição, e mais avançadas técnicas de indexação que o ajudarão a obter o máximo proveito do Lucene. Aqui você também encontrará informações sobre a estrutura de um Lucene índice, questões importantes para manter em mente quando estiver acessando um índice Lucene com vários segmentos e processos, o mecanismo de bloqueio que Lucene emprega para impedir a modificação de índice simultâneas.

## 2.1 Compreender o processo de indexação

---

Como você viu no capítulo 1, apenas alguns métodos da API pública do Lucene necessidade de ser chamado, a fim de índice de um documento. Como resultado, a partir da indexação, fora com Lucene parece uma operação aparentemente simples e monolítica. No entanto, por trás da API simples reside um conjunto interessante e relativamente complexo de operações que podemos decompor-se em três grandes grupos distintos e funcionalmente, como descrito nas seções seguintes e ilustrado na figura 2.1.

### 2.1.1 A conversão para texto

Aos dados de índice com Lucene, você deve primeiro convertê-lo para um fluxo de texto simples tokens, o formato que o Lucene pode digerir. No capítulo 1, limitamos nossos exemplos de indexação e busca de arquivos. txt, o que nos permitiu slurp seu conteúdo e usá-lo para preencher Campo instâncias. No entanto, as coisas nem sempre são tão simples.

Suponha que você precisa para indexar um conjunto de manuais em formato PDF. Para preparar esses manuais de indexação, você deve primeiro encontrar uma maneira de extrair a informação textual dos documentos PDF e usar que os dados extraídos para criar Lucene Documentos e seus Campos. Se você olhar para trás, quadro 1.2, página 21, você verá que Campo métodos de sempre ter Corda valores e, em alguns casos, Data e Leitor valores. Nenhum método aceitaria um tipo Java PDF, mesmo se tal tipo existiu. Você enfrenta mesma situação se você quer indexar documentos do Microsoft Word ou qualquer documento outro formato de texto puro. Mesmo quando você está lidando com XML ou HTML documentos, que usam caracteres de texto simples, você ainda precisa ser esperto sobre pre-comparando os dados para a indexação, para evitar coisas indexação como elementos XML ou Tags HTML, e indexar os dados reais nesses documentos.

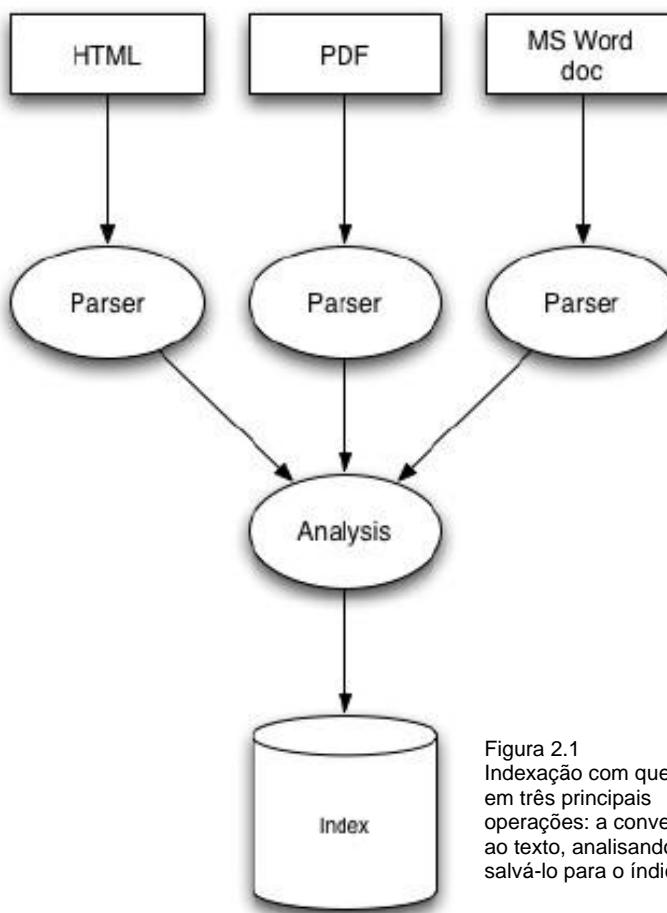


Figura 2.1  
Indexação com quebras Lucene  
em três principais  
operações: a conversão de dados  
ao texto, analisando-o, e  
salvá-lo para o índice.

Os detalhes da extração de texto estão no capítulo 7, onde construímos um pequeno, mas completa estrutura para a indexação de todos os formatos de documento representado na figura 2.1, mais uma alguns outros. Por uma questão de fato, você vai perceber que a figura 2.1 e figura 7.3 resemvel uns aos outros.

### 2.1.2 Análise

Uma vez que você preparou os dados para a indexação e criou Lucene Documentos popovoadas com Campos, você pode chamar `IndexWriter's addDocument (Documento)` método e mão de seus dados off Lucene para indexar. Quando você faz isso, Lucene primeira analisa as dados para torná-la mais adequada para a indexação. Para fazer isso, ele divide o textual dados em blocos, ou tokens, e executa uma série de operações opcional em -los. Por exemplo, as fichas podem ser lowercased antes da indexação, para fazer Buscas maiúsculas e minúsculas. Normalmente é também desejável para remover todos os freqüentes, mas

tokens sentido a partir da entrada, como palavras de parada (a, um, a, em, em, e assim on), em texto em Inglês. Da mesma forma, é comum para analisar fichas de entrada e reduzir -los às suas raízes.

Este passo muito importante é chamado análise. A entrada para o Lucene pode ser analisados de tantas maneiras interessantes e úteis que cobrir este processo em detalhe no capítulo 4. Por agora, acho que dessa etapa como um tipo de filtro.

### 2.1.3 Índice de escrever

Após a entrada tem sido analisada, ela está pronta para ser adicionada ao índice. Lucene lojas de entrada a em uma estrutura de dados conhecida como índice invertido. Esta estrutura de dados

faz uso eficiente de espaço em disco, permitindo pesquisas rápidas palavras-chave. O que torna esta estrutura invertida é que ele usa tokens extraídos de entrada documentos como chaves de pesquisa em vez de tratar os documentos como as entidades centrais. Em

Por outras palavras, ao invés de tentar responder à pergunta "o que as palavras estão contidas neste documento? "esta estrutura é otimizada para fornecer respostas rápidas para "Que documentos contêm X palavra?"

Se você pensar sobre o seu motor de busca favorito web e formato de sua consulta típica, você vai ver que é exatamente a consulta que você quer ser o mais rápido possível. O núcleo de todos os motores de busca na web de hoje são invertidos índices.

O que faz com que cada motor de busca diferentes é um conjunto de truques guardados a sete chaves usadas para

melhorar a estrutura pela adição de mais parâmetros, como o Google é bem conhecido PageRank fator. Lucene, também, tem seu próprio conjunto de truques, você pode aprender sobre alguns deles no apêndice B.

## 2.2 Basic operações de índice

---

No capítulo 1, você viu como adicionar documentos a um índice. Mas vamos resumir o processo aqui, juntamente com as descrições de apagar e operações de atualização, para fornecê-lo com um ponto de referência conveniente único.

### 2.2.1 Adicionar documentos a um índice

Para resumir o que você já sabe, vamos olhar para o trecho de código que serve como a classe base para testes de unidade neste capítulo. O código na listagem de 2.1 cria um com-índice de libra imaginativamente chamado index-dir, armazenadas no sistema de temporários diretório: / tmp em UNIX, ou C: \ TEMP em computadores com Windows. (Compound índices são cobertas no apêndice B.) Usamos SimpleAnalyzer para analisar a entrada texto, e nós índice então duas simples Documentos, cada um contendo todos os quatro tipos de Campos: Palavra chave, UnIndexed, UnStored E Texto.

2.1 listagem Preparando um novo índice antes de cada teste em uma classe de caso base de teste

```

public abstract class BaseIndexingTestCase estende TestCase {
    protected String [] keywords = {"1", "2"};
    protected String [] unindexed = {"Países Baixos", "Itália"};
    protected String [] UnStored = {"Amsterdam tem muitas pontes",
                                    "Veneza tem muitos canais"};
    protegidos String [] {text = "Amsterdam", "Veneza"};
    Diretório protegido dir;

    setUp protected void () throws IOException {
        String = indexDir
        System.getProperty ("java.io.tmpdir", "tmp") +
        System.getProperty ("File.separator") + "index-dir";
        dir = FSDirectory.getDirectory (indexDir, true);
        addDocuments (dir);
    }

    protegidos addDocuments void (Directory dir)
        throws IOException {
        Escritor IndexWriter IndexWriter = new (dir, getAnalyzer (),
        true);
        writer.setUseCompoundFile (isCompound ());
        for (int i = 0; i keywords.length <; i + +) {
            Documento doc = new Document ();
            doc.add (Field.Keyword ("id", palavras-chave [i]));
            doc.add (Field.UnIndexed ("country", não indexados [i]));
            doc.add (Field.UnStored ("conteúdo", UnStored [i]));
            doc.add (Field.Text ("city" de texto, [i]));
            writer.addDocument (doc);
        }
        writer.optimize ();
        writer.Close ();
    }

    protegidos Analyzer getAnalyzer () {
        return new SimpleAnalyzer ();
    }

    protegidos isCompound boolean () {
        return true;
    }
}

```

Executado antes  
todos os testes

← Padrão  
Analizador

Uma vez que este `BaseIndexingTestCase` classe será prorrogado por outras classes de teste da unidade neste capítulo, vamos destacar alguns detalhes importantes. `BaseIndexingTestCase` cria o mesmo índice cada vez que a sua `setUp ()` método é chamado. Desde `setUp ()` é chamado antes de um teste é executado, cada teste executado contra um índice de recém-criado.

Embora a classe base usa `SimpleAnalyzer`, As subclasses podem substituir o `getAnalyzer()` método para retornar um tipo diferente de Analisador.

### Documentos heterogêneos

Um recurso útil do Lucene é que ele permite Documentos com diferentes conjuntos de Campos para coexistir no mesmo índice. Isto significa que você pode usar um único índice a segurar Documentos que representam entidades diferentes. Por exemplo, você poderia ter Documentos que representam produtos de varejo com Campos, tais como nome e preço, e Documentos que representam as pessoas com Campos, tais como nome, idade E gênero.

### Campos appendable

Suponha que você tenha um aplicativo que gera uma série de sinônimos para uma determinada palavra, e você deseja usar Lucene para indexar a palavra base e todos os seus sinônimos. Uma maneira de fazê-lo seria fazer um loop por todos os sinônimos e anexá-la um único `Corda`, Que você poderia usar para criar um Lucene `Campo`. Outra, perhaps maneira mais elegante para indexar todos os sinônimos, juntamente com a palavra base é apenas manter-se adicionar o mesmo `Campo` com valores diferentes, como este:

```
String baseWord = "rápido";
Sinônimos String [] = {"quick", "rápido", "speedy"};
Documento doc = new Document ();
doc.add (Field.Text ("palavra", baseWord));
for (int i = 0; i < synonyms.length; i++) {
    doc.add (Field.Text ("palavra", sinônimos [i]));
}
```

Internamente, acrescenta Lucene todas as palavras juntas e indexá-los em um único Campo chamado `palavra`, Permitindo que você usar qualquer uma das palavras dadas na busca.

### 2.2.2 Remover documentos de um índice

Embora a maioria das aplicações estão mais preocupados com a obtenção de Documentos em um Lucene índice, alguns também necessidade de removê-los. Por exemplo, um jornal publicador pode querer manter só vale a última semana de notícias em seu searchable índices. Outras aplicações pode querer remover todos os Documentos que contêm um certain prazo.

Documento exclusão é feito usando uma classe que é um tanto inadequadamente chamado `IndexReader`. Esta classe não exclui Documentos do índice imediatamente. Em vez disso, ele marca-os como excluídos, esperando que o real Documento eliminação até `IndexReader's close()` método é chamado. Com isto em mente, vamos olhar para Listing 2.2: Ele herda `BaseIndexingTestCase` classe, o que significa que antes de cada teste método é executado, a classe base re-cria a dois Documento índice, como descrito no ponto 2.2.1.

Listagem 2,2 Retirar Documentos de um índice Lucene por internas Documento número

```

public class DocumentDeleteTest estende BaseIndexingTestCase {

    testDeleteBeforeIndexMerge public void () throws IOException {
        IndexReader reader = IndexReader.open (dir);
        assertEquals (2, reader.maxDoc()); Número do documento seguinte é 2
        assertEquals (2, reader.numDocs()); 2 Os documentos no índice de
        reader.delete (1); Excluir Documento com id 1      C
            d

        assertTrue (reader.isDeleted (1));
        assertTrue (reader.hasDeletions ());
        assertEquals (2, reader.maxDoc ());
        assertEquals (1, reader.numDocs ());

        reader.Close ();

        reader = IndexReader.open (dir);

        assertEquals (2, reader.maxDoc ());
        assertEquals (1, reader.numDocs ());
            h Documento próxima
            número é 2, após
            IndexReader reaberto
        reader.Close ();
    }

    testDeleteAfterIndexMerge public void () throws IOException {
        IndexReader reader = IndexReader.open (dir);
        assertEquals (2, reader.maxDoc ());
        assertEquals (2, reader.numDocs ());
        reader.delete (1);
        reader.Close ();

        Escritor IndexWriter IndexWriter = new (dir, getAnalyzer (),
            false);
        writer.optimize ();
        writer.Close ();

        reader = IndexReader.open (dir);

        assertFalse (reader.isDeleted (1));
        assertFalse (reader.hasDeletions ());
        assertEquals (1, reader.maxDoc ());
        assertEquals (1, reader.numDocs ());
            Eu Otimizando
            renumerando
            Documentos
        reader.Close ();
    }
}

```

bcd O código na listagem 2.2 mostra como excluir um Documento especificando seu internas Documento número. Ela também mostra a diferença entre dois IndexReader métodos

que muitas vezes são misturadas: `maxDoc ()` e `numDocs ()`. O primeiro retorna o próximo interno disponível Documento número, e este retorna o número de Documentos em um índice. Porque o nosso índice contém apenas dois Documentos, `numDocs ()` retorna 2, e desde Documento números começam a partir de zero, `maxDoc ()` retorna 2 também.

**NOTA** Cada Lucene Documento tem um número único interno. Esses números atribuições não são permanentes, porque renomeia Lucene Documentos internamente segmentos quando o índice são mesclados. Assim, você não deve assumir que um determinado Documento terá sempre o mesmo Documento número.

**ef** O teste de unidade no `testDeleteBeforeIndexMerge ()` método também demonstra o uso de `IndexReader's hasDeletions ()` método para verificar se um índice contém qualquer Documentos marcado para exclusão e as `isDeleted (int)` método para verificar a status de um Documento especificado pelo seu Documento número.

**gh** Como você pode ver, `numDocs ()` está ciente da Documento remoção imediatamente, enquanto que

`maxDoc ()` não é.

**Eu** Além disso, no método `testDeleteAfterIndexMerge ()` fechamos a índice Leitor e força Lucene para fundir segmentos de índice por meio da otimização do índice. Quando

que posteriormente abrir o índice com `IndexReader`, O `maxDoc ()` método retorna um ao invés de dois, porque depois de um Lucene excluir e mesclar, renomeado o restante- ing Documentos. Apenas um Documento permanece no índice, de modo que o próximo disponível Documento número é 1.

Além de excluir um único Documento especificando seu Documento número, como temos feito, você pode excluir vários Documentos usando `IndexReader's delete (Term)` método. Usando este método de eliminação permite excluir todos os Documentos que contêm o especificado prazo. Por exemplo, para remover um Documento que contém a palavra Amster- barragem em um cidade campo, você pode usar `IndexReader` assim:

```
IndexReader reader = IndexReader.open (dir);
reader.delete (Termo de novo ("city", "Amsterdam"));
reader.Close ();
```

Você deve ter cuidado extra ao utilizar esta abordagem, porque especificar um termo presente em todos os indexados Documentos vai acabar com um índice inteiro. O uso deste método é semelhante ao Documento número baseado em método de eliminação, você pode vê- lo

no ponto 2.2.4.

Você pode se perguntar por Lucene executa Documento exclusão da `IndexReader` e não `IndexWriter` instâncias. Que pergunta é feita na comunidade Lucene a cada poucos meses, provavelmente devido a nomes de classes imperfeitos e talvez enganosa. Lucene usuários costumam pensar que `IndexWriter` é a única classe que pode modificar uma

índice e que `IndexReader` acessa um índice de um modo read-only. Na realidade, `IndexWriter` toca apenas a lista de segmentos de índice e um pequeno subconjunto do índice arquivos quando os segmentos são mesclados. Por outro lado, `IndexReader` saiba como analisar todos os arquivos de índice e fazer sentido fora delas. Quando um `Documento` é excluído,

`IndexReader` necessidades em primeiro lugar para localizar o segmento que contém o especificado `Documento`

antes que possa marcá-lo como excluído. Actualmente não há planos para mudar tanto o nomes ou o comportamento dessas duas classes Lucene.

### 2.2.3 Documentos Recuperando

Porque `Documento` exclusão é dia de hoje é o encerramento do `IndexReader` exemplo, Lucene permite que um aplicativo para mudar sua mente e `undelete` `Documento`s que foram marcados como excluídos. Uma chamada para `IndexReader`'s `undeleteAll ()` método

`undeletes` todos os excluídos `Documento`, removendo todos os arquivos. del do diretório índice. Posteriormente, fechar o `IndexReader` exemplo, portanto, deixa todos os `Documento`s em o índice. `Documento`s podem ser recuperados somente se a chamada para `undeleteAll ()` foi feito

usando a mesma instância de `IndexReader` que foi usado para apagar o `Documento`s em primeiro lugar.

### 2.2.4 Atualizando Documentos em um índice

"Como faço para atualizar um documento em um índice?" É uma pergunta freqüente na a lista de discussão Lucene usuário. Lucene não oferece uma `update (Documento)` método; em vez disso, uma `Documento` deve primeiro ser excluídos de um índice e, em seguida, re-adicionado a ele, como mostrado na listagem 2.3.

Listagem 2.3 Atualizando indexados Documentos pela primeira excluí-los e depois re-adicioná-los

```
public class DocumentUpdateTest estende BaseIndexingTestCase {

    testUpdate public void () throws IOException {
        assertEquals (1, getHitCount ("city", "Amsterdam"));

        IndexReader reader = IndexReader.open (dir);
        reader.delete (Termo de novo ("city", "Amsterdam"));
        reader.Close ();

        assertEquals (0, getHitCount ("city", "Amsterdam"));

        Escritor IndexWriter IndexWriter = new (dir, getAnalyzer (), false); Re-add Documento
        Documento doc = new Document (); com nova cidade
        doc.add (Field.Keyword ("id", "1")); name: "Haag"
```

```
doc.add (Field.UnIndexed ("country", "Países Baixos"));
doc.add (Field.UnStored ("conteúdo",
    "Amsterdam tem muitas pontes"));
doc.add (Field.Text ("city", "Haag"));
writer.addDocument (doc);
writer.optimize ();
writer.Close ();

assertEquals (1, getHitCount ("city", "Haag"));

protected Analyzer getAnalyzer () {
    return new WhitespaceAnalyzer ();
}

private int getHitCount (String fieldName, String searchString)
throws IOException {
    IndexSearcher searcher = new IndexSearcher (dir);
    Termo t = novo Termo (fieldName, searchString);
    Query = new TermQuery (t);
    Hits hits = searcher.search (query);
    int HitCount hits.length = ();
    searcher.close ();
    retorno HitCount;
}

}
```

↑  
Re-add Documento  
com nova cidade  
name: "Haag"

← Verifique Documento  
atualizar

Nós primeiramente removemos todos os Documentos cuja cidade Campo contém o termo Amsterdam; então  
nós adicionamos um novo Documento cuja Campos são as mesmas que as do removido Documento,  
exceto por um novo valor na Campo cidade. Em vez do Amsterdam, o novo Documento tem Haag em seu Campo cidade. Temos efetivamente atualizado um dos Documentos no índice.

#### Atualização por lotes exclusões

Exclui o nosso exemplo e re-adiciona uma única Documento. Se você precisa excluir e adicionar vários Documentos, é melhor fazê-lo em lotes. Siga estes passos:

- 1 Aberto IndexReader.
- 2 Apagar todas as Documentos precisa de apagar.
- 3 Fechar IndexReader.
- 4 Aberto IndexWriter.
- 5 Adicione todos os Documentos você precisa adicionar.
- 6 Fechar IndexWriter.

Isto é importante para lembrar: Batching Documento exclusão e de indexação ser sempre mais rápido do que interleaving excluir e adicionar operações.

Com adicionar, atualizar e excluir operações sob seu cinto, vamos discutir como aperfeiçoar o desempenho de indexação e fazer o melhor uso de disponível hardware.

**TIP** Ao excluir e adicionar Documentos, fazê-lo em lotes. Esta será sempre ser mais rápido que interleaving excluir e adicionar operações.

## 2,3 Documentos reforço e Campos

---

Nem todos os Documentos e Campos são criados iguais, ou pelo menos você pode ter certeza esse é o caso seletivamente impulsionar Documentos ou Campos. Imagine que você tem que escrever um aplicativo que indexa e pesquisa de e-mail corporativo. Talvez a exigência é para dar e-mails funcionários da empresa "mais importância do que outros mensagens de e-mail. Como é que você vai fazer sobre isso?

Documento impulsionar é um recurso que faz essa exigência simples de implement. Por padrão, todos os Documentos não tem impulso, ou melhor, todos têm o mesmo aumentar factor de 1,0. Alterando uma Documento "Fator de impulso s, você pode instruir o Lucene

considerá-la mais ou menos importante em relação aos outros Documentos no índice. A API para fazer isso consiste em um único método, `setBoost (float)`, Que pode ser usados como segue:

```
String public static final COMPANY_DOMAIN = "example.com";
String public static final BAD_DOMAIN = "yucky-domain.com";

Documento doc = new Document ();
String = senderEmail getSenderEmail ();
String = senderName getSenderName ();
String subject = getSubject ();
Corpo StringgetBody = ();
doc.add (Field.Keyword ("senderEmail", senderEmail));
doc.add (Field.Text ("senderName", senderName));
doc.add (Field.Text ("sujeito", assunto));
doc.add (Field.UnStored ("corpo", corpo));
if (getSenderDomain ().endsWithIgnoreCase (COMPANY_DOMAIN)) {
    doc.setBoost (1,5);Fator empregado boost: 1,5
}
else if (getSenderDomain ().endsWithIgnoreCase (BAD_DOMAIN)) {
    doc.setBoost (0,1);Fator de impulso mau domínio: 0,1
}
writer.addDocument (doc);
```

Neste exemplo, vamos verificar o nome de domínio do remetente da mensagem de e-mail para determinar se o remetente é um funcionário da empresa.

- b Quando as mensagens enviadas por nós do índice os funcionários da empresa, que definiu seu impulso factor de 1,5, que é maior do que o fator padrão de 1,0.
- c Quando nos deparamos com mensagens de um remetente associado a uma má ficção domínio, classificá-los como quase insignificante, reduzindo o seu fator de impulso para 0,1.

Assim como você pode aumentar Documentos, você também pode aumentar individual Campos. Quando você

um impulso Documento, Lucene internamente usa o fator de mesmo impulso para aumentar a cada um de seus

Campos. Imagine que outro requisito para a aplicação de e-mail de indexação é a considerar o assunto Campo mais importante que o Campo com um remetente nome. Em outras palavras, os jogos realizados na pesquisa o assunto Campo deve ser mais valioso do que partidas equivalente no senderName Campo em nosso exemplo anterior.

Para atingir esse comportamento, usamos o setBoost (float) método da Campo classe:

```
Campo senderNameField = Field.Text ("senderName", senderName);
Campo subjectField = Field.Text ("sujeito", assunto);
subjectField.setBoost (1.2);
```

Neste exemplo, nós arbitrariamente escolhido um fator de impulso de 1.2, assim como nós arbitrariamente

escolhido Documento aumentar os fatores de 1,5 e 0,1 anterior. O fator de impulso valores que você

deve usar dependerá do que você está tentando alcançar, você pode precisar fazer um pouco de experimentação e tuning para alcançar o efeito desejado.

É interessante notar que mais curto Campos têm um impulso implícita associada a eles, devido às obras do Lucene forma de algoritmo de pontuação. Aumento é, em geral, um recurso avançado que muitas aplicações podem funcionar muito bem sem.

Documento e Campo impulsionar entra em jogo no momento da pesquisa, como você vai aprender em

seção 3.5.9. Resultados de busca do Lucene são classificados de acordo com a proximidade de cada

Documento corresponda à consulta, e cada correspondência Documento é atribuída uma pontuação.

Lucene fórmula de pontuação é composto por uma série de fatores, eo fator de impulso é um deles.

## 2.4 datas de indexação

Mensagens de e-mail incluem datas enviadas e recebidas, arquivos têm timestamps várias que lhes estão associados, e as respostas HTTP ter um cabeçalho Last-Modified que inclui a data da última modificação é a página solicitada. As possibilidades são, como muitos usuários Lucene outros, você vai precisar datas índice. Lucene vem equipado com um Field.Keyword (Date, String) método, bem como um DateField classe, que fazer a indexação data fácil. Por exemplo, para a data de hoje índice, você pode fazer isso:

```
Documento doc = new Document ();
doc.add (Field.Keyword ("indexDate", new Date ()));
```

Internamente, utiliza o Lucene `DateField` classe para converter a data dada a um `Corda` adequados para a indexação. Manipulação de datas desta maneira é simples, mas você deve ter cuidado

Ao usar esse método: Datas convertido para indexável `Cordas` por `DateField` incluir todas as partes da data, até o milésimo de segundo. À medida que você vai ler na seção 6.5,

isso pode causar problemas de desempenho para determinados tipos de consultas. Na prática, você raramente precisa datas que são precisos em milissegundos, pelo menos para consulta em. Geralmente, você pode rodada datas a uma hora ou até mesmo para um dia.

Uma vez que todos `Campo` valores são, eventualmente, se transformou em texto, você pode muito bem `index`

datas como `Cordas`. Por exemplo, se você pode arredondar a data para um dia, o índice de datas como

`YYYYMMDD` `Cordas` usando o `Field.Keyword (String, String)` método. Outro boa razão para esta abordagem é que você vai ser capaz de datas índice antes a Era Unix (01 de janeiro de 1970), que `DateField` não pode lidar. Apesar de vários soluções e patches para resolver esta limitação tem sido contribuiu com mais de últimos anos, nenhum deles era suficientemente elegante. Como consequência, eles ainda podem ser encontrados na fila do Lucene patch, mas eles não são incluídos no Lucene. A julgar pela freqüência com que os usuários Lucene trazer esta limitação, não sendo capaz de índice de datas anteriores a 1970 não é um problema geral.

**NOTA** Se você só precisa da data de busca, e não o timestamp, o índice de `Field.Keyword ("data", "AAAAMMDD")`. Se o timestamp completo precisa ser preservado para a recuperação, o índice de um segundo `Campo` como `Field.Keyword ("Timestamp", <java.util.Date>)`.

Se você optar por formatar datas ou horas de alguma outra forma, tome muito cuidado para que o `Corda` representação é lexicographically orderable, pois isso permite a sensível intervalo de datas-queries. Um benefício de datas no formato `YYYYMMDD` indexação é a capacidade de consulta por ano apenas, por ano e por mês ou por ano exato, mês e dia. Para consulta por ano apenas, usar um `PrefixQuery` para AAAA, por exemplo. Discutimos `PrefixQuery` ainda mais no ponto 3.4.3.

## 2.5 números de indexação

---

Há dois cenários comuns em que a indexação número é importante. Em um cenário, os números são incorporados no texto a ser indexado, e você quer certifique-se que esses números são indexados para que você possa usá-los mais tarde, em buscas.

Por exemplo, os documentos podem conter frases como "Mt. Everest é 8848

metros de altura ": Você quer ser capaz de procurar o número 8848 como você pode procurar a palavra Evereste e recuperar o documento que contém a sentença.

No outro cenário, você tem Campos que contenham apenas valores numéricos e você quer ser capaz de indexá-los e usá-los para pesquisa. Além disso, você pode querer executar consultas intervalo usando tais Campos. Por exemplo, se você está indexação de mensagens de e-mail, um dos índices possíveis Campos poderia ser a message tamanho, e você pode querer ser capaz de encontrar todas as mensagens de um determinado tamanho, ou, você pode querer usar consultas faixa para encontrar todas as mensagens cujo tamanho é em certo alcance. Você também pode ter de classificar os resultados por tamanho.

Lucene pode indexar os valores numéricos por tratá-los como strings internamente. Se você necessidade de números de índice que aparecem no texto de formato livre, a primeira coisa que você deve fazer é escolher o Analisador que não descarta números. Como discutimos na seção 4.3, WhitespaceAnalyzer e StandardAnalyzer são dois possíveis candidatos. Se você alimentá-los uma frase como "Mt. Everest é 8.848 metros de altura, "extraem 8848 como um sinal e passá-la para a indexação, permitindo que você pesquise mais tarde 8848. Em Por outro lado, SimpleAnalyzer e StopAnalyzer throw números fora do token stream, o que significa a busca por 8848 não há documentos correspondentes.

Campos, cujo único valor é um número não precisam ser analisados, por isso devem ser indexado como Field.Keyword. No entanto, antes apenas adicionando seus valores brutos para o índice, é preciso manipulá-los um pouco, para consultas de intervalo para trabalhar como esperado. Ao realizar consultas de intervalo, Lucene usa lexicographical valores de Campos para encomendar. Considere três numéricos Campos cujos valores são de 7, 71, e 20. Apesar de sua ordem natural é de 7, 20, 71, sua ordem é lexicográfica 20, 7, 71. Um truque simples e comuns para resolver esta inconsistência é prepad numérico Campos com zeros, assim: 007, 020, 071. Observe que a natural e a ordem lexicográfica dos números agora é consistente. Para mais detalhes sobre busca numérica Campos, consulte a seção 6.3.3.

**NOTA** Quando você index Campos com valores numéricos, pad-los se você quiser usá-los para consultas

## 2.6 campos de indexação utilizado para classificar

---

Ao retornar os resultados das pesquisas, as ordens Lucene-los por sua pontuação por padrão. Alguns-vezes, no entanto, é preciso ordenar os resultados usando alguns outros critérios. Por exemplo, se você estiver procurando mensagens de email, você pode querer resultados por fim enviados ou recebeu hoje, ou talvez pelo tamanho da mensagem. Se você quiser ser capaz de classificar os resultados por um Campo valor, você deve adicioná-lo como um Campo que é indexada, mas não tokenized (para

exemplo, `Field.Keyword`). Campos usado para classificação deve ser conversível para Número inteiroS,

**FlutuarS, OU Cordas:**

```
Field.Keyword ("Tamanho", "4096");
Field.Keyword ("preço", "10,99");
Field.Keyword ("autor", "Arthur C. Clark");
```

Embora tenhamos indexados valores numéricos como `Cordas`, você pode especificar o correto Campo tipo (como Número inteiro ou Longo) No momento da espécie, como descrito na seção 5.1.7.

**NOTA** Campos usado para classificação têm de ser indexados e não deve ser indexado.

## 2.7 Controlar o processo de indexação

---

Indexação coleções de documentos pequenas e médias funciona bem com o padrão Setup Lucene. No entanto, se o seu aplicativo lida com índices muito grandes, você provavelmente quer algum controle sobre o processo de indexação Lucene para garantir melhor desempenho da indexação. Por exemplo, você pode ser a indexação de vários milhões de documentos e quer acelerar o processo para que ele leva alguns minutos em vez de horas.

Seu computador pode ter memória RAM de reposição, mas você precisa saber como deixar Lucene fazer mais uso dele. Lucene tem vários parâmetros que permitem controlar a sua desempenho e utilização de recursos durante a indexação.

### 2.7.1 desempenho da indexação Tuning

Em uma aplicação típica de indexação, o gargalo é o processo de escrita do índice arquivos em um disco. Se você fosse o perfil de um aplicativo de indexação, você veria que a maior parte do tempo é gasto em seções de código que manipulam arquivos de índice. Portanto,

você precisa instruir Lucene que ser esperto sobre indexação de novos Documentos e modificação arquivos de índice existente.

Como mostrado na figura 2.2, quando novos Documentos são adicionados a um índice do Lucene, eles são

inicialmente armazenadas na memória em vez de ser imediatamente gravado no disco.

Este buffer é feito por razões de desempenho, e felizmente, a `IndexWriter` classe expõe várias variáveis de instância que lhe permitem ajustar o tamanho deste buffer e freqüência de escritas em disco. Essas variáveis são resumidas na tabela 2.1.

Tabela 2.1 Parâmetros para o ajuste de desempenho de indexação

IndexWriter variável	Propriedade do sistema	Padrão valor	Descrição
mergeFactor	org.apache.lucene.mergeFactor	10	Controles segmento de fusão e freqüência eo tamanho

continua na página seguinte

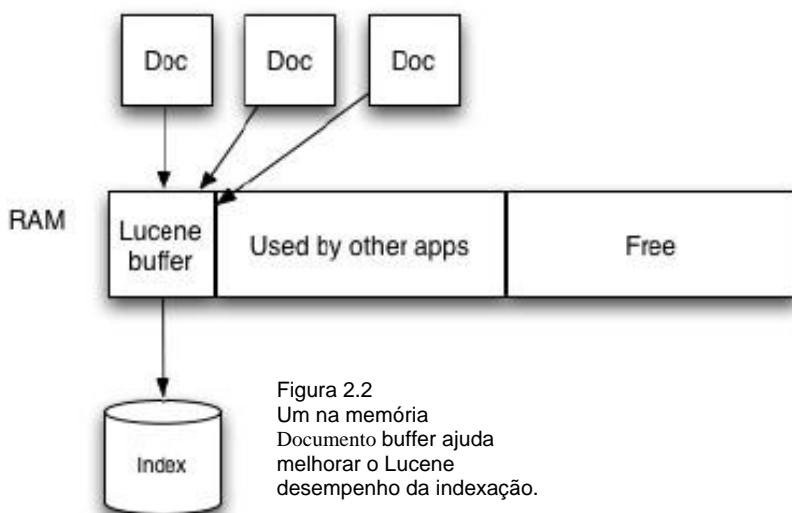
Tabela 2.1 Parâmetros para o ajuste de desempenho de indexação (Continuação)

IndexWriter variável	Propriedade do sistema	Padrão valor	Descrição
maxMergeDocs	org.apache.lucene.maxMergeDocs	Integer.MAX_VALUE	Limita o número de documentos por segmento
minMergeDocs	org.apache.lucene.minMergeDocs	10	Controla a quantidade de RAM utilizado quando indexação

IndexWriter's mergeFactor permite controlar quantas Documentos para armazenar em memory antes de escrevê-las para o disco, bem como a freqüência com a fusão de vários índices segmentos juntos. (Segmentos de índice são cobertas no apêndice B.) Com o padrão valor de 10, Lucene 10 lojas Documentos na memória antes de escrevê-las a um pecado de segmento no disco. O mergeFactor valor de 10 significa também que uma vez que o número de segmentos no disco atingiu a potência de 10, Lucene funde estes segmentos em um único segmento.

Por exemplo, se você definir mergeFactor a 10, um novo segmento é criado no disco para cada 10 Documentos adicionado ao índice. Quando o segmento décimo do tamanho 10 é adicionado, todos os 10 são fundidos em um único segmento de tamanho 100. Quando tais 10 segmentos de tamanho 100 foram adicionados, eles são fundidos em um único segmento containing 1000 Documentos, e assim por diante. Portanto, a qualquer momento, não há mais de 9 segmentos no índice, eo tamanho de cada segmento de fundidos é a potência de 10.

Há uma pequena exceção a esta regra, que tem a ver com maxMergeDocs, outro IndexWriter variável de instância: Enquanto fusão segmentos, garante Lucene



que nenhum segmento com mais de Documento maxMergeDocs é criado. Por exemplo, suponha que você defina maxMergeDocs para 1000. Quando você adiciona o décimo de milésimo Documento, Em vez de reunir vários segmentos em um único segmento de tamanho 10000, Lucene cria o segmento décimo do tamanho de 1000 e continua a adicionar novos segmentos de tamanho 1.000 para cada 1.000 Documentos acrescentou.

Agora que você já viu como mergeFactor e maxMergeDocs trabalham, você pode deduzir que o uso de um valor maior para mergeFactor causa Lucene usar mais RAM mas deixá-lo gravar dados em disco com menos freqüência, consequentemente acelerar a indexação processo. Um menor mergeFactor usa menos memória e faz com que o índice a ser atualizado com mais freqüência, o que torna mais atualizado, mas também diminui a processo de indexação. Da mesma forma, uma maior maxMergeDocs é mais adequado para lote indexação, e uma menor maxMergeDocs é melhor para indexação mais interativo. Ser ciente de que, pois uma maior mergeFactor significa funde menos freqüentes, que resulta em um índice com mais arquivos de índice. Embora isso não afeta desempenho de indexação desempenho, pode retardar a pesquisa, porque Lucene precisará abrir, ler, e processo mais arquivos de índice.

minMergeDocs é outra IndexWriter variável de instância que afeta a indexação desempenho. Seu valor controla quantas Documentos tem que ser tamponado antes eles são fundidos para um segmento. O minMergeDocs parâmetro permite o comércio de mais de sua RAM para uma mais rápida indexação. Ao contrário mergeFactor, Este parâmetro não afeta o tamanho dos segmentos de índice no disco.

### Exemplo: IndexTuningDemo

Para ter uma melhor idéia de como diferentes valores de mergeFactor, maxMergeDocs e minMergeDocs afetam a velocidade de indexação, olhar para o IndexTuningDemo classe na listagem 2.4.

Esta classe tem quatro de linha de comando argumentos: o número total de Documentos para adicionar ao índice, o valor a ser usado para mergeFactor, O valor a ser usado para maxMergeDocs, E o valor para minMergeDocs. Todos os quatro argumentos devem ser especificado, deve ser inteiros, e devem ser especificados por esta ordem. A fim de manter o código curto e limpo, não há nenhuma verificação para o uso indevido.

Demonstração de listagem 2,4 de usar mergeFactor, maxMergeDocs, e minMergeDocs

```
public class {IndexTuningDemo

    public static void main (String [] args) throws Exception {
        int docsInIndex = Integer.parseInt (args [0]);

        / / Cria um índice chamado "índice-dir" em um diretório temp
        Diretório dir = FSDirectory.getDirectory (
```

```

System.getProperty ("java.io.tmpdir", "tmp") +
System.getProperty ("File.separator") + "index-dir", true);
Analizador analisador = new SimpleAnalyzer ();
Escritor IndexWriter IndexWriter = new (dir, analisador, true);

// Definir as variáveis que afetam a velocidade de indexação
writer.mergeFactor = Integer.parseInt (args [1]); Ajustar as configurações que
writer.maxMergeDocs = Integer.parseInt (args [2]); afetam indexação
execução writer.minMergeDocs = Integer.parseInt (args [3]);
writer.infoStream = System.out; Diga IndexWriter para imprimir
C info para System.out
System.out.println ("factor Merge:" + writer.mergeFactor);
System.out.println ("Max fundir docs:" + writer.maxMergeDocs);
System.out.println ("Min fundir docs:" + writer.minMergeDocs);

longo start = System.currentTimeMillis ();
for (int i = 0; i < docsInIndex; i + +) {
    Documento doc = new Documento ();
    doc.add (Field.Text ("nome_do_campo", "Bibamus"));
    writer.addDocument (doc);
}
writer.Close ();
longa stop = System.currentTimeMillis ();
System.out.println ("Time:" + (stop - start) + "ms");

}

}

```

O primeiro argumento representa o número de Documentos para adicionar o índice, o segundo argumento é o valor a ser usado para o `mergeFactor`, Seguido por `maxMergeDocs` valor, eo último argumento é o valor a ser usado para o `minMergeDocs` parâmetro:

```

% Java lia.indexing.IndexTuningDemo 100000 10 9999999 10

Merge fator: 10
Max fundir docs: 9999999
Min fundir docs: 10
Time: 74136 ms

```

```

% Java lia.indexing.IndexTuningDemo 100000 100 10 9999999
Merge factor: 100
Max fundir docs: 9999999
Min fundir docs: 10
Time: 68307 ms

```

Ambos invocações criar um índice com 100.000 Documentos, mas o primeiro leva mais tempo para concluir (74.136 versus 68.307 ms ms). Isso porque o primeiro invocações ção usa o padrão `mergeFactor` de 10, que faz com que o Lucene para escrever Documentos para

o disco com mais freqüência do que a segunda chamada (`mergeFactor` de 100). Vamos olhar algumas corridas mais com diferentes valores de parâmetros:

```
% Java lia.indexing.IndexTuningDemo 100000 10 9999999 100
```

```
Merge factor: 10
Max fundir docs: 9999999
Min fundir docs: 100
Time: 54050 ms
```

```
% Java lia.indexing.IndexTuningDemo 100000 100 9999999 100
```

```
Merge factor: 100
Max fundir docs: 9999999
Min fundir docs: 100
Time: 47831 ms
```

```
% Java lia.indexing.IndexTuningDemo 100000 100 9999999 1000
```

```
Merge factor: 100
Max fundir docs: 9999999
Min fundir docs: 1000
Time: 44235 ms
```

```
% Java lia.indexing.IndexTuningDemo 100000 1000 9999999 1000
```

```
Merge factor: 1000
Max fundir docs: 9999999
Min fundir docs: 1000
Time: 44223 ms
```

```
% Java -server-Xms128m-Xmx256m
```

```
⇒ lia.indexing.IndexTuningDemo 100000 1000 9999999 1000
```

```
Merge factor: 1000
Max fundir docs: 9999999
Min fundir docs: 1000
Time: 36335 ms
```

```
% Java lia.indexing.IndexTuningDemo 100000 1000 9999999 10000
Exceção em java.lang.OutOfMemoryError thread "main"
```

Velocidade de indexação melhora à medida que aumentamos `mergeFactor` e `minMergeDocs` quando damos a JVM maior de início e máximo de heap. Note como o uso de 10.000 para `minMergeDocs` resultou em um `OutOfMemoryError`; Isso também pode acontecer se você escolha muito grande um `mergeFactor` valor.

**NOTA** Aumentando `mergeFactor` e `minMergeDocs` melhora a velocidade de indexação, mas apenas até certo ponto. Valores mais altos também usam mais memória RAM e pode causar o seu processo de indexação para executar fora de memória, se eles estão muito alto.

Tenha em mente que o `IndexTuningDemo` é, como o próprio nome indica, apenas uma demonstração do uso e efeito de `mergeFactor`, `maxMergeDocs` E `minMergeDocs`. Em esta classe, adicionamos Documentos com um único Campo consistindo de uma única palavra. Conseqüentemente, podemos usar um nível muito elevado `mergeFactor`. Na prática, as aplicações que utilizam Lucene tendem a trabalhar com índices cujos documentos têm várias Campos e cuja Campos contêm pedaços maiores do texto. Essas aplicações não serão capazes de uso `mergeFactor` e `minMergeDocs` valores tão elevados como os que usamos aqui, a menos que eles correm em computadores com grandes quantidades de RAM, que é o fator que limites `mergeFactor` e `minMergeDocs` para um determinado índice. Se você optar por executar `IndexTuningDemo`, Tenha em mente o efeito que o sistema operacional e arquivos sistema-cache de temperatura podem ter sobre o seu desempenho. Certifique-se de aquecer os caches e executar cada configuração várias vezes, de preferência no computador ocioso. Furdido, criar um índice grande o suficiente para minimizar o efeito desses caches. Finalmente, vale a pena repetir que o uso de uma maior `mergeFactor` afetará pesquisa desempenho-aumentar o seu valor com cautela.

**NOTA** Não se esqueça que dar a sua JVM uma pilha maior de memória pode melhorar desempenho da indexação. Isso é muitas vezes feito com uma combinação de `-Xms` e `-Xmx` linha de comando argumentos para o interpretador Java. Dando a JVM heap maior também permite que você aumentar os valores dos `mergeFactor` e `minMergeDocs` parâmetros. Certificando-se que o HotSpot, JIT, ou JVM opção similar é habilitado também tem efeitos positivos.

## Alterando o limite máximo de arquivos abertos em UNIX

Note que, embora estas três variáveis pode ajudar a melhorar a indexação de desempenho, eles também afetar o número de descritores de arquivos que usa e pode Lucene portanto, fazer com que o "Excesso de abrir arquivos" exceção quando usado com vários arquivos

índices. (Índices Multifile e índices compostos são cobertos no apêndice B.) Se você receber esse erro, você deve primeiro verificar o conteúdo do seu diretório de índice. Se contém vários segmentos, você deve otimizar o índice usando `IndexWriter's` `optimize()` método, como descrito na seção 2.8; otimização ajudam índices que conter mais de um segmento, fundindo-os em um segmento de índice único. Se otimizar o índice não resolve o problema, ou se o seu índice já só tem um único segmento, você pode tentar aumentar o número máximo de arquivos abertos permitido em seu computador. Isso geralmente é feito ao nível do sistema operacional e

varia de SO para SO. Se você estiver usando Lucene em um computador que usa um sabor de o sistema operacional UNIX, você pode ver o número máximo de arquivos abertos permitida a partir do linha de comando.

Em bater, Você pode ver as configurações atuais com o built-in `ulimit` comando:

```
% Ulimit-n
```

Em tcsh, O equivalente é

```
Limite de descritores%
```

Para alterar o valor sob bater, Use este comando:

```
% Ulimit-n número de <max aqui> arquivos abertos
```

Em tcsh, Use o seguinte:

```
% Limitar o número de descritores <max aqui> arquivos abertos
```

Para estimar uma definição para o número máximo de arquivos abertos enquanto o índice utilizado-

ing, tenha em mente que o número máximo de arquivos Lucene irá abrir em qualquer uma vez durante a indexação é

```
(1 + mergeFactor) FilesPerSegment *
```

Por exemplo, com um padrão `mergeFactor` de 10, ao criar um índice com 1000000 Documentos, Lucene exigirá a maioria dos arquivos abertos em um 88-unoptí Índice de vários minimizada com um único campo indexado. Chegamos a este número usando a seguinte fórmula:

```
11 segmentos / index * (7 files / segmento + 1 arquivo para campo indexado)
```

Se mesmo isso não elimina o problema de muitos arquivos abertos simultaneamente, e você estiver usando uma estrutura de indexação de vários arquivos, você deve converter o seu índice para a estrutura composta. Conforme descrito no apêndice B, isso irá fur- não reduzir o número de arquivos Lucene precisa abrir ao acessar o índice.

**NOTA** Se o seu computador está se esgotando de descritores de arquivos disponíveis, e seu índice não é otimizado, considere otimizando-o.

## 2.7.2 indexação em memória: RAMDirectory

Na seção anterior, mencionamos que Lucene faz buffer interno titulares de documentos recém-adicionado na memória antes de escrevê-las para o disco. Isso é feito automaticamente e de forma transparente quando você usa `FSDirectory`, Um arquivo-baseado Diretório implementação. Mas talvez você queira ter mais controle

sobre a indexação, o seu uso de memória, e a freqüência de lavagem a na memória buffer no disco. Você pode usar `RAMDirectory` como uma forma de em-memória buffer.

### RAMDirectory versus FSDirectory

Tudo o que `FSDirectory` faz no disco, `RAMDirectory` executa na memória, e assim é muito mais rápido. O código na listagem de 2,5 cria dois índices: um apoiado por um `FSDirectory` e a outra pelo `RAMDirectory`. Exceto por esta diferença, eles são idênticos, cada um contém 1.000 documentos com conteúdo idêntico.

#### RAMDirectory listagem 2,5 sempre fora executa FSDirectory

```
public class FSversusRAMDirectoryTest extends TestCase {

    fsDir Directory privado;
    ramDir Directory privado;
    Coleção privada docs = loadDocuments (3000, 5);

    setUp protected void () throws Exception {
        String = fsIndexDir
        System.getProperty ("java.io.tmpdir", "tmp") +
        System.getProperty ("File.separator") + "fs-index";

        ramDir = new RAMDirectory ();
        fsDir = FSDirectory.getDirectory (fsIndexDir, true);
    }

    testTiming public void () throws IOException {
        longo ramTiming = timeIndexWriter (ramDir);
        longo fsTiming = timeIndexWriter (fsDir);

        assertTrue (fsTiming > ramTiming);

        System.out.println ("Time RAMDirectory:" + (ramTiming) + "ms");
        System.out.println ("Time FSDirectory:" + (fsTiming) + "ms");
    }

    timeIndexWriter longo privado (Directory dir) throws IOException {
        longo start = System.currentTimeMillis ();
        addDocuments (dir);
        longa stop = System.currentTimeMillis ();
        retorno (stop - start);
    }

    addDocuments private void (Directory dir) throws IOException {
        Escritor IndexWriter IndexWriter = new (dir, novos SimpleAnalyzer (), true);

        / **

```

```

    / / Altere para ajustar o desempenho da indexação com FSDirectory
writer.mergeFactor = writer.mergeFactor;Parâmetros que
writer.maxMergeDocs = writer.maxMergeDocs;afetar o desempenho
                                de FSDirectorywriter.minMergeDocs = writer.minMergeDocs;
* /

for (Iterator iter = docs.iterator (); iter.hasNext ()) {
    Documento doc = new Document ();
    String palavra = (String) iter.next ();
    doc.add (Field.Keyword ("keyword", palavra));
    doc.add (Field.UnIndexed ("não indexados", palavra));
    doc.add (Field.UnStored ("UnStored", palavra));
    doc.add (Field.Text ("text", palavra));
    writer.addDocument (doc);
}
writer.optimize ();
writer.Close ();

}

privada loadDocuments Collection (int numDocs, int wordsPerDoc) {
    Coleccão docs = new ArrayList (numDocs);
    for (int i = 0; i < numDocs; i + +) {
        Doc = new StringBuffer StringBuffer (wordsPerDoc);
        for (int j = 0; j < wordsPerDoc; j + +) {
            doc.append ("Bibamus");
        }
        docs.add (doc.toString ());
    }
    retorno docs;
}

}

```

Embora existam maneiras melhores para a construção de benchmarks (ver secção 6.5 para uma exemplo de como você pode usar JUnitPerf para medir o desempenho do índice de busing), esta referência é suficiente para ilustrar a vantagem de desempenho que RAMDirectory tem mais FSDirectory. Se você executar o teste a partir da listagem 2.5 e gradivamente aumentar o valor de mergeFactor ou minMergeDocs, Você notará que o FSDirectoryIndexação baseada começa a se aproximar da velocidade do RAMDirectory-base um. No entanto, você também notará que não importa qual a combinação de parâmetros que você usa, o FSDirectoryÍndice baseado nunca supera a sua RAM-primo base.

Mesmo que você pode usar parâmetros de indexação para instruir Lucene para fundir segmentos em disco com menos freqüência, FSDirectoryIndexação baseada tem de escrevê-los para o disco acabou, que é a fonte da diferença de desempenho entre os dois Diretório implementações. RAMDirectory simplesmente nunca escreve nada

no disco. É claro, isso significa que uma vez que seu aplicativo será encerrado indexação, seu RAMDirectoryBaseados num índice está desaparecido.

### Indexação lote usando RAMDirectory como um buffer

Suponha que você queira melhorar o desempenho de indexação com o Lucene, e manipulando IndexWriter's mergeFactor, maxMergeDocs E minMergeDocs prova insuficiente. Você tem a opção de tomar o controle em suas próprias mãos, usando RAMDirectory buffer para escrever para um FSDirectoryÍndice baseado em si mesmo. Aqui está uma receita simples para fazer isso:

- 1 Crie uma FSDirectoryÍndice baseado.
- 2 Crie uma RAMDirectoryÍndice baseado.
- 3 Adicionar Documentos para o RAMDirectoryÍndice baseado.
- 4 Todo tantas vezes, tudo o flush no buffer RAMDirectory em FSDirectory.
- 5 Vá para a etapa 3. (Quem diz GOTO está morto?)

Podemos traduzir esta receita para a seguinte mistura de pseudocódigo e o reais Lucene use API:

```
FSDirectory fsDir = FSDirectory.getDirectory ("index / tmp /",
    true);
RAMDirectory ramDir = new RAMDirectory ();

FsWriter IndexWriter IndexWriter = (fsDir,
    nova SimpleAnalyzer (), true);
RamWriter IndexWriter IndexWriter = new (ramDir,
    nova SimpleAnalyzer (), true);

while (existem documentos para indexar) {
    ... criar documento ...
    ramWriter.addDocument (doc);

    if (condição para a lavagem de memória para o disco foi atingida) {
        fsWriter.addIndexes (Directory [] {} ramDir); Merge in-memory RAMDirectory
        ramWriter.close (); com em disco FSDirectory
        ramWriter IndexWriter = new (ramDir, novos SimpleAnalyzer (),
            true); Criar novas na memória
    } Buffer RAMDirectory
}
```

Esta abordagem dá-lhe a liberdade para flush Documentos com buffer na memória RAM para disco sempre que você escolher. Por exemplo, você poderia usar um contador que desencadeia lavagem após cada NDocumentos adicionados a um RAMDirectoryÍndice baseado. Da mesma forma, você poderia ter um timer que, periodicamente, força o flush, independentemente do número

de Documentos acrescentou. Uma abordagem mais sofisticada implicaria manter o controle de RAMDirectory's de memória, a fim de evitar RAMDirectory a partir de crescimento muito grande.

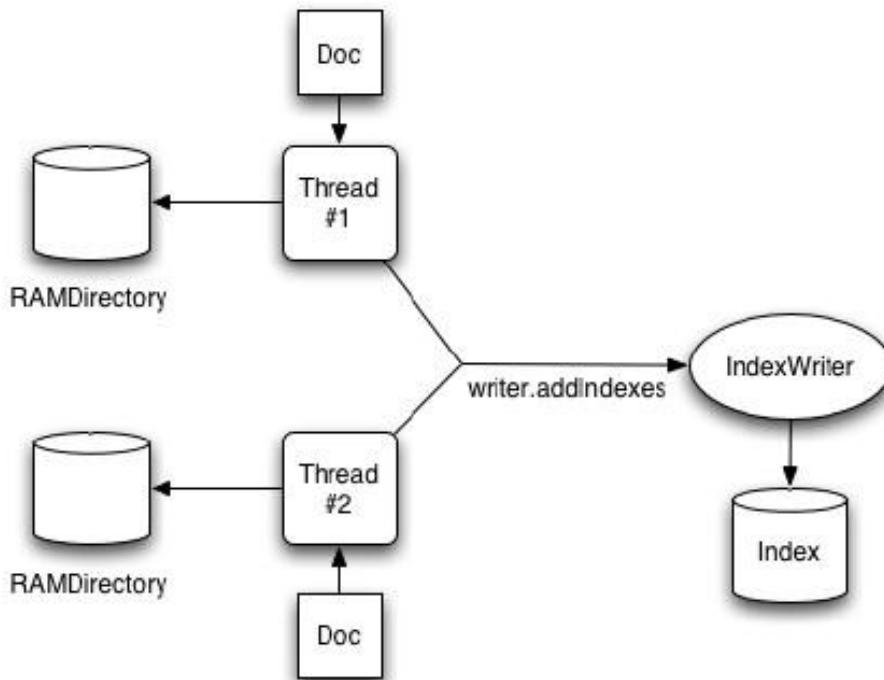
Qualquer que seja a lógica que você escolher, e eventualmente você vai usar IndexWriter's addIndexes

(Directory []) método para fundir o seu RAMDirectory índice baseado em um com o disco. Este método recebe um array de diretórios de qualquer tipo e funde-os todos em um único Diretório cuja localização é especificada na IndexWriter construtor.

#### Paralelização indexação, trabalhando com vários índices

A idéia de usar RAMDirectory como um buffer pode ser levada mais longe, como mostrado na figura 2.3. Você pode criar um aplicativo multithreaded indexação que usa múltiplos RAMDirectory baseada em índices em paralelo, um em cada thread, e se funde-los em um único índice no disco usando IndexWriter's addIndexes (Direc-história []) método.

Mais uma vez, quando e como você optar por sincronizar o tópicos e mesclar seus RAMDirectories para um único índice em disco é até você. Claro, se você tiver



A figura 2.3 aplicação multithreaded que utiliza múltiplos RAMDirectory instâncias para indexação paralelo.

vários discos rígidos, você também pode paralelizar os índices baseados em disco, já que o dois discos podem operar de forma independente.

E se você tiver vários computadores ligados a uma rede rápida, tais como Fiber Channel? Que, também, podem ser exploradas usando um conjunto de computadores como um cluster de indexação. A aplicação de indexação sofisticada poderia criar na memória ou do sistema de arquivos baseado em índices em vários computadores em paralelo e periodicamente envie seu índice para um servidor centralizado, onde todos os índices são fundidos em um Índice de grande porte.

A arquitetura na figura 2.4 tem duas falhas óbvias: o índice centralizado representa um ponto único de falha e é obrigado a se tornar um gargalo quando o número de nós de indexação aumenta. Independentemente, isto deve dar-lhe algumas ideias. Quando você aprender a usar o Lucene para realizar pesquisas sobre vários índices em paralelo e até mesmo fazê-lo remotamente (ver secção 5.6), você verá que Lucene permite criar muito grande de indexação distribuída e clusters busca.

Até agora, você pode ver claramente alguns padrões. RAM é mais rápida do disco: Se você necessidade de espremer mais fora do Lucene, use `RAMDirectory` para fazer a maioria de seu índice

ING mais rápido RAM. Minimizar funde índice. Se você tem recursos suficientes, como como múltiplas CPUs, discos ou mesmo computadores, paralelizar indexação e usar o `addIndexes (Directory [])` método para gravar em um único índice, que você deve eventualmente, construir e pesquisar. A fazer pleno uso desta abordagem, é preciso garantir que o segmento ou o computador que executa a indexação do disco é nunca ocioso, porque traduz o ócio ao tempo desperdiçado.

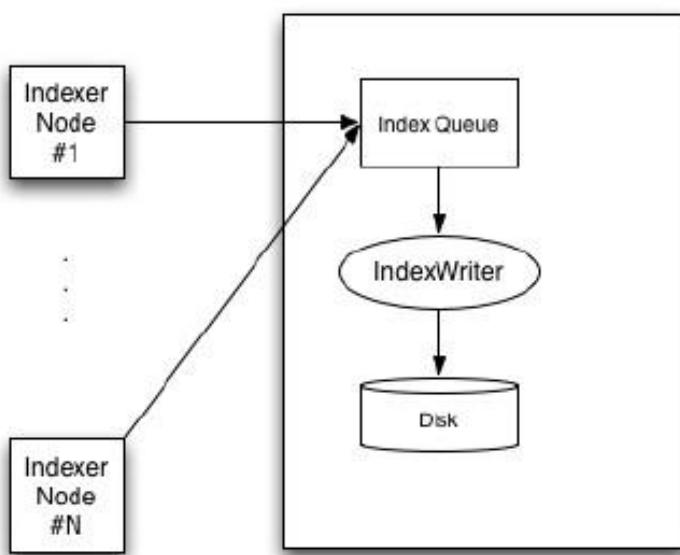


Figura 2.4  
Um cluster de nós indexador que enviam os seus pequenos índices para um grande servidor de indexação centralizado.

No ponto 3.2.3, discutimos o movimento na direção oposta: como obter um índice existente armazenados no sistema de arquivos em RAM. Este tópico é reservado para o capítulo sobre o pesquisar, pois a pesquisa é a razão mais adequada para levar um índice de sistema de arquivos em RAM.

### 2.7.3 Limitando o tamanho de campo: MaxFieldLength

Algumas aplicações indexar documentos cujos tamanhos não são conhecidos antecipadamente. Para con-

trole da quantidade de RAM e disco rígido de memória usados, eles precisam limitar o quantidade de entrada que índice. Outras aplicações lidar com documentos de conhecidos tamanho, mas deseja indexar apenas uma parte de cada documento. Por exemplo, você pode deseja indexar apenas a primeira de 200 palavras de cada documento. Lucene `IndexWriter` expõe `MaxFieldLength`, Uma variável de instância que permite que você programaticamente truncar documento muito grande `Campos`. Com um valor padrão de 10.000, Lucene índices apenas os primeiros 10.000 termos em cada `Field` documento. Isso efetivamente significa

que somente os primeiros 10 mil termos são relevantes para pesquisas, e qualquer texto para além do

décimo de milésimo termo não é indexado.

Para limitar Campo tamanhos de 1.000 termos, um aplicativo define `MaxFieldLength` para 1000, para praticamente eliminar o limite, um aplicativo deve definir `MaxFieldLength` para `Integer.MAX_VALUE`. O valor de `MaxFieldLength` podem ser alteradas a qualquer momento

durante a indexação, e a alteração tenha efeito para todos os adicionados posteriormente documentos. A mudança não é retroativa, portanto, qualquer campos já truncado devido a uma menor `MaxFieldLength` continuará a ser truncado. Listagem 2.6 mostra um exemplo concreto.

#### Listagem 2.6 tamanho do campo de controle com `MaxFieldLength`

```
public class FieldLengthTest estende TestCase {

    privado Diretório dir;
    privado String [] keywords = {"1", "2"};
    privado String [] unindexed = {"Paises Baixos", "Itália"};
    privado String [] UnStored = {"Amsterdam tem muitas pontes",
                                  "Veneza tem muitos canais"};
    private String [] {text = "Amsterdam", "Veneza"};

    setUp protected void () throws IOException {
        String = indexDir
        System.getProperty ("java.io.tmpdir", "tmp") +
        System.getProperty ("File.separator") + "index-dir";
        dir = FSDirectory.getDirectory (indexDir, true);
    }

    testFieldSize public void () throws IOException {
        addDocuments (dir, 10);Index 10 primeiros termos de cada Campo
    }
}
```

```

        assertEquals (1, getHitCount ("Conteúdo", "pontes"));

        addDocuments (dir, 1); Índice dimeiro mandato de cada Campo
        assertEquals (0, getHitCount ("Conteúdo", "pontes"));
    }

private int getHitCount (String fieldName, String searchString)
throws IOException {
    IndexSearcher searcher = new IndexSearcher (dir);
    Termo t = novo Termo (fieldName, searchString);
    Query query = new TermQuery (t);
    Hits hits = searcher.search (query);
    int HitCount hits.length = ();
    searcher.close ();
    retorno HitCount;
}

privada addDocuments void (Directory dir, MaxFieldLength int)
throws IOException {
    Escritor IndexWriter = new (dir, novos SimpleAnalyzer (),
        true);
    writer.maxFieldLength = MaxFieldLength; Número definido de
    for (int i = 0; i keywords.length <; i + +) {termos de índice
        Documento doc = new Document ();
        doc.add (Field.Keyword ("id", palavras-chave [i]));
        doc.add (Field.UnIndexed ("country", não indexados [i]));
        doc.add (Field.UnStored ("conteúdo", UnStored [i]));
        doc.add (Field.Text ("city" de texto, [i]));
        writer.addDocument (doc);
    }
    writer.optimize ();
    writer.Close ();
}
}

```

A partir desta lista, você ver como podemos limitar o número de Documento termos que índice:

- bf** Primeiro vamos instruir IndexWriter para indexar os primeiros 10 termos.
- c** Após os primeiros Documento é adicionado, somos capazes de encontrar uma correspondência para o termo pontes
- d** porque é o quinto mandato no documento contendo o texto "Amsterdam tem lotes de pontes".
- e** Nós reindexar esta Documento, Instruindo IndexWriter para indexar apenas o primeiro termo.
- f** Agora estamos incapazes de encontrar um Documento que continha o termo pontes porque Lucene indexados somente o primeiro termo, Amsterdam. O resto dos termos, incluindo pontes, foram ignorados.

## 2,8 Otimizando um índice

---

Índice otimização é o processo que mescla vários arquivos de índice em conjunto fim de reduzir o seu número e, assim, minimizar o tempo que leva para ler no índice no momento da pesquisa. Lembre-se da seção 2.7 que, embora seja adicionando novas Documentos

a um índice, Lucene vários buffers Documentos na memória antes de combiná-los em um segmento que escreve em um disco, opcionalmente fusão este novo segmento com criado anteriormente segmentos. Embora você pode controlar o segmento-MERGING processo com mergeFactor,maxMergeDocsE minMergeDocs, Quando a indexação é feito, você ainda pode ficar com vários segmentos no índice.

Pesquisando um índice composto de vários segmentos funciona corretamente, mas API do Lucene permite otimizar aí aí o índice e, assim, reduzir o Lucene consumo de recursos e melhorar o desempenho da pesquisa. Índice de otimização funde todos os segmentos de índice em um único segmento. Você pode otimizar um índice com uma única chamada para IndexWriter's otimizar () método. (Você pode ter notado, tais chamadas em listagens de código anterior, por isso vamos omitir uma lista separada aqui.) Índice de optimização envolve um monte de IO de disco, para usá-lo judiciosamente.

Figuras 2.5 e 2.6 mostram a diferença na estrutura do índice entre uma unoptimized e um índice de vários otimizado respectivamente.

É importante ressaltar que a otimização de um índice só afeta a velocidade das buscas contra esse índice, e não afeta a velocidade de indexação. Adição de novos Documentos para uma índice unoptimized é tão rápido como adicioná-los a um índice otimizado. O aumento em busca de desempenho vem do fato de que, com um índice otimizado, Lucene precisa abrir e processar arquivos menos de quando executar uma busca contra um unoptimized índice. Se você tomar um outro olhar para figuras 2.5 e 2.6, você pode ver que o índice otimizado tem arquivos de índice muito menor.

### Otimização de espaço em disco

Vale a pena mencionar que além de otimizar um índice, funde-Lucene existem segmentos ing, criando um segmento totalmente novo, cujo conteúdo, no final, representa o conteúdo de todos os segmentos de idade combinado. Assim, enquanto a otimização é em andamento, o uso de espaço em disco aumenta progressivamente. Quando terminar a criação de o novo segmento, Lucene descarta todos os segmentos de idade através da remoção de seus arquivos de índice.

Conseqüentemente, pouco antes de os segmentos antigos são removidos, o uso de espaço em disco um índice dobra porque tanto o novo segmento combinado unificado e todos os segmentos de idade estão presentes no índice. Após a otimização, o uso do disco índices cai de volta para o mesmo nível de antes de otimização. Tenha em mente que as regras do otimização índice de espera por índices de ambos os vários arquivos e compostos.

-rw-rw-r--	1	otis	otis	1000 Aug 25 12:45 _1pp.f1
-rw-rw-r--	1	otis	otis	1000 Aug 25 12:45 _1pp.f2
-rw-rw-r--	1	otis	otis	1000 Aug 25 12:45 _1pp.f3
-rw-rw-r--	1	otis	otis	130000 Aug 25 12:45 _1pp.fdt
-rw-rw-r--	1	otis	otis	8000 Aug 25 12:45 _1pp.fdx
-rw-rw-r--	1	otis	otis	39 Aug 25 12:45 _1pp.fnm
-rw-rw-r--	1	otis	otis	5558 Aug 25 12:45 _1pp.frq
-rw-rw-r--	1	otis	otis	11000 Aug 25 12:45 _1pp.prx
-rw-rw-r--	1	otis	otis	19 Aug 25 12:45 _1pp.tii
-rw-rw-r--	1	otis	otis	98 Aug 25 12:45 _1pp.tis
-rw-rw-r--	1	otis	otis	1000 Aug 25 12:45 _2kk.f1
-rw-rw-r--	1	otis	otis	1000 Aug 25 12:45 _2kk.f2
-rw-rw-r--	1	otis	otis	1000 Aug 25 12:45 _2kk.f3
-rw-rw-r--	1	otis	otis	130000 Aug 25 12:45 _2kk.fdt
-rw-rw-r--	1	otis	otis	8000 Aug 25 12:45 _2kk.fdx
-rw-rw-r--	1	otis	otis	39 Aug 25 12:45 _2kk.fnm
-rw-rw-r--	1	otis	otis	5558 Aug 25 12:45 _2kk.frq
-rw-rw-r--	1	otis	otis	11000 Aug 25 12:45 _2kk.prx
-rw-rw-r--	1	otis	otis	19 Aug 25 12:45 _2kk.tii
-rw-rw-r--	1	otis	otis	98 Aug 25 12:45 _2kk.tis
-rw-rw-r--	1	otis	otis	4 Aug 25 12:45 deletable
-rw-rw-r--	1	otis	otis	42 Aug 25 12:45 segments

Figura estrutura índice 2,5 de um índice de vários unoptimized mostrando vários segmentos em índice de um diretório

### Por que otimizar?

Embora os índices totalmente unoptimized executar na perfeição para a maioria das aplicações, aplicativos que lidam com grandes índices se beneficiarão do trabalho com otimizado índices. Ambientes que guardam referências a vários índices aberto para pesquisa irá beneficiar especialmente, porque o seu uso de índices totalmente otimizado vai requerem menos descritores de arquivos abertos.

Suponha que você está escrevendo um aplicativo de servidor que acabará por resultar em todos os

usuário tenha seu próprio índice em que novos documentos lentamente vai ser adicionado mais tempo. Como documentos são adicionados a cada índice, o número de segmentos em cada índice vai crescer também. Isto significa que enquanto procurava tais índices unoptimized, Lucene terá que manter as referências a um grande número de arquivos abertos, mas vai mesmo-

tualmente atingir o limite estabelecido pelo seu sistema operacional. Para auxiliar o situação, é

-rw-rw-r--	1	otis	otis	3000	Aug	25	12:43	_2kl.f1	
-rw-rw-r--	1	otis	otis	3000	Aug	25	12:43	_2kl.f2	
-rw-rw-r--	1	otis	otis	3000	Aug	25	12:43	_2kl.f3	
-rw-rw-r--	1	otis	otis	390000	Aug	25	12:43	_2kl.fdt	
-rw-rw-r--	1	otis	otis	24000	Aug	25	12:43	_2kl.fdx	
-rw-rw-r--	1	otis	otis		39	Aug	25	12:43	_2kl.fnm
-rw-rw-r--	1	otis	otis	16683	Aug	25	12:43	_2kl.frq	
-rw-rw-r--	1	otis	otis	33000	Aug	25	12:43	_2kl.prx	
-rw-rw-r--	1	otis	otis		19	Aug	25	12:43	_2kl.tii
-rw-rw-r--	1	otis	otis		98	Aug	25	12:43	_2kl.tis
-rw-rw-r--	1	otis	otis		4	Aug	25	12:43	deletable
-rw-rw-r--	1	otis	otis		25	Aug	25	12:43	segments

Figura estrutura índice 2,6 de um índice totalmente otimizado de vários mostrando um único segmento em índice de um diretório

deve desenvolver um sistema que permite uma otimização índice periódica. O mecanismo pode ser tão simples como ter um aplicativo independente que periodicamente itera sobre todos os seus usuários índices e executa o seguinte:

```
Escritor IndexWriter writer = new ("/ caminho / para / index",
    analisador, false);
writer.optimize ();
writer.Close ();
```

Claro que, se este é executado a partir de um aplicativo independente, você deve ter cuidado com modificação do índice simultâneas. Um índice deve ser modificado apenas por um único processo do sistema operacional de cada vez. Em outras palavras, apenas um único processo deve índice aberto com `IndexWriter` ao mesmo tempo. Como você verá nas demais seções do Neste capítulo, Lucene usa um mecanismo de travamento baseado em arquivo para tentar evitar que isso tipo de modificação do índice simultâneas.  
Quando a optimizar

Apesar de um índice pode ser otimizada por um único processo a qualquer momento durante indexação, e fazendo isso não irá danificar o índice ou torná-lo indisponível para buscas, otimização de um índice ao executar operação de indexação não é recomendadas. É melhor para otimizar um índice apenas no final, quando você sabe que o índice permanecerá inalterado por um tempo. Otimizando durante a indexação será só fazem indexação demorar mais tempo.

**NOTA** Ao contrário de uma crença popular, otimizando um índice não melhorar velocidade de indexação. Otimizando um índice de melhora apenas a velocidade de busca por minimizar o número de arquivos de índice que precisam ser abriu, processados e pesquisados. Otimizar um índice apenas no final de o processo de indexação, quando você sabe o índice permanecerá inalterado por um tempo.

## 2.9 Concorrência, fio de segurança, e problemas de bloqueio

---

Nesta seção, abordamos três temas intimamente relacionados: o acesso índice de concorrentes, thread de segurança da `IndexReader` e `IndexWriter`, E o mecanismo de bloqueio que Lucene usa para prevenir a corrupção do índice. Estas questões são muitas vezes incompreendido por novos usuários para Lucene. Compreensão destes tópicos é importante, porque vai eliminar surpresas que podem resultar quando seu aplicativo de indexação começa a servir vários usuários simultaneamente, ou quando tem que lidar com uma súbita necessidade de escala paralelização por algumas de suas operações.

### 2.9.1 regras Concurrency

Lucene fornece várias operações que podem modificar um índice, como o documento indexação, atualização e exclusão, quando usá-los, você precisa seguir certos regras para evitar a corrupção do índice. Estas questões levantam suas cabeças freqüentemente em web aplicações, onde várias solicitações são normalmente tratados simultaneamente. Lucene regras de concorrência são simples, mas devem ser rigorosamente seguidas:

- Qualquer número de operações somente leitura podem ser executadas simultaneamente. Para exemplo, vários segmentos ou processos podem pesquisar o mesmo índice em paralelo.
- Qualquer número de operações somente leitura pode ser executado enquanto um índice é sendo modificado. Por exemplo, os usuários podem pesquisar um índice enquanto ele está sendo otimizado ou enquanto novos documentos estão sendo adicionados ao índice, atualizado ou excluído do índice.
- Apenas uma operação de índice de modificação simples pode executar de cada vez. Um índice deve ser aberto por um único `IndexWriter` ou um único `IndexReader` ao mesmo tempo.

Com base nessas regras de concorrência, podemos criar um conjunto mais abrangente de exemplos, mostrados na tabela 2.2. Essas regras representam o permitido e não permitido operações simultâneas em um único índice.

Tabela 2.2 Exemplos de permitidos e não permitidos operações simultâneas realizadas em um único Lucene índice

Operação	Permitidos ou proibidos
Execução de vários busca simultâneas contra o mesmo índice	Permitido
Execução de vários busca simultâneas contra um índice que está sendo construído, documentos otimizado, ou se fundir com outro índice, ou cujo estão sendo apagados ou atualizados	Permitido
Adição ou atualização de documentos no mesmo índice usando várias instâncias de <code>IndexWriter</code>	Não permitido
Não para fechar o <code>IndexReader</code> que foi usado para apagar documentos de um índice antes de abrir um novo <code>IndexWriter</code> para adicionar mais documentos para o mesmo índice	Não permitido
Não para fechar o <code>IndexWriter</code> que foi usado para adicionar documentos a um índice antes de abrir um novo <code>IndexReader</code> para excluir ou atualizar documentos do mesmo índice	Não permitido

**NOTA** Quando você estiver executando operações que modificam um índice, manter sempre em mente que apenas uma operação de índice de modificação deve ser executado no mesmo índice de cada vez.

### 2.9.2 Tópicos de segurança

É importante saber que, apesar de fazer modificações índice simultânea com várias instâncias do `IndexWriter` ou `IndexReader` não é permitido, como mostrado na tabela 2.2, ambas as classes são thread-safe. Portanto, um única instância de uma ou outra classe pode ser compartilhado entre vários segmentos, e todas as chamadas ao seu índice de modificação métodos serão devidamente sincronizados, de modo que as modificações do índice são executados um após o outro. Figura 2.7 retrata um cenário como esse.

Sincronização externa adicional é desnecessário. Apesar do fato de que ambos classes são thread-safe, um aplicativo usando Lucene deve assegurar que o índice de operações de modificação destas duas classes não se sobreponem. Isto é, antes de acrescentando novos documentos a um índice, você deve fechar todos os `IndexReader` instâncias que foram excluídos Documentos do mesmo índice. Da mesma forma, antes de excluir ou atualizar documentos em um índice, você deve fechar o `IndexWriter` exemplo, que abriu que antes mesmo índice.

A matriz de simultaneidade na tabela 2.3 dá uma visão geral de operações que pode ou não pode ser executado simultaneamente. Assume-se que uma única instância `IndexWriter` ou uma única instância de `IndexReader` é usado. Note que não lista

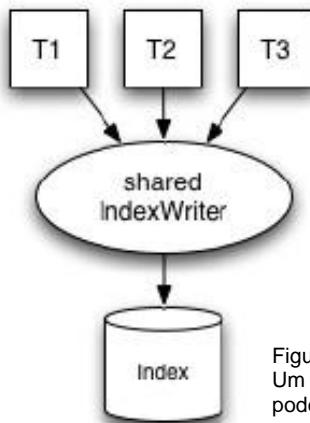


Figura 2.7  
Um único IndexWriter ou IndexReader pode ser compartilhado por vários segmentos.

atualização como uma operação separada, porque uma atualização é realmente uma operação de exclusão

seguido por uma operação de adição, como você viu na seção 2.2.4

Tabela matriz Concurrency 2,3 quando a mesma instância IndexWriter ou IndexReader é usado.  
Interseções Marcado significam operações que não podem ser executadas simultaneamente.

	Pergunta	Ler o documento	Adicionar	Excluir	Optimize	Fundir
Pergunta						
Ler o documento						
Adicionar				X		
Excluir			X		X	X
Optimize				X		
Fundir				X		

Esta matriz pode ser resumido da seguinte forma:

- Um documento não pode ser adicionado (`IndexWriter`), Enquanto um documento está sendo excluído (`IndexReader`).
- Um documento não pode ser excluído (`IndexReader`), Enquanto o índice está sendo otimizado (`IndexWriter`).
- Um documento não pode ser excluído (`IndexReader`), Enquanto o índice está sendo fundidas (`IndexWriter`).

Da matriz e seu resumo, você pode ver um padrão: um índice de modificação `IndexReader` operação não pode ser executado enquanto um índice de modificação `IndexWriter` operação está em andamento. Esta regra é simétrico: Um índice de modificação `IndexWriter` operador não pode ser executado enquanto um índice de modificação `IndexReader` operação está em andamento.

Você pode pensar destas regras de concorrência Lucene como análogas às regras do boas maneiras e conduta adequada e legal na nossa sociedade. Embora estas regras não têm de ser rigorosamente seguidas, não segui-las pode ter repercussões. Em vida real, quebrando uma regra pode aterrá-lo na prisão; no mundo do Lucene, poderia corromper seu índice. Lucene antecipa mau uso e até mesmo mal-entendido problemas de concorrência, por isso ele usa um mecanismo de travamento para fazer o seu melhor para evitar inadecção do índice vertente. Lucene mecanismo de travamento do índice é descrito na próxima seção.

### 2.9.3 Índice de bloqueio

Relacionadas com as questões de concorrência no Lucene é o tema do bloqueio. Para evitar inadecção do índice de utilização indevida de sua API, Lucene cria baseado em arquivo travas em todas as segmentos de código que devem ser executadas por um único processo de cada vez. Cada índice tem seu próprio conjunto de arquivos de bloqueio, por padrão, todos os arquivos de bloqueio são criados em um computador tem diretório temporário, conforme especificado pelo Java `java.io.tmpdir` propriedade do sistema.

Se você olhar para esse diretório, enquanto documentos de indexação, você verá Lucene `write.lock` arquivo, se você pegar Lucene enquanto ele está fundindo segmentos, você vai notar a `commit.lock` arquivo, também. Você pode alterar o diretório de bloqueio, definindo o `org.apache.lucene.lockDir` propriedade do sistema para o diretório desejado. Este sistema propriedade pode ser definida programaticamente usando uma API Java, ou pode ser definido a partir do

linha de comando usando `-Dorg.apache.lucene.lockDir = / caminho / para / lock / dir` sintaxe. Se

você tiver vários computadores que precisam acessar o mesmo índice armazenados em um disco compartilhado, você deve definir o diretório bloqueio explicitamente para que as aplicações em

ver computadores diferentes uns dos outros bloqueios. Por causa de problemas conhecidos com arquivos de bloqueio

e NFS, escolha um diretório que não reside em um volume NFS. Aqui está o que ambas as fechaduras podem olhar como:

```
lucene-de61b2c77401967646c18916982a09a0-write.lock
lucene-de61b2c77401967646cf8916982a09a0-commit.lock
```

O arquivo `write.lock` é usado para manter os processos de simultaneamente tentar modificar um índice. Mais precisamente, o `write.lock` é obtido pela `IndexWriter` quando `IndexWriter` é instanciado e mantida até que seja fechada. O arquivo de bloqueio mesmo também é

obtidos por `IndexReader` quando é usada para apagar Documentos, undeleting eles, ou a criação Campo normas. Como tal, tende a `write.lock` bloquear o índice para escrever para longos períodos de tempo.

O `commit.lock` é usado sempre que os segmentos estão sendo lidos ou mescladas. É obtido por um `IndexReader` antes que ele lê o arquivo de segmentos, que todos os nomes segmentos de índice, e é liberado somente após `IndexReader` abriu e leu todos os os segmentos referenciados. `IndexWriter` também obtém o direito `commit.lock` antes ele cria um arquivo de novos segmentos e mantém até que ele remove os arquivos de índice que ter sido tornada obsoleta pela operações como fusões segmento. Assim, a `commit.lock` podem ser criadas com mais freqüência do que o `write.lock`, mas nunca deve bloquear o índice por muito tempo uma vez que durante a sua existência arquivos de índice são apenas aberto ou excluída e apenas um arquivo de pequenos segmentos são gravados no disco. Tabela 2.4 resume todos os manchas na API Lucene que travam um índice.

Tabela 2.4 Um resumo de todos os bloqueios Lucene e operações que criam e liberá-los

File Lock	Classe	Obtidos em	Lançado em	Descrição
<code>write.lock</code>	<code>IndexWriter</code>	Construtor	<code>close ()</code>	Bloqueio liberado quando <code>IndexWriter</code> está fechado
<code>write.lock</code>	<code>IndexReader</code>	<code>delete (int)</code>	<code>close ()</code>	Bloqueio liberado quando <code>IndexReader</code> está fechado
<code>write.lock</code>	<code>IndexReader</code>	<code>undeleteAll (int)</code>	<code>close ()</code>	Bloqueio liberado quando <code>IndexReader</code> está fechado
<code>write.lock</code>	<code>IndexReader</code>	<code>setNorms (Int, String, byte)</code>	<code>close ()</code>	Bloqueio liberado quando <code>IndexReader</code> está fechado
<code>commit.lock</code>	<code>IndexWriter</code>	Construtor	Construtor	Bloqueio liberado tão logo informação por segmentos é lido ou escrito
<code>commit.lock</code>	<code>IndexWriter</code>	<code>addIndexes (IndexReader [])</code>	<code>addIndexes (IndexReader [])</code>	Bloqueio obtido enquanto o novo segmento está escrito
<code>commit.lock</code>	<code>IndexWriter</code>	<code>addIndexes (Directory [])</code>	<code>addIndexes (Directory [])</code>	Bloqueio obtido enquanto o novo segmento está escrito
<code>commit.lock</code>	<code>IndexWriter</code>	<code>mergeSegments (Int)</code>	<code>mergeSegments (Int)</code>	Bloqueio obtido enquanto o novo segmento está escrito
<code>commit.lock</code>	<code>IndexReader</code>	<code>open (Diretório)</code>	<code>open (Diretório)</code>	Bloqueio obtido até que todos os segmentos são lidos

continua na página seguinte

Tabela 2.4 Um resumo de todos os bloqueios Lucene e operações que criam e liberá-los (Continuação)

File Lock	Classe	Obtidos em	Lançado em	Descrição
commit.lock	SegmentReader	doClose ()	doClose ()	Bloqueio obtido enquanto o arquivo de segmento está escrito ou reescrito
commit.lock	SegmentReader	undeleteAll ()	undeleteAll ()	Bloqueio obtido enquanto o segmento. ficheiro del é removidas

Você deve estar ciente de dois métodos adicionais relacionados com bloqueio:

- `IndexReader's isLocked (Diretório)` Informa-se o índice especificado em seu argumento está bloqueado. Este método pode ser útil quando um aplicativo precisa verificar se o índice está bloqueado antes de tentar uma das índice de modificação operações.
- `IndexReader's desbloquear (Diretório)` -Não exatamente o que seu nome implica. Embora este método dá-lhe poder para desbloquear qualquer índice do Lucene, em qualquer tempo, utilizá-lo é perigoso. Lucene cria bloqueios para uma boa razão, e desbloquear um índice enquanto ele está sendo modificado pode resultar em um corrupto e índice inutilizável.

Embora você já sabe que os arquivos de lock usa Lucene, quando usa-los, por que usa-los, e onde eles estão armazenados no sistema de arquivos, você deve resistir a tocar -los. Além disso, você deve sempre contar com a API Lucene para manipulá-los. Se você não fizer isso, seu código pode quebrar se Lucene começa a usar um bloqueio diferentes mecanismo, no futuro, ou mesmo se ela muda o nome ou a localização dos arquivos de seu bloqueio.

### Bloqueio em ação

Para demonstrar bloqueio, listando 2.7 fornece um exemplo de uma situação onde Lucene usa bloqueios para impedir múltiplas índice modificando a execução de operações contra o mesmo índice simultaneamente. No `testWriteLock ()` método, Lucene bloqueia a segunda `IndexWriter` abertura de um índice que já foi aberto por outro `IndexWriter`. Este é um exemplo de write.lock em ação.

### Listagem 2.7 Usando baseado em arquivo bloqueios para evitar a corrupção de

```
public class LockTest extends TestCase {
    private Directory dir;
    setUp protected void () throws IOException {
```

```

String = indexDir
    System.getProperty ("java.io.tmpdir", "tmp") +
    System.getProperty ("File.separator") + "index";
dir = FSDirectory.getDirectory (indexDir, true);

}

public void testWriteLock () throws IOException {
    IndexWriter writer1 = null;
    IndexWriter writer2 = null;

    try {
        writer1 = IndexWriter novo (dir, novos SimpleAnalyzer (), true);
        writer2 IndexWriter = new (dir, novos SimpleAnalyzer (), true);

        falha ("Nunca devemos chegar a este ponto");
    }
    catch (IOException e) {
        e.printStackTrace ();
    }
    finally {
        writer1.close ();
        assertNull (writer2);
    }
}

testCommitLock public void () throws IOException {
    IndexReader reader1 = null;
    IndexReader reader2 = null;

    try {
        Escritor IndexWriter IndexWriter = new (dir, novos SimpleAnalyzer (),
            true);
        writer.Close ();
        reader1 = IndexReader.open (dir);
        reader2 = IndexReader.open (dir);
    }
    finally {
        reader1.close ();
        reader2.close ();
    }
}
}

```

O `testCommitLock ()` método demonstra o uso de um `commit.lock` que é obtido em `IndexReader's open (Diretório)` método e liberado pelo mesmo método, assim como todos os segmentos de índice foram lidos. Porque o bloqueio é liberado pelo mesmo método que obteve, somos capazes de acessar o mesmo diretório com o segundo `IndexReader` mesmo antes de o primeiro ter sido fechado. (Você pode se perguntar

sobre o `IndexWriter` você vê neste método: Seu único propósito é semear o índice de criando o arquivo segmentos necessários, que contém informações sobre todos os existentes índices. Sem o arquivo de segmentos `IndexReader` estaria perdido, porque não saberia quais os segmentos de ler a partir do diretório índice.)

Quando executamos esse código podemos ver um rastreamento de pilha de exceção causada pela bloqueado índice, que assemelha-se ao rastreamento de pilha a seguir:

```
java.io.IOException: Lock obter expirou
    em org.apache.lucene.store.Lock.obtain (Lock.java: 97)
    em
    . org.apache.lucene.index.IndexWriter <init> (IndexWriter.java: 173)
    em lia.indexing.LockTest.writeLock (LockTest.java: 34)
```

Como mencionamos anteriormente, os novos usuários do Lucene, por vezes, não têm uma boa compreensão das questões de concorrência descrito nesta seção e consequentemente correr em problemas de bloqueio, como o show um no rastreamento de pilha anterior. Se você ver as exceções similar em seus aplicativos, por favor, não ignorá-las, se a consistência dos índices é de todo importante para você. Lock-relacionados exceções são geralmente um sinal de um mau uso da API Lucene, se eles ocorrem em sua aplicação, você deve resolvê-los prontamente.

#### 2.9.4 bloqueio índice de desativação

Desaconselhamos fortemente intromissão com mecanismo de travamento Lucene e desconsiderando sistência à exceção lock-relacionados. No entanto, em algumas situações você pode querer desativar o bloqueio no Lucene, e fazendo isso não irá danificar seu índice. Por exemplo, sua aplicação pode precisar acessar um índice Lucene armazenado em um CD-ROM. Um CD é um meio de somente leitura, o que significa que sua aplicação estará operando em uma leitura apenas o modo, também. Em outras palavras, o aplicativo estará usando Lucene apenas para pesquisar o índice e não irá modificar o índice de qualquer forma. Embora já Lucene armazena os arquivos de bloqueio em temporário do sistema diretório um diretório normalmente aberto para escrever por qualquer usuário do sistema, você pode desativar tanto `write.lock` e se comprometer. bloco, definindo o `disableLuceneLocks` propriedade do sistema para a cadeia "true".

### 2,10 indexação Depuração

Vamos discutir uma final, característica Lucene bastante desconhecida (se assim podemos chamá-lo). Se você sempre necessidade de depuração do índice Lucene processo de escrita, lembre-se que você pode começar Lucene à informação sobre as operações de saída de sua indexação através da criação `Índice EscritorVariável`'s instância pública `Infostream` a um dos `OutputStreams`, como `System.out`:

```
Escriptor IndexWriter IndexWriter = new (dir, novos SimpleAnalyzer (),  
    true);  
writer.infoStream = System.out;  
...
```

Isso revela informações sobre fundo segmento, como mostrado aqui, e pode ajudar parâmetros de indexação que você tune descrito anteriormente neste capítulo:

```
segmentos fusão 0 (1 docs) 1 (1 docs) 2 (1 docs)  
_3 (1 docs) _4 (1 docs) _5 (1 docs) _6 (1 docs)  
_7 (1 docs) _8 (1 docs) _9 (1 docs) em a (10 documentos)  
fusão segmentos B (1 docs) c (1 docs) D (1 docs)  
e (1 docs) f (1 docs) g (1 docs) h (1 docs)  
i (1 docs) J (1 docs) k (1 docs) em l (10 documentos)  
fusão segmentos m (1 docs) n (1 docs) o (1 docs)  
p (1 docs) condutores Q (1 docs) r (1 docs) s (1 docs)  
t (1 docs) u (1 docs) V (1 docs) em w (10 documentos)
```

Além disso, se você precisa peek dentro de seu índice uma vez que é construída, você pode usar Lucas: uma ferramenta de terceiros útil que discutimos na seção 8.2, página 269.

## Resumo 2,11

---

Este capítulo lhe deu uma sólida compreensão de como um índice Lucene opera-Ates. Além de adicionar Documentos a um índice, você deve ser capaz de remover e atualizar indexados Documentos, bem como manipular um par de índice fatores ing para ajustar vários aspectos da indexação para atender às suas necessidades. O conhecimento sobre a concorrência, fio de segurança, e bloqueio é essencial se você está usando o Lucene em um aplicativo multithread ou um sistema multiprocessado. Até agora você deve estar morrendo de vontade de aprender a busca com Lucene, e é isso que você vai leia sobre no próximo capítulo.

# Adicionando pesquisa para sua aplicação

Este capítulo aborda

- Consultando um índice Lucene
- Trabalhando com os resultados da pesquisa
- Compreensão de pontuação Lucene
- Análise humana entrou expressões de consulta

Se não podemos encontrá-lo, ele efetivamente não existe. Mesmo que tenhamos indexados documentos, nosso esforço é desperdiçado se não compensa, fornecendo uma maneira confiável e rápido para encontrar esses documentos. Por exemplo, considere o seguinte cenário:  
 Dê-me uma lista de todos os livros publicados nos últimos 12 meses sobre o tema da "Java", onde "open source" ou "Jakarta" é mencionado no conteúdo. Restringir os resultados para apenas livros que estão no especial. Ah, e debaixo das cobertas, também assegurar que os livros que citam "Apache" são apanhados, porque nós explicitamente especificado "Jakarta". E torná-lo mal-humorado, na ordem de milissegundos para resposta tempo.<sup>1</sup>

Você tem um repositório de centenas, milhares ou milhões de documentos que necessidades capacidade de pesquisa semelhante?

Fornecimento de capacidade de busca usando a API Lucene é simples e fácil, mas espreitando debaixo das cobertas é um sofisticado mecanismo que pode atender às suas requisitos de busca como o retorno dos documentos mais relevantes primeiro e recuperar os resultados incrivelmente rápido. Este capítulo aborda formas comuns de pesquisa usando a API Lucene. A maioria das aplicações usando a busca Lucene pode provide um recurso de busca que executa muito bem usando as técnicas mostradas neste capítulo. Capítulo 5 investiga mais recursos avançados de pesquisa, e capítulo 6 elabora sobre as formas de estender classes Lucene para poder pesquisar ainda maior.

Começamos com um exemplo simples mostrando que o código que você escreve para implemento de pesquisa é geralmente não mais que algumas linhas de comprimento. Em seguida, ilustrar o

scoring fórmula, proporcionando um olhar profundo em um dos mais Lucene é especial atributos. Com este exemplo e um entendimento de alto nível de como Lucene resultados fileiras de busca, nós vamos, então, explorar os vários tipos de consultas de pesquisa Lucene trata nativamente.

### 3.1 Implementação um recurso de pesquisa simples

Suponha que você esteja encarregado com a adição de pesquisa para um aplicativo. Você abordados get-

ting os dados indexados, mas agora é hora de expor o texto completo buscando o usuários finais. É difícil imaginar que a adição de pesquisa poderia ser mais simples do que é com Lucene. Obtenção de resultados de pesquisa requer apenas algumas linhas de código, literalmente.

Lucene fornece acesso fácil e altamente eficiente para os resultados da pesquisa, também, de livre

ing que você se concentre sua lógica de aplicação e interface com o usuário em torno desses

<sup>1</sup> Nós cobriremos as peças para fazer isso acontecer com Lucene, incluindo um filtro specials no capítulo 6, syn-injeção onym no capítulo 4, ea lógica booleana neste capítulo.

Neste capítulo, vamos limitar nossa discussão para as classes primárias na API do Lucene que você normalmente usa para integração da pesquisa (como mostrado na tabela 3.1). Claro, há mais para a história, e vamos além do básico nos capítulos 5 e 6. Neste cap-  
ter, nós vamos cobrir os detalhes que você precisa para a maioria de suas aplicações.

Tabela 3.1 API do Lucene procura primária

Classe	Propósito
IndexSearcher	Porta de entrada para a pesquisa em um índice. Todas as pesquisas vêm através de um <code>IndexSearcher</code> exemplo, usando qualquer um dos vários sobrecarregado <code>pesquisa</code> métodos.
Consulta (e subclasses)	Subclasses concretas encapsular a lógica de um tipo de consulta particular. Instâncias de <code>Pergunta</code> são passados para um <code>IndexSearcher</code> busca de método.
QueryParser	Processos de um ser humano a entrar expressão (e legível) em um concreto <code>Pergunta</code> objeto.
Hits	Fornecere acesso a resultados de busca. <code>Hits</code> é retornado de <code>IndexSearcher</code> de <code>pesquisa</code> método.

Quando você está consulta um índice Lucene, uma coleção ordenada de hits é retornado. A coleção de hits é ordenada por partitura por default.<sup>2</sup> Lucene calcula uma pontuação (a valor numérico de relevância) para cada documento, dada uma consulta. O bate-los-mesmos não são os documentos reais de correspondência, mas são as referências ao documentos encontrados. Na maioria das aplicações que são apresentados os resultados da pesquisa, os usuários  
acessar apenas os primeiros documentos poucos, por isso não é necessário recuperar o real documents para todos os resultados, você precisa recuperar somente os documentos que serão pre-apresentados ao usuário. Para índices grandes, não seria sequer possível coletar todos os correspondentes documentos disponíveis na memória do computador físico.

Na próxima seção, nós colocamos `IndexSearcher`, `Pergunta` E `Hits` de trabalhar com alguns básicos buscas prazo.

### 3.1.1 Procurar um termo específico

`IndexSearcher` é a classe central usado para procurar documentos em um índice. Ele tem vários métodos de pesquisa sobrecarregado. Você pode procurar por um determinado prazo utilização  
o mais comumente usado `pesquisa` método. Um termo é um valor que está emparelhado com o seu  
contendo campo de nome, neste caso, assunto.

<sup>2</sup> A palavra coleção neste sentido se não referem-se a `java.util.Collection`.

**NOTA** Importante: O texto original pode ter sido normalizada em termos de o analisador, o que pode eliminar os termos (como palavras de parada), converter termos em letras minúsculas, converter termos de formas de palavras base (decorrentes), ou inserir termos adicionais (sinônimo processamento). É crucial que os termos passaram para `IndexSearcher` ser consistente com os termos produzido pela análise de os documentos de origem. O Capítulo 4 discute o processo de análise em detalhe.

Usando nosso exemplo de índice de dados de livros, vamos de consulta para as palavras formiga e junit, que são palavras que conhecemos foram indexadas. Listagem 3.1 executa uma consulta prazo e afirma que o único documento esperado é encontrado. Lucene fornece vários embutido Pergunta tipos (ver secção 3.4), `TermQuery` sendo o mais básico.

Listagem 3.1 SearchingTest: Demonstra a simplicidade de busca utilizando um `TermQuery`

```
public class SearchingTest estende LiaTestCase {

    testTerm public void () throws Exception {
        IndexSearcher searcher = new IndexSearcher (diretório);
        Termo t = novo Termo ("sujeito", "ant");
        Query = new TermQuery (t);
        Hits hits = searcher.search (query);
        assertEquals ("JDwA", 1, hits.length ());

        t = novo Termo ("subject", "junit");
        hits = searcher.search (novo TermQuery (t));
        assertEquals (2, hits.length ());

        searcher.close ();
    }
}
```

A `Hits` objeto é retornado da nossa pesquisa. Vamos discutir este objeto na seção 3.2, mas por agora apenas uma nota que o `Hits` objeto encapsula o acesso à base Documentos. Este encapsulamento faz sentido para um acesso eficiente aos documentos. Completo documentos não são imediatamente devolvidos, eles são buscados na demanda. Nesta exemplo, nós não nos preocuparmos com os documentos reais associados os hits retornado, pois estávamos interessados apenas em afirmar que o bom número de documentos foram encontrados.

Em seguida, vamos discutir como transformar um usuário entrou em uma expressão de consulta Pergunta objeto.

### 3.1.2 Análise de um usuário entrou expressão de consulta: QueryParser

Duas características mais redondo para fora o que a maioria das aplicações requerem busca: análise de expressão sofisticada de consulta e acesso aos documentos devolvidos.

Métodos de busca do Lucene exigir um `Pergunta` objeto. Análise uma expressão de consulta é a ato de transformar uma introduzidos pelo utilizador de consulta, como "mock OU junit" para uma adequada

`Pergunta` instância do objeto; 3 neste caso, o `Pergunta` objeto seria uma instância de `BooleanQuery` com duas cláusulas nonrequired, um para cada termo. O seguinte código analisa duas expressões de consulta e afirma que eles trabalharam como esperado.

Depois

retornando a hits, nós recuperamos o título do primeiro documento encontrado:

```
testQueryParser public void () throws Exception {
    IndexSearcher searcher = new IndexSearcher (diretório);

    Query = QueryParser.parse ("+ JUnit + ANT-MOCK",
        "Conteúdo",
        nova SimpleAnalyzer ());
    Hits hits = searcher.search (query);
    assertEquals (1, hits.length ());
    Document d = hits.doc (0);
    assertEquals ("Java Development com Ant", d.get ("title"));

    query = QueryParser.parse ("mock OU junit",
        "Conteúdo",
        nova SimpleAnalyzer ());
    hits = searcher.search (query);
    assertEquals ("JDwA e AIJ", 2, hits.length ());
}
```

Lucene inclui uma característica interessante que analisa as expressões de consulta através do `QueryParser` classe. Ele analisa expressões ricos, como os mostrados dois ("+ JUnit + ANT -MOCK " e "Mock OU junit") Em uma das `Pergunta` implementações. Lidando com humanos e entrou consultas é o principal objetivo do `QueryParser`.

`QueryParser` requer um analisador para quebrar pedaços da consulta em termos. No primeira expressão, a consulta foi inteiramente maiúscula. Os termos do conteúdo campo, no entanto, foram em maiúsculo quando indexados. `QueryParser`, Neste exemplo, usa `SimpleAnalyzer`, Que minúsculas os termos antes de construir um `Pergunta` objeto. (Análise é coberto em grande detalhe no próximo capítulo, mas é intimamente intertorcido com o texto indexação e pesquisa com `QueryParser`.) O ponto principal com relação a análise a considerar neste capítulo é que você precisa ter a certeza de consulta sobre os termos reais indexados. `QueryParser` é a única busca que usa um

---

<sup>3</sup> Expressões de consulta são semelhantes às expressões SQL usado para consultar um banco de dados em que a expressão deve ser analisado em algo em um nível inferior que o servidor de banco de dados pode entender diretamente.

analisador. Consultando através da API usando `TermQuery` e os outros discutidos em seção 3.4 não usa um analisador, mas se baseia em termos de correspondência com o que foi indexados. Na seção 4.1.2, falamos mais sobre as interações de `QueryParser` e o processo de análise.

Equipado com os exemplos mostrados até agora, você está mais do que pronto para começar pesquisar seus índices. Há, é claro, muitos mais detalhes para saber sobre busca. Em particular, `QueryParser` requer explicação adicional. Seguinte é uma visão geral de como usar `QueryParser`. Que voltamos a em maior detalhe mais tarde, em neste capítulo.

### Usando `QueryParser`

Antes de mergulhar nos detalhes de `QueryParser` (O que fazemos na seção 3.5), vamos primeiro olhar como ele é usado em um sentido geral. `QueryParser` tem uma estática `parse ()` método para permitir o uso mais simples. Sua assinatura é

```
Consulta public static
    parse (String query, campo String, Analyzer analisador)
        throws ParseException
```

O `String query` é a expressão a ser analisada, como "cão + + gato". O segundo parâmetro `campo`, é o nome do campo padrão para associar com termos em a expressão (mais sobre isso na seção 3.5.4). O argumento final é uma `Analizador` exemplo. (Discutimos analisadores em detalhe no próximo capítulo e, em seguida, cobrir o interações entre `QueryParser` e analisador a na seção 4.1.2.) A `testQueryParser ()` método mostrado na seção 3.1.2 demonstra o uso da estática `parse ()` método.

Se a expressão não para analisar, uma `ParseException` é lançada, uma condição que sua aplicação deve lidar de forma graciosa. `ParseException` "Mensagem de s dá uma indicação razoável do porque a análise falhou, no entanto, essa descrição pode ser demasiado técnico para usuários finais.

A estática `parse ()` método é rápido e conveniente de usar, mas não pode ser suficientes. Nos bastidores, o método `static` instancia uma instância de `Consulta-Analisador` e invoca o exemplo `parse ()` método. Você pode fazer a mesma coisa si mesmo, o que lhe dá um nível mais refinado de controle. Existem várias opções que pode ser controlada em um `QueryParser` instância, como o operador padrão (que padrões para OR). Essas configurações também incluem locale (para análise de data), padrão slop frase, e se para minúsculas consultas curinga. O `QueryParser` construtor leva o campo padrão e analisador. A instância `parse ()` método é passaram a expressão seja analisada. Consulte a seção 3.5.6 para um exemplo.

### Manipulação de expressões de consulta básica com QueryParser

QueryParser traduz expressões de consulta em uma das Lucene é built-in consulta tipos. Falaremos sobre cada tipo de consulta na secção 3.4; por enquanto, tomar a maior imagem fornecida pela tabela 3.2, que mostra alguns exemplos de expressões e sua tradução.

Tabela 3.2 Exemplos de expressões que QueryParser manipula

Expressão de consulta	Documentos partidas que ...
Java	Contém o termo Java no campo padrão
java junit java ou junit	Contém o termo Java ou junit, ou ambos, no padrão FieldA
+ Java + junit E java junit	Contém tanto Java e junit no campo padrão
Título: ant	Contém o termo formiga no título campo
título: extrema -Sujeito: esportes título: extrema E não sujeito: esportes	Ter extremo no título campo e não tem esportes no assunto campo
(Ágil ou extremo) E metodologia	Conter metodologia e também deve conter ágil e / ou extremas, todos no campo padrão
title: "junit em ação"	Contém a frase exata "junit em ação" no título campo
title: "ação junit" ~ 5	Contém os termos junit e ação dentro de cinco posições um do outro
java *	Conter termos que começam com java, como JavaSpaces, JavaServer, e java.net
java ~	Conter termos que estão perto para a palavra java, tal como lava
LastModified: [1/1/04 TO 12/31/04]	Ter LastModified valores de campo entre as datas 01 de janeiro, 2004 e 31 de dezembro de 2004

<sup>um</sup> O operador por defeito é OR. Pode ser definido como E (ver secção 3.5.2).

Com este quadro amplo de capacidades de busca do Lucene, você está pronto para mergulhar detalhes. Vamos revisitar QueryParser na seção 3.5, depois de cobrir o mais fundo-peças tradicionais.

## Usando 3,2 IndexSearcher

---

Vamos dar uma olhada no Lucene `IndexSearcher` classe. Como o resto do Lucene API principal, é simples de usar. Buscas são feitas usando uma instância de `Índice Searcher`. Normalmente, você vai usar uma das seguintes abordagens para a construção de um `IndexSearcher`:

- Por Diretório
- Por um caminho do sistema de arquivos

Recomendamos o uso do `Diretório` -construtor é melhor separar pesquisa ing de onde reside o índice, permitindo que seu código em busca de ser agnóstico se o índice está sendo pesquisado no sistema de arquivos, na RAM, ou em outro lugar. Nosso caso de teste base, `LiaTestCase`, Fornece `diretório`, Um `Diretório` implementação. Sua implementação real é um `FSDirectory` carregado de um sistema de arquivos índice. Nossa `setUp ()` método abre um índice usando o método estático `FSDirectory.getDirectory ()` método, com o caminho índice definido a partir de uma propriedade do sistema JVM:

```
public abstract class LiaTestCase estende TestCase {  
    private String indexDir = System.getProperty ("index.dir");  
    diretório protegido Directory;  
  
    setUp protected void () throws Exception {  
        = diretório FSDirectory.getDirectory (indexDir, false);  
    }  
  
    / / ...  
}
```

O último argumento para `FSDirectory.getDirectory ()` é falso, Indicando que queremos para abrir um índice existente, não construir um novo. Um `IndexSearcher` é criado usando um `Diretório` exemplo, da seguinte forma:

```
IndexSearcher searcher = new IndexSearcher (diretório);
```

Depois de construir uma `IndexSearcher`, Chamamos um dos seus pesquisas métodos para performar uma pesquisa. Os três assinaturas de busca principais método disponível para um `Índice Searcher` exemplo, são mostrados na tabela 3.3. Este capítulo trata apenas de pesquisa (Query) método, e que pode ser o único que você precisa para preocupação your-auto com. Os outros pesquisas assinaturas de métodos, incluindo as variantes de classificação, são cobertos no capítulo 5.

Tabela 3.3 Primário IndexSearcher pesquisa métodos

IndexSearcher.search assinatura do método	Quando usar
Hits de busca (query)	Buscas simples não necessitando de filtragem.
Hits de busca (Query, Filtro filtro)	Pesquisas restrito a um subconjunto de documentos disponíveis, com base em critérios de filtro.
pesquisa void (Query, Resultados HitCollector)	Usado somente quando todos documentos encontrados a partir de uma pesquisa serão necessários. Geralmente apenas os documentos top poucos a partir de uma pesquisa são necessários, de modo que se este método poderia ser um assassino de desempenho.

Um IndexSearcher exemplo busca somente o índice tal como existia no momento da IndexSearcher foi instanciado. Se a indexação está ocorrendo simultaneamente com pesquisa, novos documentos indexados não será visível para pesquisas. A fim de ver os novos documentos, você deve instanciar um novo IndexSearcher.

### 3.2.1 Trabalhando com Hits

Agora que nós chamamos pesquisa (Query), Temos um Hits objeto à nossa disposição. O resultados da pesquisa são acessados através Hits. Normalmente, você usar um dos pesquisa métodos que retorna um Hits objeto, como mostrado na tabela 3.3. O Hits objeto fornece acesso eficiente aos resultados de pesquisa. Resultados são ordenados por relevância em outras palavras, pela maneira como cada documento corresponde à consulta (triagem resultados em outras maneiras é discutido na seção 5.1).

Existem apenas quatro métodos em um Hits exemplo, eles são listados na tabela 3.4. O método Hits.length () retorna o número de correspondentes documentos. Um jogo-documento ing é um com uma pontuação maior que zero, conforme definido pela pontuação formula tratados na secção 3.3. Os hits, por padrão, são em ordem decrescente de pontuação.

Tabela 3.4 Hits métodos para a eficiente acesso aos resultados da busca

Hits método	Valor de retorno
length ()	Número de documentos na Hits coleção
doc (n)	Documento instância do nth maior pontuação documento
id (n)	ID do documento nth maior pontuação documento
pontuação (n)	Pontuação normalizada (com base na pontuação do documento de nível superior) do nth top-documento de pontuação, a garantia de ser maior que 0 e menor ou igual a 1

O `Hits` objeto caches um número limitado de documentos e mantém uma mais-recentemente utilizado lista. Os primeiros 100 documentos são automaticamente recuperados e cache inicialmente. O `Hits` coleção presta-se a ambientes onde os usuários são apresentados apenas com os documentos top poucos e normalmente não precisam mais do que aqueles, porque só os hits best-de pontuação são os documentos desejados.

Os métodos `doc (n)`, `id (n)` E `pontuação (n)` exigir documentos para ser carregado do índice quando eles já não estiverem em cache. Isso nos leva a recomendar apenas chamar esses métodos para documentos que você realmente precisa para mostrar ou acesso; adiar a chamá-los até que seja necessário.

### 3.2.2 Ao folhear Hits

Apresentação de resultados de busca para usuários finais na maioria das vezes envolve apresentando apenas os primeiros 20 ou assim os documentos mais relevantes. Folhear `Hits` é uma necessidade comum.

Há ~~um grande aborrecimento de implementação:~~ instâncias disponíveis enquanto o usuário está navegando os resultados da pesquisa.

- Requery cada vez que o usuário navega para uma nova página.

Acontece que requerying é na maioria das vezes a melhor solução. Requerying elimina a necessidade de armazenar por usuário estado. Em uma aplicação web, permanecendo apátrida (sem Sessão HTTP) é muitas vezes desejável. Requerying à primeira vista parece um desperdício, mas Incrível velocidade Lucene é mais do que compensa.

A fim de repetir a consulta, a pesquisa original é reexecutado e os resultados são dis-jogado início na página desejada. Como a consulta original é mantido depende em sua arquitetura de aplicação. Em uma aplicação web onde o usuário digita em um expressão que é analisado com `QueryParser`, A expressão original poderia ser fez parte da hyperlinks para navegar as páginas e reparsed para cada pedido, ou a expressão poderia ser mantido em um campo oculto HTML ou como um cookie.

Não prematuramente otimizar suas implementações de paginação com cache ou persistência. Primeiro implementar seu recurso de paginação com uma repetição da consulta simples; é provável que você encontrará este suficiente para suas necessidades.

### 3.2.3 índices de leitura na memória

Utilização `RAMDirectory` é adequado para situações que exigem apenas índices transitória, mas a maioria das aplicações é necessário persistir seus índices. Eles acabarão por necessidade de uso `FSDirectory`, Como mostramos nos dois últimos capítulos.

No entanto, em alguns cenários, os índices são usados de uma forma só de leitura. Supõe, por exemplo, que você tem um computador cuja principal de memória excede o tamanho de um índice do Lucene armazenados no sistema de arquivos. Embora seja bom para sempre pesquisar o índice armazenado no diretório de índice, você poderia fazer melhor uso de seu recursos de hardware de carregar o índice do disco mais lento para o mais rápido RAM e depois que o índice de procura na memória. Em tais casos, `RAMDirectory`'S construtor pode ser usado para ler um arquivo de sistema baseado em índice na memória, permitindo que o aplicativo que acessa-lo para beneficiar da velocidade superior da RAM:

```
RAMDirectory ramDir = new RAMDirectory (dir);
```

`RAMDirectory` tem vários construtores sobrecarregados, permitindo uma `java.io.File`, Um caminho Corda, Ou outro Diretório para carregar na memória RAM. Usando um `IndexSearcher` com um `RAMDirectory` é simples e não é diferente do que usar um `FSDirectory`.

### 3,3 de pontuação Lucene Compreensão

---

Escolhemos para discutir este tema complexo no início deste capítulo assim você terá uma gen-sentido gerais dos vários fatores que entram em marcar Lucene como você continuar a ler. Sem mais delongas, conheça Lucene fórmula de pontuação de similaridade, mostrado na figura 3.1. A pontuação é calculada para cada documento ( $d$ ) Correspondente a um específico.

**NOTA** Se essa equação ou o pensamento de cálculos matemáticos assusta você, você pode seguramente pular esta seção. Pontuação Lucene é top-notch como é, e uma compreensão detalhada do que faz funcionar não é necessário tirar partido das capacidades do Lucene.

Esta pontuação é a escore bruto. Pontuações retornou de `Hits` não são necessariamente as matérias-primas pontuação, no entanto. Se a pontuação máxima pontuação documento maior que 1.0, todos os escores são

normalizado a partir dessa pontuação, de modo que todos os escores de `Hits` são garantidos para ser

1.0 ou menos. Tabela 3.5 descreve cada um dos fatores na fórmula de pontuação.

$$\sum_{t \in q} \frac{tf(t \text{ in } d) \cdot df(t) \cdot boost(t) \cdot fieldNorm(t \text{ in } d)}{\text{lengthNorm}(t \text{ in } d)}$$

Figura 3.1 Lucene usa essa fórmula para determinar uma pontuação documento com base em uma consulta.

Tabela 3.5 Fatores na fórmula de pontuação

Fator	Descrição
<code>tf (t em d)</code>	Fator de freqüência prazo para o termo (t) no documento (d).
<code>idf (t)</code>	Freqüência documento inversa do termo.
<code>aumento (t.field em d)</code>	Aumentar a campo, como definido durante a indexação.
<code>lengthNorm (t.field em d)</code>	Valor de normalização de um campo, dado o número de termos dentro da de campo. Este valor é calculado durante a indexação e armazenadas no índice.
<code>coord (q, d)</code>	Fator de coordenação, com base no número de termos de consulta a documento contém.
<code>queryNorm (q)</code>	Valor de normalização para uma consulta, dada a soma dos pesos ao quadrado de cada um dos termos da consulta.

Fatores de impulso são construídos na equação para que você afetará uma consulta ou campo de influ-

referência na pontuação. Aumenta a entrar em campo explicitamente na equação como a `aumento (t.field em d)` fator, definido em tempo de indexação. O valor padrão do campo aumenta, logicamente, é de 1,0.

Durante a indexação, um `Documento` pode ser atribuído um impulso, também. A `Documento` aumentar factor

define implicitamente o impulso campo inicial de todos os campos para o valor especificado. Campo-es-

aumenta específicos são multiplicados pelo valor inicial, que dá o valor final do campo impulsor fator. É possível adicionar o campo de mesmo nome a um `Documento` múltiplo vezes, e em tais situações, o impulso de campo é calculado como todos os impulsora especificado

para esse campo e documentar multiplicados juntos. Seção 2.3 discute-índice de tempo aumentar em mais detalhes.

Além dos fatores explícitos nesta equação, outros fatores podem ser com-discutíveis em uma base per-consulta, como parte do `queryNorm` fator. Consultas se pode ter um impacto sobre a classificação do documento. Impulsionar um `Pergunta` exemplo, é sensível apenas

em uma consulta cláusula de múltipla, se apenas um único termo é usado para pesquisa, impulsorando-o

impulsionaria todos os documentos encontrados igualmente. Em um múltiplo-cláusula de consulta boolean,

alguns documentos podem corresponder a uma cláusula, mas não outra, permitindo que o fator de impulso

a discriminar entre as consultas. As consultas também padrão para um fator de 1,0 impulso.

A maioria destes fatores fórmula de pontuação são controlados através de uma imple-

ção da `Similaridade classe`. `DefaultSimilarity` é a implementação usada

salvo indicação em contrário. Mais cálculos são realizados sob as cobertas

de `DefaultSimilarity`, Por exemplo, o fator de freqüência do termo é a raiz quadrada de

a freqüência real. Porque este é um "em ação" livro, é para além da do livro

espaço para aprofundar o funcionamento interno destes cálculos. Na prática, é

extremamente raro precisar de uma mudança nesses fatores. Caso você precise alterar esses fatores, consulte a [Similaridade](#) Javadoc s, e estar preparado com um sólido sub-pé desses fatores e os efeitos que as alterações terão.

É importante notar que a mudança no índice de tempo aumenta ou o [Similaridade](#) métodos utilizados durante a indexação exigir que o índice ser reconstruído para todos os fatores para estar em sincronia.

### 3.3.1 Lucene, você tem um monte de 'splainin' para fazer!

Ufa! A fórmula de pontuação parece assustador, e é. Estamos falando de fatores que o documento rank maior do que um outro com base em uma consulta, para que em e por si só merece a sofisticação acontecendo. Se você quiser ver como todos estes fatores jogar fora, Lucene fornece um recurso chamado `Explicação.IndexSearcher` tem um explicar método, que requer um `Pergunta` e identificação de um documento e retorna um `Explicação` objeto.

O `Explicação` objeto internamente contém todos os detalhes que o fator em o cálculo de pontuação. Cada detalhe pode ser acessado individualmente, se você gosta, mas geralmente, despejar a explicação em sua totalidade é desejado. O `.ToString()` dumps um método de representação de texto bem formatado do `Explicação`. Nós escreveu um programa simples de despejo `Explicação`s, mostrado aqui:

```
public class {Explicador
    public static void main (String [] args) throws Exception {
        if (args.length! = 2) {
            System.err.println ("Uso: Explicador <query> dir> <index");
            System.exit (1);
        }

        String indexDir = args [0];
        QueryExpression String = args [1];

        Diretório FSDirectory =
            FSDirectory.getDirectory (indexDir, false);

        Query = QueryParser.parse queryExpression (,
            "Conteúdo", novo SimpleAnalyzer ());

        System.out.println ("Query:" queryExpression +);

        IndexSearcher searcher = new IndexSearcher (diretório);
        Hits hits = searcher.search (query);

        for (int i = 0; i <hits.length (); i + +) {
```

```
Explicação explicação =  
    searcher.explain (consulta, hits.id (i));  
  
System.out.println ("-----");  
Documento doc = hits.doc (i);  
System.out.println (doc.get ("title"));  
System.out.println (explanation.toString());  
  
}  
}  
}
```

Usando a consulta `junit` contra o nosso índice de amostra produziu o seguinte resultado; notar que o título mais relevante marcou o melhor:

```
Consulta: junit
-----
JUnit em Ação
= 0.65311843 fieldWeight (conteúdo: junit no 2), produto de:
 1.4142135 = tf(termFreq(conteúdo: junit) = 2)"Junit"
                     aparece duas vezes no1.8472979 = idf(docFreq = 2)
                                                 conteúdo0,25 = fieldNorm
(campo = conteúdo, doc = 2)

-----
Java Development com Ant
= 0.46182448 fieldWeight (conteúdo: junit em 1), produto de:
 1.0 = tf(termFreq(conteúdo: junit) = 1)"Junit"
                     aparece uma vez1.8472979 = idf(docFreq = 2)
                                                 em conteúdos0,25 =
fieldNorm (campo = conteúdo, doc = 1)
```

- b JUnit em Ação tem o prazo junit duas vezes em sua conteúdo de campo. O conteúdo campo em índice é uma agregação dos título e assunto campos para permitir que um único campo para pesquisa.
  - c Java Development com Ant tem o prazo junit apenas uma vez em sua conteúdo de campo. Há também uma . ToHtml () método que gera a mesma estrutura hierárquica, exceto como HTML aninhadas <ul> elementos adequados para a saída em um navegador web. De fato, o Explicação recurso é uma parte essencial do projeto Nutch (ver o caso estudo na seção 10.1), permitindo ranking transparente.  
Explicações são úteis para ver o funcionamento interno do cálculo de pontuação, mas que gastam a mesma quantidade de esforço como uma consulta. Assim, não se esqueça de usar extrane-ous Explicacão geracão.

### 3.4 consultas Criar programaticamente

Como você viu na seção 3.2, a consulta Lucene, em última análise requer uma chamada para índice Searcher's pesquisa usando uma instância de Pergunta. Pergunta subclasses pode ser instanciado diretamente, ou, como discutimos na seção 3.1.2, uma Pergunta podem ser construídos através de

o uso de um parser, como `QueryParser`. Se sua aplicação vai depender exclusivamente dos `QueryParser` para a construção de `Pergunta` objetos, a compreensão direta API do Lucene capa-dades ainda é importante, porque `QueryParser` utiliza-los.

Mesmo se você estiver usando `QueryParser`, Combinando uma expressão de consulta analisado com uma API-criado `Pergunta` é uma técnica comum para aumentar, aperfeiçoar ou restringir um humano entrou consulta. Por exemplo, você pode querer restringir de forma livre analisado expressões a um subconjunto do índice, como documentos só dentro de uma categoria. Dependendo da interface do usuário a sua pesquisa, você pode ter selecionadores de data para selecionar uma intervalo de data, drop-downs para selecionar uma categoria, e uma forma livre caixa de pesquisa.

Cada uma dessas cláusulas podem ser agrupados usando uma combinação de `Consulta-Analisador`, `BooleanQuery`, `RangeQuery`, E um `TermQuery`. Nós demonstramos a construção de uma consulta agregada similar na seção 5.5.4.

Esta seção aborda cada um dos Lucene é built-in `Pergunta` tipos. O `QueryParser` sintaxe da expressão que mapeia para cada `Pergunta` tipo é fornecido.

### 3.4.1 Pesquisa por termo: `TermQuery`

A forma mais elementar para procurar um índice é de um termo específico. Um termo é a menor parte indexada, consistindo de um nome de campo de texto e um par de valor. Listagem 3.1

forneceu um exemplo de busca de um termo específico. Este código constrói um `Prazo` instância do objeto:

```
Termo t = novo Termo ("conteúdo", "java");
```

ATermQuery aceita uma única Prazo:

```
Query = new TermQuery (t);
```

Todos os documentos que têm a palavra Java em um conteúdo campo são retornados de pesquisas usando este `TermQuery`. Note que o valor é case-sensitive, por isso certifique-se de coincidir com o caso de termos indexados, o que não pode ser o caso exato no original doc-texto que este documento, porque um analisador (ver capítulo 5) pode ter indexado as coisas de forma diferente.

`TermQuery`s são especialmente úteis para a recuperação de documentos por uma chave. Se documentos foram indexados usando `Field.Keyword ()`, O mesmo valor pode ser usado para recuperar esses documentos. Por exemplo, dado o nosso livro de dados de teste, o seguinte código recupera o único documento correspondente ao ISBN, desde que:

```
testKeyword public void () throws Exception {
    IndexSearcher searcher = new IndexSearcher (diretório);
    Termo t = novo Termo ("isbn", "1930110995");
    Query = new TermQuery (t);
    Hits hits = searcher.search (query);
    assertEquals ("JUnit em Ação", 1, hits.length ());
}
```

A `Field.Keyword` campo não implica que ele é único, porém. Cabe a você para garantir a exclusividade durante a indexação. Em nossos dados, `isbn` é único entre os todos os documentos.

### TermQuery e QueryParser

Uma única palavra em uma expressão de consulta corresponde a um termo. A `TermQuery` é retornou de `QueryParser` se a expressão consiste em uma única palavra. O expressão Java cria uma `TermQuery`. Assim como fizemos com a API em `testKeyword`.

#### 3.4.2 Buscando dentro de uma faixa: RangeQuery

Termos são ordenados lexicograficamente dentro do índice, permitindo uma eficiente busca de termos dentro de uma faixa. Lucene `RangeQuery` facilita as buscas a partir de um termo de partida através de um termo final. Os termos de início e fim podem ser incluídas ou excluídas. O código a seguir ilustra consultas de intervalo inclusive do começar e terminar termos:

```
public class RangeQueryTest estende LiaTestCase {
    Prazo privada com início e final;

    setUp protected void () throws Exception {
        begin = novo Termo ("pubmonth", "198805");

        Data / pub / do TTC foi out 1988
        Prazo final = new ("pubmonth", "198810");

        super.setUp ();
    }

    public void testInclusive () throws Exception {
        Consulta RangeQuery RangeQuery = new (início, fim, true);
        IndexSearcher searcher = new IndexSearcher (diretório);

        Hits hits = searcher.search (query);
        assertEquals ("tao", 1, hits.length ());
    }
}
```

Nosso teste conjunto de dados tem apenas um livro, Tao Te Ching por Stephen Mitchell, publicado entre maio de 1988 e outubro de 1988, que foi publicado em Outubro de 1988. O terceiro argumento para construir um `RangeQuery` é um sinalizador booleano que indica se o intervalo é inclusiva. Usando os mesmos dados e alcance, embora não exclusivamente, um a menos livro é encontrado:

```
public void testExclusive () throws Exception {
    Consulta RangeQuery RangeQuery = new (início, fim, false);
```

```

IndexSearcher searcher = new IndexSearcher (diretório);

Hits hits = searcher.search (query);
assertEquals ("não há tao", 0, hits.length ());
}

```

### RangeQuery e QueryParser

QueryParser construções RangeQuery s a partir da expressão [Begin TO end] ou {Início a fim}. Colchetes denotam um intervalo inclusivo e chaves denotam uma exclusiva gama. Se o início eo final termos representam datas (e analisar sucessivamente como tal), então varia sobre os campos criados como datas usando DateField ou Palavra-chave (String, Data) pode ser construído. Consulte a seção 3.5.5 para mais informações sobre Faixa-Pergunta e QueryParser.

### 3.4.3 Pesquisando em uma string: PrefixQuery

Pesquisando com uma PrefixQuery documentos contendo termos partidas início com uma seqüência de caracteres especificada. É aparentemente acessível. O código a seguir demonstra como você pode consultar uma estrutura hierárquica recursivamente com um simples PrefixQuery. Os documentos contêm uma categoria campo palavra-chave que representa uma hierarquia:

```

PrefixQueryTest estende LiaTestCase {
    testPrefix public void () throws Exception {
        IndexSearcher searcher = new IndexSearcher (diretório);

        Termo termo Term = new ("categoria",
            "/ Tecnologia / computadores / programação");
        PrefixQuery query = new PrefixQuery (prazo);

        Hits hits = searcher.search (query);
        int programmingAndBelow hits.length = ();

        hits = searcher.search (novo TermQuery (termo));
        int justProgramming hits.length = ();

        assertTrue (programmingAndBelow > justProgramming);
    }
}

```

Procurar programação livros, incluindo subcategorias

Pesquisar apenas para livros de programação, não subcategorias

Nosso PrefixQueryTest demonstra a diferença entre um PrefixQuery e um TermQuery. A metodologia categoria existe abaixo da / Tecnologia / computadores / programação categoria. Livros desta subcategoria são encontrados com uma PrefixQuery mas não com o TermQuery na categoria pai.

### PrefixQuery e QueryParser

`QueryParser` cria uma `PrefixQuery` para um mandato de quando ele termina com um asterisco (\*) em expressões de consulta. Por exemplo, `luc *` é convertido em um `PrefixQuery` utilizando `luc` como o termo. Por padrão, o texto do prefixo é em minúscula por `QueryParser`. Ver seção 3.5.7 para obter detalhes sobre como controlar essa configuração.

#### 3.4.4 Combinando consultas: BooleanQuery

Os tipos de consulta vários discutidas aqui podem ser combinadas de formas complexas usando `BooleanQuery`. `BooleanQuery` se é um recipiente de Boolean cláusulas. Uma cláusula é uma subconsulta que pode ser opcional, exigido, ou proibido. Estes atributos permitem combinações lógicas AND, OR e NOT. Você pode adicionar uma cláusula a uma `BooleanQuery` usando este método API:

```
public void add (Query, boolean exigido, boolean proibida)
```

A `BooleanQuery` pode ser uma cláusula dentro de outro `BooleanQuery`, Permitindo a sofisticados agrupamentos sofisticadas. Vejamos alguns exemplos. Primeiro, aqui está uma consulta E para encontrar os livros mais recentes em um dos nossos temas favoritos, busca:

```
testAnd public void () throws Exception {
    TermQuery searchingBooks =
        nova TermQuery (novo Termo ("subject", "busca"));
```

Todos os livros com  
"search" sujeito

b

```
CurrentBooks RangeQuery =
    RangeQuery novo (novo Termo ("pubmonth", "200401")
                    novo prazo ("pubmonth", "200412"),
                    true);
```

Todos os livros  
em 2004

c

```
BooleanQuery currentSearchingBooks BooleanQuery = new ();
currentSearchingBooks.add (searchingBook s, true, false);
currentSearchingBooks.add (currentBooks, true, false);
```

Combina  
duas consultas

d

```
IndexSearcher searcher = new IndexSearcher (diretório);
Hits hits = searcher.search (currentSearchingBooks);
```

```
assertHitsIncludeTitle (hits, "Lucene in Action");
}
```

```
/ / Método a seguir a partir da base LiaTestCase classe
assertHitsIncludeTitle protected void final (
    Hits hits, titulo String)
throws IOException {
for (int i = 0; i < hits.length (); i + +) {
    Documento doc = hits.doc (i);
    if (title.equals (doc.get ("title"))) {
        assertTrue (true);
```

Personalizado  
conveniência  
método assert

e

```

        retorno;
    }
}

e Personalizado
conveniência
método assert

falha ("title" + título + "não encontrado");
}

```

- b Esta consulta encontra todos os livros que contenham o assunto "Search".
- c Esta consulta encontra todos os livros publicados em 2004. (Note que isso também poderia ser feito com um "2004" PrefixQuery.)
- d Aqui nós combinamos as duas consultas em uma única consulta booleana com ambas as cláusulas necessário (o segundo argumento é verdadeiro).
- e Esta conveniência personalizado método permite afirmar casos de teste mais legível. BooleanQuery.add tem duas assinaturas de método sobrecarregado. Se aceita uma BooleanCláusula, E o outro aceita um Pergunta e duas bandeiras boolean. A BooleanClause é um recipiente de uma consulta e as duas bandeiras boolean, então nós omitir a cobertura do mesmo. O bandeiras são boolean exigido e proibido. Respectivamente. Há quatro lógicas combinações desses sinalizadores, mas o caso em que ambos são verdadeiro é um ilógico e inválido combinação. Ima cláusula exigida significa exatamente isso: Apenas os documentos correspondência que a cláusula são considerados. Tabela 3.6 mostra as várias combinações e efeito dos sinalizadores necessários e proibido.

Tabela 3.6 BooleanQuery atributos cláusula

		exigido	
		falso	verdadeiro
proibido	falso	Cláusula é opcional	Cláusula deve coincidir com
	verdadeiro	Cláusula deve não combinar	Inválido

Executar uma consulta ou só é preciso definir o exigido e proibido bandeiras tanto para falso, Como neste exemplo:

```

Testor public void () throws Exception {
    TermQuery methodologyBooks = new TermQuery (
        Termo de novo ("categoria",
            "/ Tecnologia / computadores / programação / metodologia"));

    TermQuery easternPhilosophyBooks = new TermQuery (
        Termo de novo ("categoria",
            "/ Filosofia / leste"));

    BooleanQuery enlightenmentBooks = new BooleanQuery ();

```

```
enlightenmentBooks.add (methodologyBooks, false, false);
enlightenmentBooks.add (easternPhilosophyBooks, false, false);

IndexSearcher searcher = new IndexSearcher (diretório);
Hits hits = searcher.search (enlightenmentBooks);

assertHitsIncludeTitle (hits, "Extreme Programming Explained");
assertHitsIncludeTitle (hits,
    "Tao Te Ching \ u9053 \ u5FB7 \ u7D93" 4);
}
```

BooleanQuery's são restritos a um número máximo de cláusulas; 1024 é o padrão. Esta limitação está no local para evitar que consultas prejudiquem desempenho. A `TooManyClauses` exceção é lançada se o máximo for excedido. Pode parecer que este é um número extrema e que a construção deste número de cláusulas é improvável, mas sob o Lucene não cobre algumas de suas própria consulta rewriting para consultas como `RangeQuery` e os transforma em um `BooleanQuery` com aninhadas opcional (não obrigatório, não é proibido) `TermQuery`s. Você já deve ter a necessidade incomum de aumentar o número de cláusulas permitido, há uma `setMaxClauseCount (int)` método em `BooleanQuery`.

### BooleanQuery e QueryParser

`QueryParser` handily construições `BooleanQuery`s, quando vários termos são especificados. O agrupamento é feito com parênteses, e os proibido e exigido bandeiras são definido quando o -, +, AND, OR, e NOT operadores são especificados.

#### 3.4.5 Pesquisa por frase: PhraseQuery

Um índice contém informações posicionais de termos. `PhraseQuery` usa essa informação para localizar documentos onde os termos estão dentro de uma determinada distância de um o outro. Por exemplo, suponha um campo continha a frase "A ligeira raposa marrom saltou sobre o cão preguiçoso ". Sem saber a frase exata, você ainda pode encontrar neste documento, procurando documentos com campos tendo rápido e raposa perto uns aos outros. Claro, uma planície `TermQuery` faria o truque para localizar este documento conhecer qualquer uma daquelas palavras, mas neste caso queremos apenas documentos que tenham frases onde as palavras são ou exatamente lado a lado (rápida fox) ou ter um palavra no meio (rápida [irrelevante] fox).

A distância máxima permitida posicional entre os termos a serem considerados um jogo é chamado slop. Distância é o número de movimentos posicionais de termos a

<sup>4</sup> O \ u notação é uma seqüência de escape Unicode. Neste caso, estes são os caracteres chineses para Tao Te Ching. Usamos isso para a nossa busca de caracteres asiáticos na seção 4.8.3.

reconstruir a frase em ordem. Vamos dar a frase que acabamos de mencionar e ver como o fator de slop joga fora. Primeiro precisamos de uma infra-estrutura de pequeno teste, que inclui um `setUp ()` método para indexar um documento único e um costume `combinados (String [], int)` método para construir, executar e fazer valer uma consulta correspondência de frase

o documento de teste:

```
public class PhraseQueryTest estende TestCase {
    privada IndexSearcher pesquisador;

    setUp protected void () throws IOException {
        / / Set up documento de exemplo
        Diretório RAMDirectory = new RAMDirectory ();
        Escritor IndexWriter IndexWriter = new (diretório,
            nova WhitespaceAnalyzer (), true);
        Documento doc = new Document ();
        doc.add (Field.Text ("campo",
            "A ligeira raposa marrom saltou sobre o cão preguiçoso"));
        writer.addDocument (doc);
        writer.Close ();

        searcher = new IndexSearcher (diretório);
    }

    boolean privada correspondente (String [] frase, slop int)
        throws IOException {
        Consulta PhraseQuery PhraseQuery = new ();
        query.setSlop (slop);

        for (int i = 0; i frase.length <; i + +) {
            query.add (Termo de novo ("campo", frase [i]));
        }

        Hits hits = searcher.search (query);
        hits.length return ()> 0;
    }
}
```

Porque queremos demonstrar exemplos de consulta vários frase, nós escrevemos o combinados método para simplificar o código. Consultas frase são criadas adicionando termos na ordem desejada. Por padrão, um `PhraseQuery` tem o seu fator de slop definido para zero, especificando uma correspondência frase exata. Com o nosso `setUp ()` e ajudante combinados método, nosso caso de teste apresenta sucintamente como `PhraseQuery` se comporta. Falhando e passagem fatores slop mostram os limites:

```
testSlopComparison public void () throws Exception {
    String [] frase = new String [] {"quick", "fox"};

    assertFalse ("frase exata não encontrada", combinado (frase, 0));
    assertTrue ("perto o suficiente", combinado (frase, 1));
}
```

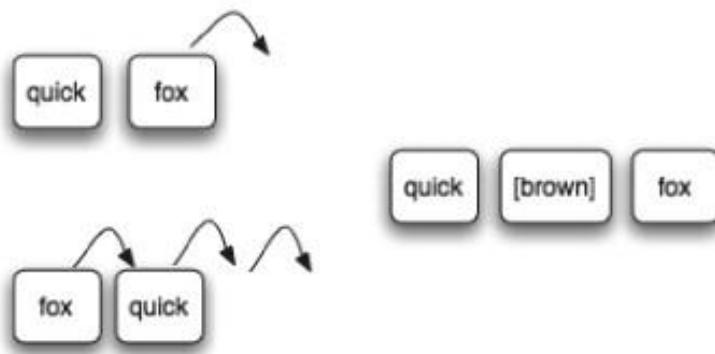


Figura 3.2 Ilustrando PhraseQuery fator de slop: "fox rápido" requer um slop de 1 a partida, enquanto a "raposa rápido" exige uma poça de 3 a partida.

Termos adicionado a uma consulta frase não tem que ser na mesma ordem encontrada no campo, embora a ordem tem impacto slop fator de considerações. Por exemplo, tinha os termos sido invertida na consulta (raposa e, em seguida, rápida), o número de movimentos necessários para combinar com o documento seria três, não um. Para visualizar isso, consider quantos movimentos que seria necessário para mover fisicamente a palavra raposa dois slots

passado rápida; você verá que é preciso um movimento para mover raposa na mesma posição rápida e depois mais dois para mover raposa além rápido suficiente para corresponder "quick raposa marrom".

Figura 3.2 mostra como as posições slop trabalho em ambos os consulta frase cenários, e este caso de teste mostra o jogo em ação:

```
testReverse public void () throws Exception {
    String [] frase = new String [] {"raposa", "rápido"};

    assertFalse ("flop hop", combinado (frase, 2));
    assertTrue ("slop hop hop", combinado (frase, 3));
}
```

Vamos agora examinar como vários prazo frase trabalho consultas.

### Múltiplas prazo frases

PhraseQuery suporta múltiplos prazo frases. Independentemente de quantos termos são utilizado por uma frase, o fator de slop é o máximo total número de movimentos permitidos para colocar os termos em ordem. Vejamos um exemplo de uma consulta frase múltiplas prazo:

```
testMultiple public void () throws Exception {
    assertFalse ("não perto o suficiente",
        combinados (new String [] {"quick", "saltou", "preguiçosos"}, 3));
```

```

        assertTrue ("apenas o suficiente",
            combinados (new String [] {"quick", "saltou", "preguiçosos"}, 4));

        assertFalse ("quase mas não completamente",
            combinados (new String [] {"preguiçoso", "saltou", "rápido"}, 7));

        assertTrue ("bingo",
            combinados (new String [] {"preguiçoso", "saltou", "rápido"}, 8));
    }
}

```

Agora que você já viu como consultas frase partida, voltamos nossa atenção para como consultas frase afetar o placar.

#### Marcar consulta frase

Consultas frase são pontuadas com base na distância necessária para editar

1

coincidir com a frase. Partidas mais exata contam para mais peso

$\frac{1}{distance + 1}$

do que os mais negligentes. O fator de consulta frase é mostrado na figura 3.3.

A relação inversa com a distância garante que a maior distâncias têm notas mais baixas.

Figura 3.3  
Frase Sloppy  
pontuação

**NOTA** Termos entre aspas duplas em `QueryParser` expres-analisado  
sões são traduzidos em um `PhraseQuery`. O slop padrões fator a ze-  
ro, mas você pode ajustar o fator de slop, adicionando um til (~) Seguido por um  
inteiro. Por exemplo, a expressão "Fox rápido" ~ 3 é uma `PhraseQuery`  
com os termos rápido e raposa e um fator de slop de 3. Há adicionais  
detalhes sobre `PhraseQuery` eo fator de slop na seção 3.5.6. Frases  
são analisados pelo analisador passado para o `QueryParser`, Acresentando um outro  
eixo  
er camada de complexidade, como discutido na seção 4.1.2.

#### 3.4.6 Pesquisando por curinga: WildcardQuery

Consultas curinga permitem que você consulta para termos com peças em falta, mas ainda encontrar

jogos. Dois caracteres curinga padrão são usados: \*para zero ou mais caracteres,  
e ?para zero ou um personagem. Listagem 3.2 demonstra `WildcardQuery` em ação.

Pesquisando listagem 3.2 no lado selvagem (cartão)

```

indexSingleFieldDocs private void (Field [] fields) throws Exception {
    Escritor IndexWriter writer = new (diretório,
        nova WhitespaceAnalyzer (), true);
    for (int i = 0; i < fields.length; i++) {
        Documento doc = new Document ();
        doc.add (campos [i]);
        writer.addDocument (doc);
    }
}

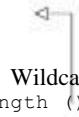
```

```
writer.optimize ();
writer.Close ();
}

testWildcard public void () throws Exception {
    indexSingleFieldDocs (Field new []
        {Field.Text ("conteúdo", "selvagem"),
         Field.Text ("conteúdo", "criança"),
         Field.Text ("conteúdo", "mild"),
         Field.Text ("Conteúdo", "mofo")});

IndexSearcher searcher = new IndexSearcher (diretório);
Query = nova WildcardQuery (
    Termo de novo ("conteúdo", "? ILD *"));
    Construir
Hits hits = searcher.search (query);
assertEquals ("criança não corresponder", 3, hits.length ());
assertEquals ("a mesma pontuação", hits.score (0),
                hits.score (1), 0,0);
assertEquals ("a mesma pontuação", hits.score (1),
                hits.score (2), 0,0);

}
```



WildcardQuery

usando o termo

Note como o padrão curinga é criado como um `Prazo` (O padrão a ser correspondido), mesmo embora não seja explicitamente usado como um termo exato debaixo das cobertas. Internamente, é usado como um padrão para coincidir com termos no índice. A `Prazo` instância é um conveniente espaço reservado para representar um nome de campo e uma string.

#### AVISO

Degradações de desempenho pode ocorrer quando você usa `WildcardQuery`. A maior prefixo (caracteres antes do primeiro caractere curinga) diminui os termos enumerados para encontrar correspondências. Início um padrão com uma selvagem consulta cartão de forças a enumeração termo para pesquisa todos termos do índice para os jogos.

Estranhamente, a proximidade de um jogo de curinga não tem efeito sobre a pontuação. Os dois últimos afirmações na listagem 3.2, quando selvagem e suave são jogos mais próximo do padrão que mofo, demonstrar isso.

#### WildcardQuery e QueryParser

`QueryParser` suporta `WildcardQuery` usando a mesma sintaxe para um mandato como o usado por da API. Existem algumas diferenças importantes, no entanto. Com `QueryParser`, O primeiro caractere curinga de um termo não pode ser um caractere curinga, o que restrição impede que os usuários colocando asterisk-prefixado termos em uma expressão de busca,

inserir em uma operação cara de enumerar todos os termos. Além disso, se a única caractere curinga no termo é um asterisco, a consulta é otimizada para uma `PrefixQuery`. Termos curinga são escritos em minúscula automaticamente por padrão, mas esta pode ser alterado. Consulte a seção 3.5.7 para mais informações sobre consultas e curinga `QueryParser`.

### 3.4.7 Pesquisando por termos semelhantes: `FuzzyQuery`

A consulta built-in final é um dos mais interessantes. Lucene `FuzzyQuery` termos partidas similar para um período especificado. O Distância Levenshtein algoritmo determina como termos similares no índice são para um destino especificado term.<sup>5</sup> Edit distância é um outro termo para a distância Levenshtein, é uma medida de similaridade entre duas cordas, onde a distância é medido como o número de caracteres DELEções, inserções ou substituições necessárias para transformar uma string para o outro string. Por exemplo, a distância entre a edição três e árvore é 1, porque só um exclusão personagem é necessário.

Levenshtein distância não é o mesmo que o cálculo de distância usada em `PhraseQuery` e `PhrasePrefixQuery`. A distância consulta frase é o número de move-se prazo para a partida, enquanto Levenshtein distância é um cálculo intraterm da personagem se move. O `FuzzyQuery` teste demonstra seu uso e comportamento:

```
testFuzzy public void () throws Exception {
    indexSingleFieldDocs (Field new [] {
        Field.Text ("conteúdo", "fuzzy"),
        Field.Text ("Conteúdo", "Wuzzy")
    });

    IndexSearcher searcher = new IndexSearcher (diretório);
    Query = FuzzyQuery novo (novo Termo ("Conteúdo", "wuzza"));
    Hits hits = searcher.search (query);
    assertEquals ("ambos perto o suficiente", 2, hits.length ());

    assertTrue ("Wuzzy mais perto do que fuzzy",
        hits.score (0) > hits.score (1));

    assertEquals ("wuzza ursa",
        "Wuzzy", hits.doc (0).get ("conteúdo"));
}
```

Este teste mostra um par de pontos-chave. Ambos os documentos partida; o termo pesquisado (wuzza) não foi indexado, mas estava perto o suficiente para corresponder. `FuzzyQuery` usa um limiar ao invés de uma distância de edição pura. O limite é um fator de edit distância dividida pelo comprimento da corda.

---

<sup>5</sup> Ver <http://www.merriampark.com/ld.htm> para mais informações sobre a distância Levenshtein.

$$1 - \frac{distance}{\min(textlen, targetlen)}$$

Figura 3.4  
FuzzyQuery fórmula de distância.

Distância editar afeta scoring, de modo que termos com menor distância de edição são pontuadas superior. Distância é calculada utilizando a fórmula apresentada na figura 3.4.

**AVISO** `FuzzyQuery` enumera todos os termos de um índice para encontrar termos dentro do limite permitido. Use esse tipo de consulta com moderação, ou pelo menos com o conhecimento de como funciona eo efeito que pode ter sobre o desempenho.

### FuzzyQuery e QueryParser

`QueryParser` suporta `FuzzyQuery` por sufixação um termo com um til (~). Por exemplo, o `FuzzyQuery` do exemplo anterior seria `wuzza ~` em uma expressão de consulta. Note-se que o til é usado também para especificar consultas frase descuidada, mas o contexto é diferentes. Aspas indicam uma consulta frase e não são utilizados para consultas fuzzy.

## 3.5 Parsing expressões de consulta: QueryParser

Embora API-criado consultas podem ser poderosas, não é razoável que todas as consultas deve ser explicitamente escrito em código Java. Usando uma consulta legível textual representação, Lucene `QueryParser` constrói um dos mencionados anteriormente `Pergunta` subclasses. Esta construída `Pergunta` exemplo, poderia ser uma entidade complexa, consistindo de nested `BooleanQuery`s e uma combinação de quase todos os `Pergunta` tipos mencionadas, mas uma expressão digitada pelo usuário pode ser tão legível como esta:

```
Pubdate +: [20040101 TO 20041231] E Java (Jakarta Apache OR)
```

Esta busca é para todos os livros sobre Java que também incluem Jacarta ou Apache em seu conteúdo e foram publicados em 2004.

**NOTA** Sempre que os caracteres especiais são usados em uma expressão de consulta, é preciso fornecer um mecanismo de fuga para que os caracteres especiais podem ser utilizado de forma normal. `QueryParser` usa uma barra invertida (\) Para escapar caracteres especiais dentro dos prazos. Os caracteres escapable são as seguintes:

```
\ + - ! ( ) : ^ ] { } ~ * ?
```

As seguintes seções detalham a sintaxe da expressão, exemplos de uso `Consulta-Analisador` personalizando `QueryParser` comportamento's. A discussão de `QueryParser` em Nesta seção pressupõe o conhecimento dos tipos de consulta previamente discutido na seção 3.4.

Começamos com uma maneira prática de vislumbrar o que `QueryParser` se a expressões.

### 3.5.1 Query.toString

Coisas aparentemente estranhas podem acontecer com uma expressão de consulta, como é analisado com

`QueryParser`. Como você pode dizer o que realmente aconteceu com a sua expressão? Foi traduzido corretamente para o que você pretendia? Uma maneira de espreitar uma resultante Pergunta exemplo é a utilização do `toString ()` método.

Todos núcleo de concreto `Pergunta` classes que discutimos neste capítulo têm uma especial `toString ()` implementação. Eles saída válida `QueryParser` parsable strings. O padrão `Object.toString ()` método é substituído e os delegados a um `toString (String campo) ()` método, onde `campo` é o nome do campo padrão. Chamando o não-arg `toString ()` método usa um nome de campo padrão vazio, fazendo com que explicitamente a saída para usar a notação de campo seletor para todos os termos. Aqui está um exemplo de usar o `toString ()` método:

```
testToString public void () throws Exception {
    Query = BooleanQuery BooleanQuery new ();
    query.add (
        nova FuzzyQuery (novo prazo ("campo", "kountry6")), true, false);
    query.add (
        nova TermQuery (novo Termo ("title", "western")), false, false);

    assertEquals ("ambos os tipos",
        "Kountry + ~ título: ocidental",
        query.toString ("campo"));
}
```

O `toString ()` métodos (particularmente o `CordaArg-um`) são úteis para visual depuração de consultas API complexa, bem como obter uma alça sobre como Consulta-Analisador interpreta expressões de consulta. Não confie na capacidade de voltar e diante precisa entre uma `Query.toString ()` representação e uma `QueryParser`-analizado expressão, no entanto. Em geral, é preciso, mas um analisador está envolvido e pode confundir as coisas, esta questão é discutida na seção 4.1.2.

### 3.5.2 operadores booleanos

A construção de queries Boolean textualmente via `QueryParser` é feito usando o operadores AND, OR e NOT. Termos listado sem um operador especificado usar um operador implícito, que por padrão é OR. A consulta `abc xyz` será interpretado como quer ABC ou xyz ou abc e xyz, Com base na definição de operador implícito. Para switch de análise para usar E, use uma instância de `QueryParser` em vez da estática analisar método:

---

<sup>6</sup> Incorretas de propósito para ilustrar FuzzyQuery.

```
Parser QueryParser = new ("conteúdo", analisador);
parser.setOperator (QueryParser.DEFAULT_OPERATOR_AND);
```

Colocação de um NÃO na frente de um termo, exclui documentos correspondentes aos seguintes prazo. Negando um termo deve ser combinada com pelo menos um termo para nonnegated documentos de retorno, em outras palavras, não é possível usar uma consulta como NÃO prazo para localizar todos os documentos que não contêm um termo. Cada um a palavra em maiúsculas opera-  
res tem uma sintaxe de atalho; tabela 3.7 ilustra equivalentes sintaxe vários.

Tabela 3.7 Atalhos operador booleano consulta

Sintaxe detalhada	Sintaxe de atalho
E um b	+ A + b
a ou b	um b
A e não b	+ A-b

### 3.5.3 Agrupamento

Lucene BooleanQuery permite construir complexo cláusulas aninhadas, do mesmo modo, Consulta-  
Analizador permite-a com expressões de consulta. Vamos encontrar todos os livros de metodologia  
que são ou sobre metodologias ágeis ou extremas. Usamos parênteses para formar subqueries, permitindo a construção de avançados BooleanQuerys:

```
testGrouping public void () throws Exception {
    Query = QueryParser.parse (
        "(Ágil ou extremo) e metodologia",
        "Sujeito",
        analisador);
    Hits hits = searcher.search (query);

    assertEquals (hits, 2);
    assertEquals (hits[0].getScore(), hits[1].getScore());
    assertEquals (hits[0].getScore(), hits[1].getScore());
}
```

Em seguida, discutiremos como um campo específico pode ser selecionado. Observe que a seleção de campo também pode alavancar parênteses.

### 3.5.4 seleção Field

QueryParser precisa saber o nome do campo para utilização na construção de consultas, mas que geralmente seria hostil a exigir que os usuários para identificar o campo de pesquisa (o usuário final pode não precisar ou quiser saber os nomes dos campos). Como você viu, a inadimplência do nome do campo está prevista para o analisar método. Consultas analisado não se restringem, como-  
sempre, a procurar apenas o campo padrão. Usando a notação de seletor de campo, você pode

especificar os termos em campos não-padrão. Por exemplo, quando os documentos HTML são indexados com o título e as áreas do corpo como campos separados, o campo padrão provavelmente ser `corpo`. Os usuários podem procurar `título` campos usando uma consulta como `título: lucene`. Você pode agrupar seleção de campo sobre vários termos usando campo: `(a, b, c)`.

### 3.5.5 Faixa de buscas

Consultas intervalo de texto ou data de usar a sintaxe de colchetes, com `TO` entre o início prazo e terminando prazo. O tipo de suporte determina se o intervalo é inclusive (entre colchetes) ou exclusivas (entre chaves). Nosso `testRangeQuery ()` método demonstra consultas gama tanto, inclusivo e exclusivo:

```
AM
testRangeQuery public void () throws Exception {
    Query = QueryParser.parse (
        "Pubmonth: [200401 TO 200412]", "sujeito", analisador);
    assertTrue (RangeQuery instanceof consulta);

    Hits hits = searcher.search (query);
    assertHitsIncludeTitle (hits, "Lucene in Action");

    TE
    query = QueryParser.parse (
        "{200201 TO 200208}", "pubmonth", analisador);
    hits = searcher.search (query);
    assertEquals ("JDwA em 200.208", 0, hits.length ());
}
FL
    b Inclusivo alcance
    c Exclusivo alcance
    d Demonstra exclusão de pubmonth 200208
```

- b Esta gama inclusive usa um seletor de campo desde o campo padrão é `assunto`.
- c Esta gama exclusiva usa o campo padrão `pubmonth`.
- d Java Development com Ant foi publicado em agosto de 2002, por isso temos demonstrado que o `pubmonth` valor de 200.208 é excluído da escala.

**NOTA** Consultas gama Nondate usar os termos de início e fim como o usuário entrou neles, sem modificação. Em outras palavras, o início e termos estão terminando não analisados. Termos de início e fim não deve conter espaço em branco, ou analisar falhar. Em nosso índice exemplo, o campo `pubmonth` não é um campo de data, é texto do formato AAAAMM.

### Manipulação de intervalos de datas

Quando uma consulta ampla é encontrado, o código do analisador primeiras tentativas para converter o início e fim para termos datas. Se os termos são datas válidas, de acordo com `DateFormat`. `SHORT` e análise lenient dentro da localidade padrão ou especificado, então as datas são convertidos para sua representação interna textual (ver secção 2.4 em `DateField`).

Se qualquer um dos dois termos não analisar como uma data válida, ambos são usados como é para uma gama textual.

O Pergunta'S `toString ()` saída é interessante para a data-range queries. Vamos analisar um para ver:

```
Query = QueryParser.parse ("modificado: [1/1/04 TO 12/31/04]",
                           "Sujeito", analisador);
System.out.println (query);
```

Isso gera algo estranho:

```
modificada: [0dowcq3k0 TO 0e3dwg0w0]
```

Internamente, todos os termos são de texto para Lucene, e as datas são representados em um léxico-graficamente ordenou formato de texto. Prop-, desde que nosso campo modificado foi indexada Erly como um `Data`, Tudo está bem, apesar desta saída de aparência estranha.

Controlar o locale data-análise

Para alterar o idioma utilizado para analisar o momento, construir um `QueryParser` instância e chamada `setlocale ()`. Tipicamente locale do cliente seria determinada e usados, ao invés de a localidade padrão. Por exemplo, em uma aplicação web, o `HttpServletRequest` objeto contém o local definido pelo navegador do cliente. Você pode usar esta localidade para controlar o local usado por parsing data em `QueryParser`, Como mostrado na listagem 3.3.

### Listagem 3.3 Usando a localidade do cliente em uma aplicação web

```
SearchServlet public class HttpServlet {
    protected void doGet (HttpServletRequest pedido,
                          HttpServletResponse resposta)
                          throws ServletException, IOException {

        Parser QueryParser QueryParser = new ("conteúdo",
                                              nova StandardAnalyzer ());

        parser.setLocale (request.getLocale ());

        try {
            Query = parser.parse (request.getParameter ("q"));
        } Catch (ParseException e) {
            / / ... lidar com exceção
        }

        / / ... exibir os resultados ...
    }
}
```

`QueryParser's setlocale` é uma maneira em que facilita a internacionalização Lucene-mento (muitas vezes abreviado I18N) preocupações. Análise de texto é outro, mais importante, lugar, importante, onde tais preocupações são manipulados. Questões I18N ainda são discutidos na seção 4.8.2.

### 3.5.6 consultas Frase

Termos entre aspas duplas criar um `PhraseQuery`. O texto entre as citações é analisada, assim a resultante `PhraseQuery` podem não ser exatamente a frase originalmente especificado. Este processo tem sido objecto de alguma confusão. Para exemplo, consultar o "Isto é algum \* Frase", Quando analisado pelo `StandardAnalyzer`, Analisa a um `PhraseQuery` usando a frase "uma frase". O `StandardAnalyzer` remove as palavras este e é porque elas correspondem a parada padrão lista de palavras (mais na seção 4.3.2 na `StandardAnalyzer`). Uma pergunta comum é por isso que o asterisco não é interpretado como uma consulta curinga. Tenha em mente que surarredondamento texto com aspas duplas faz com que o texto cercado de ser analisado e convertido em um `PhraseQuery`. Single prazo frases são otimizados para uma `TermQuery`. O código a seguir demonstra o efeito de análise em uma consulta frase expressão e de `TermQuery` otimização:

```
testPhraseQuery public void () throws Exception {
    Query q = QueryParser.parse ("\\" Esta é uma frase *\\"", "Campo", new StandardAnalyzer ());
    assertEquals ("analisados", "\\" Alguns frase\\\"", q.toString ("campo"));

    q = QueryParser.parse ("\\" termo \\\"", "campo", analisador);
    assertTrue ("reduzida a TermQuery", q instanceof TermQuery);
}
```

O fator de slop é zero a menos que você especificá-lo usando um til à direita (~) E o valor desejado slop inteiro. Porque a análise implícita de frases não podem coincidir com o que foi indexada, o fator de slop pode ser definido como algo diferente de zero automaticamente se não for especificado usando a notação til:

```
testSlop public void () throws Exception {
    Query q = QueryParser.parse (
        "\\" Frase exata \\\"", "campo", analisador);
    assertEquals ("slop zero",
        "\\" Exata frase\\\"", q.toString ("campo"));

    QueryParser qp = QueryParser nova ("campo", analisador);
    qp.setPhraseSlop (5);
    q = qp.parse ("\\" frase sloppy \\\"");
    assertEquals ("sloppy, implicitamente",
        "\\" Sloppy frase\\\"~5", q.toString ("campo"));
}
```

A sloppy PhraseQuery, Como observado, não exige que os termos coincidir na mesma da ordem. No entanto, um SpanNearQuery (Discutidos na seção 5.4.3) tem a capacidade de garantia de uma correspondência em ordem. Na seção 6.3.4, estendemos QueryParser e sub-constituem um SpanNearQuery quando as consultas frase são analisados, permitindo sloppy em frase partidas ordem.

### 3.5.7 Wildcard e consultas prefixo

Se um termo contém um asterisco ou um ponto de interrogação, é considerado um Curinga-Pergunta. Quando o termo contém apenas um asterisco, QueryParser otimiza-la a um PrefixQuery em seu lugar. Ambos prefixo e consultas curinga são escritos em minúscula por padrão, mas esse comportamento pode ser controlado:

```
testLowercasing public void () throws Exception {
    Query q = QueryParser.parse ("* PrefixQuery", "campo",
        analisador);
    assertEquals ("minúscula",
        "* Prefixquery", q.toString ("campo"));

    QueryParser qp = QueryParser nova ("campo", analisador);
    qp.setLowercaseWildcardTerms (false);
    q = qp.parse ("PrefixQuery *");
    assertEquals ("não lowercased",
        "* PrefixQuery", q.toString ("campo"));

}
```

Para desligar o lowercasing automático, você deve construir sua própria instância QueryParser em vez de usar o método de análise estática.

Curingas no início de um termo são proibidos usando QueryParser, Mas uma API-coded WildcardQuery pode usar curingas principal (à custa de per-desempenho). Seção 3.4.6 discute mais sobre o problema de desempenho, e seção 6.3.1 fornece uma maneira de proibir WildcardQuerys das expressões analisados se você deseja.

### 3.5.8 consultas Fuzzy

Um til à direita (~) Cria uma consulta fuzzy sobre o prazo anterior. O desempenho do mesmo advertências do desempenho se aplicam a WildcardQuery também se aplicam às consultas fuzzy e pode ser com deficiência com uma personalização semelhante ao discutido na seção 6.3.1.

### 3.5.9 Estimular consultas

Um carat (^) Seguido por um número de ponto flutuante define o fator de impulso para o precedendo consulta. Seção 3.3 discute consultas aumentar em mais detalhes. Por exemplo, a expressão de consulta junit ^ 2,0 testes define o junit TermQuery a um aumento de 2,0

e deixa o teste `TermQuery` no impulso padrão de 1,0. Você pode aplicar um impulsionar a qualquer tipo de consulta, incluindo os grupos entre parênteses.

### 3.5.10 Para QueryParse ou não QueryParse?

`QueryParser` é uma maneira rápida e sem esforço para dar aos usuários consulta poderosa cons- ção, mas não é certo para todos os cenários. `QueryParser` não pode criar todo o tipo de consulta que pode ser construído utilizando a API. No capítulo 5, detalhe que um punhado de API somente consultas que não têm `QueryParser` capacidade de expressão. Você deve manter em mente todas as possibilidades disponíveis ao expor de forma livre de análise de consulta para um usuário final; algumas consultas têm o potencial de gargalos de desempenho, e a sintaxe utilizada pelo built-in `QueryParser` pode não ser adequado para suas necessidades. Você pode exercer algum controle limitado por subclassing `QueryParser` (Ver secção 6.3.1).

Caso necessite de sintaxe de expressões diferentes ou capacidades para além do que `QueryParser` oferece, tecnologias como ANTLR7 JavaCC8 e são ótimas opções. Não discutimos a criação de um analisador de consulta personalizada, no entanto, o código-fonte para Lucene `QueryParser` está disponível gratuitamente para você pedir emprestado.

Muitas vezes você pode obter um meio termo, combinando um `QueryParser-Parsed` consulta com API-criado consultas como as cláusulas em uma `BooleanQuery`. Esta abordagem é demonstrado na seção 5.5.4. Por exemplo, os usuários precisam se restringir as pesquisas a uma categoria particular ou estreito-los para um intervalo de data, você pode ter o usuário interface separar as seleções em um seletor de categoria ou separados data- campos de intervalo.

## 3.6 Resumo

---

Lucene rapidamente fornece resultados de pesquisa altamente relevantes para consultas. A maioria das aplicações

ções só precisa de uma classes Lucene poucos e métodos para permitir a pesquisa. O mais coisas fundamentais para você tomar a partir deste capítulo são uma compreensão do os tipos básicos de consulta (dos quais `TermQuery`, `RangeQuery` E `BooleanQuery` são os os primários) e como acessar os resultados da pesquisa.

Embora possa ser um pouco assustador, fórmula de pontuação do Lucene (juntamente com a formato de índice discutidos no Apêndice B e os algoritmos eficientes) fornece a magia de retornar os documentos mais relevantes primeiro. Lucene `QueryParser` analisa expressões legível consulta, dando poder de pesquisa rico de texto completo para usuários finais. `QueryParser` imediatamente satisfaz a maioria dos requisitos de candidatura;

---

<sup>7</sup> <http://www.antlr.org>.

<sup>8</sup> <http://javacc.dev.java.net>.

no entanto, não vem sem ressalvas, tenha certeza que você entender o áspero bordas. Grande parte da confusão a respeito `QueryParser` decorre de inesperado interações análise; capítulo 4 entra em grandes detalhes sobre a análise, incluindo mais no `QueryParser` questões.

E sim, há mais para pesquisar do que nós cobrimos neste capítulo, mas compreender as bases é crucial. Capítulo 5 investiga mais do Lucene características elaboradas, como restringir (ou filtragem) a busca de espaço que-Ries e classificar os resultados da pesquisa por valores de campo; capítulo 6 explora os inúmeros maneiras que você pode ampliar os recursos em busca Lucene para classificação personalizada e análise de consulta.

# Análise

---

4

## Este capítulo aborda

- Compreender o processo de análise
- Explorando questões QueryParser
- Escrever analisadores personalizadas
- Manipulação de línguas estrangeiras

Análise, no Lucene, é o processo de conversão de texto em seu campo mais fundamental: representação indexados tal, termos. Esses termos são usados para determinar o documentos correspondem a uma consulta durante as buscas. Por exemplo, se esta frase foram indexados

em um campo (vamos supor que tipo de `Field.Text`), Os termos podem começar com para e exemplo, e assim por diante, como termos separados em seqüência. Um analisador é um encapsulamento-

ção do processo de análise. Um analisador tokeniza texto através da realização de qualquer número

de operações sobre ela, o que poderia incluir palavras extrair, descartando pontuação, acentos remoção de caracteres, lowercasing (também chamado normalizing), remoção de palavras comuns, reduzindo palavras para uma forma de raiz (decorrentes), ou

mudando as palavras para a forma básica (lemmatization). Este processo também é chamado tokenization, e os pedaços de texto extraídas de um fluxo de texto são chamados tokens.

Símbolos, combinado com o seu nome do campo associado, são termos.

Principal objetivo Lucene é facilitar informações recuperação. A ênfase na recuperação é importante. Você quer jogar gobs de texto no Lucene e tê-los será amplamente pesquisado por palavras individuais dentro desse texto. Para que Lucene saber o que "palavras" são, análises o texto durante a indexação, extraíndo-o em termos. Esses termos são os blocos de construção primitiva para a pesquisa.

Escolhendo o analisador de direito é uma decisão crucial de desenvolvimento com Lucene. Um tamanho não cabe tudo quando se trata de escolher um analisador. A linguagem é uma fator na escolha de um analisador, porque cada um tem suas próprias características únicas. Outro

fator a considerar na escolha de um analisador é o domínio do texto a ser analisados, diferentes indústrias têm uma terminologia diferente, acrônimos e abreviações que podem merecer atenção. Embora nós apresentamos muitas das considerações para a escolha de analisadores, analisador não só será suficiente para todas as situações. É possi-

nenhum que ble do built-in opções de análise são adequadas para suas necessidades, e você vai precisar investir na criação de uma solução de análise de costume; agradavelmente, Lucene

blocos de construção tornam isso muito fácil.

Uma das melhores perguntas que você pode perguntar como você contempla o processo de análise

é: "O que o Google faça?" algoritmos reais do Google são proprietários e mantido relativamente secreta, mas os resultados das pesquisas dão algumas dicas. Pesquisar para a frase "ser ou não ser" com e sem as aspas é uma experiência de diversão. Sem as aspas, o Google considera apenas a palavra (no momento do writing) é, surpreendentemente, não; Um que joga fora os outros como sendo muito comum.

No entanto, o Google não jogue fora essas parar as palavras durante a indexação, como você pode

---

<sup>1</sup> Curiosamente, o primeiro resultado (no momento da escrita) para "ser ou não ser" (sem aspas) no Google é o site "Estou Hot or Not?" a sério!

ver por buscar a frase entre aspas. Este é um fenômeno interessante: Um número espantoso de palavras de parada estão sendo indexados! Como o Google realizar a indexação de cada palavra de cada página web na Internet sem acabando de armazenamento? Um analisador Lucene baseado existe que fornece uma solução a esta questão, como vamos discutir.

Neste capítulo, vamos cobrir todos os aspectos do processo de análise Lucene, incluindo como e onde usar analisadores, analisadores de que o built-in faz, e como escrever seus próprios analisadores personalizados usando os blocos de construção fornecido pelo núcleo Lucene API.

## 4.1 Usando analisadores

Antes de entrarmos os detalhes do que se esconde dentro de um analisador, vamos olhar para como um analisador é usado dentro Lucene. A análise ocorre em dois pontos: durante indexação e quando usar `QueryParser`. Em duas seções a seguir, detalhamos como um analisador é usado nesses cenários.

Antes de começarmos com todos os detalhes do código, olhar para listagem 4.1 para ter uma idéia de que o processo de análise é tudo. Duas frases são analisados, cada um por quatro dos built-in analisadores. As frases são "A ligeira raposa marrom saltou sobre o preguiçoso cães" e "XY & Z Corporation - xyz@example.com". Cada token é mostrado entre colchetes para fazer as separações aparente. Durante a indexação, os tokens extraídos durante a análise são os termos indexados. E, mais importante, os termos indexados são os termos que são pesquisáveis!

### Listagem 4.1 Efeitos analisador Visualizing

```
Analisando "A ligeira raposa marrom saltou sobre o cão preguiçoso"
WhitespaceAnalyzer:
[A] [rápida] [brown] [fox] [saltou] [mais] [o] [preguiçoso] [cães]

SimpleAnalyzer:
[A] [rápida] [brown] [fox] [saltou] [mais] [o] [preguiçoso] [cães]

StopAnalyzer:
[Rápida] [brown] [fox] [saltou] [mais] [preguiça] [cães]

StandardAnalyzer:
[Rápida] [brown] [fox] [saltou] [mais] [preguiça] [cães]

Analisando "XY & Z Corporation - xyz@example.com"
WhitespaceAnalyzer:
[XY e Z] [Corporation] [-] [xyz@example.com]
```

```

SimpleAnalyzer:
[Xy] [z] [empresa] [xyz] [exemplo] [com]

StopAnalyzer:
[Xy] [z] [empresa] [xyz] [exemplo] [com]

StandardAnalyzer:
[Xy e z] [empresa] [xyz@example.com]

```

---

O código que gerou esta saída do analisador é mostrado mais tarde, na listagem 4.2. A poucos coisas interessantes acontecem neste exemplo. Olhe como a palavra o é tratada, e também o nome da empresa XY e Z e os xyz@example.com endereço de e-mail; olhar para o caráter especial hífen (-) eo caso de cada token. Seção 4.2.3 explica mais sobre os detalhes do que aconteceu.

Lucene não faz os resultados do processo de análise visível para o usuário final. Termos retirados do texto original são indexados e são combinados durante a procura-ing. Ao pesquisar com `QueryParser`, O processo de análise acontece novamente em Para garantir as melhores correspondências possíveis.

#### 4.1.1 análise de indexação

Durante a indexação, um `Analisador` instância é entregue ao `IndexWriter` desta maneira:

```

Analizador analisador = new StandardAnalyzer ();
Escritor IndexWriter IndexWriter = new (diretório, analisador, true);

```

Neste exemplo, usamos o built-in `StandardAnalyzer`, Um dos vários disponíveis dentro da biblioteca Lucene core. Cada campo tokenized de cada documento indexado com o `IndexWriter` instância usa o analisador especificado. Dois especiais `Campo` tipos são designados para ser indexado: `Texto` e `UnStored`.

**NOTA**      `Field.Text (String, String)` cria um tokenized e armazenadas campo. A certeza da original `Corda` valor é armazenado. No entanto, a saída do designado `Analisador` dita o que é indexado.

O código a seguir demonstra a indexação de um documento com esses dois tipos de campo:

```

Documento doc = new Document ();
doc.add (Field.Text ("title", "Este é o título"));
doc.add (Field.UnStored ("Conteúdo", "... o conteúdo do documento ..."));
writer.addDocument (doc);

```

Ambos "Título" e "Conteúdo" são analisadas usando a Analisador exemplo, desde ao IndexWriter. No entanto, se um documento individual tem análise especial necessidades, o analisador pode ser especificado em uma base por documento, assim:

```
writer.addDocument (doc, analisador);
```

Durante a indexação, a granularidade de escolha analisador está no IndexWriter ou per-Dокументo nível. Parece que cada campo pode merecer uma análise única e que mesmo isso per-Documento análise é muito claro grãos. Analisadores têm acesso ao nome do campo que está sendo analisada, de modo mais refinado, análise de campo específico é possível, discutimos por campo de análise na seção 4.4.

Field.Keyword campos indexados não são tokenized. A Field.Keyword campo é indexado como um termo único com o valor exatamente como fornecido. Depois de indexados, porém, há diferença em um prazo de Field.Keyword e um termo criado a partir de uma analyzer, ambos são termos sem nenhum conhecimento de como eles foram indexadas. Isso pode levar ao comportamento problemático quando você está usando QueryParser. Como podemos citar novamente em próxima seção.

#### 4.1.2 análise QueryParser

O Analisador é a chave para os termos indexados. Como você viu no capítulo 3, você precisa ter a certeza de consulta sobre os termos exactos indexados a fim de encontrar os documentos (que coberto QueryParser análise de expressão e detalhes de uso nas seções 3.1.2 e 3.5). Quando você estiver usando API-criado, como consultas TermQuery, É o desen-oper responsabilidade para garantir que os termos utilizados vai coincidir com o que foi indexado.

Apresentando aos usuários uma opção de forma livre de consulta é muitas vezes o que lhe for perguntado de implementar, e QueryParser vem a calhar para o processamento introduzidos pelo utilizador expressões de consulta. QueryParser usa um analisador de fazer o seu melhor trabalho para coincidir com a termos que foram indexadas. Um analisador é especificado na estática analisar método:

```
Query = QueryParser.parse expressão ("conteúdo", analisador);
```

Ou, se você estiver usando uma QueryParser exemplo, o analisador é especificado na construtor:

```
Parser QueryParser QueryParser = new ("conteúdo", analisador);
query = parser.parse (expressão);
```

QueryParser análises peças individuais de expressão, não a expressão como um todo, que pode incluir operadores, parênteses, e expressão especial outras sintaxe para denotar gama, wildcard, e as buscas fuzzy.

`QueryParser` analisa todos os textos igualmente, sem o conhecimento de como foi indexados. Esta é uma questão particularmente espinhosa quando você está consultando para campos que

foram indexados como `Field.Keyword`. Nós resolver esta situação na seção 4.4.

Você deve usar o mesmo analisador com `QueryParser` que você usou durante a indexação? A curto, mais preciso responder, é: "depende". Se você ficar com o básicos built-in analisadores, então você provavelmente vai ser muito bem usando o mesmo analisador em

ambas as situações. No entanto, quando você estiver usando analisadores mais sofisticados, quirky

casos podem surgir em que usar analisadores diferentes entre indexação e Consulta-Analisador é o melhor. Discutimos esta questão em mais detalhes na seção 4.6.

#### 4.1.3 Análise versus análise: quando um analisador não é apropriado

Um ponto importante sobre analisadores é que eles são usados internamente para campos marcada para ser tokenized. Documentos, tais como HTML, Microsoft Word, XML, e outros, contêm meta-dados como autor, título, data da última modificação, e potencialmente muito mais. Quando você está indexação de documentos ricos, esses dados devem meta-  
ser separados e indexados como campos separados. Analisadores são usados para analisar um es-  
campo específicos de cada vez e quebrar as coisas em tokens apenas dentro do campo, criando novos campos não é possível dentro de um analisador.

Analisadores não ajudam na separação de campo, porque seu escopo é o de lidar com um único campo de cada vez. Em vez disso, análise estes documentos antes da análise é necessário. Por exemplo, é uma prática comum para separar pelo menos o `<title>` e `<body>` de documentos HTML em campos separados. Nestes casos, os documentos deve ser analisado, ou pré-processados, em blocos separados do texto que representam cada de campo. Capítulo 7 abrange vários tipos de documentos específicos e fornece opções para indexá-las, mas também discute análise de vários tipos de documentos em detalhe.

## 4.2 Analisando o analisador

---

, A fim de apreciar e compreender como a análise textual do Lucene obras, precisamos abrir o capô e mexer um pouco. Porque é possível que você estará construindo a sua própria analisadores, conhecendo a arquitetura e construção blocos fornecida é crucial.

O `Analizador` classe é a classe base. Muito elegante, transforma-se em um fluxo de texto de tokens, literalmente um `TokenStream`. A assinatura do método único exigido implementadas por analisadores é

```
pública TokenStream tokenStream (String fieldName, Reader leitor)
```

Observe que um analisador pode ser usado para off-chave no nome do campo. Como os nomes de campo são arbitrários e dependente da aplicação, todas as built-in analisadores ignorar o campo nome. Analisadores personalizados são livres de utilizar o nome do campo ou, mais facilmente, para usar o especial `PerFieldAnalyzerWrapper` que os delegados da análise para cada campo de analisadores você associa com nomes de campo (cobertura detalhada está na secção 4.4).

Vamos começar "simplesmente" com a `SimpleAnalyzer` e ver o que faz funcionar. O seguinte código é copiado diretamente da base de código do Lucene:

```
public final class SimpleAnalyzer estende Analyzer {
    pública TokenStream tokenStream (String fieldName, Reader leitor) {
        retorno LowerCaseTokenizer novo (leitor);
    }
}
```

O `LowerCaseTokenizer` divide o texto em nonletters (determinado pelo `Personagem.isLetter`), Removendo caracteres nonletter e, fiel ao seu nome, cada lowercasing personagem. A `TokenStream` é uma classe enumerador do tipo que retorna sucessivas Símbolos, retornando `nulo` quando o fim foi atingido (ver listagem 4.3, onde `Analyzer-Utils` enumera os tokens retornado).

Nas seções seguintes, vamos dar uma olhada detalhada em cada um dos jogadores principais utilizado por analisadores, incluindo `Símbolo` eo `TokenStream` da família.

#### 4.2.1 O que é um token?

Um fluxo de tokens é a saída fundamental do processo de análise. Durante indexação, campos designados para tokenization são processados com o especificado analyzer, e cada token é escrito para o índice como um termo. Esta distinção entre tokens e os termos podem parecer confuso no início. Vamos ver o que forma uma `Símbolo`; Vamos voltar à forma como isso se traduz em um termo.

Por exemplo, vamos analisar o texto "A ligeira raposa marrom". Cada repre-token presenta uma palavra individual daquele texto. Um token carrega consigo um valor de texto (a palavra em si), bem como alguns meta-dados: o início eo final compensa no texto original, uma tipo token, e um incremento de posição. A Figura 4.1 mostra os detalhes do token stream analisar essa frase com o `SimpleAnalyzer`.

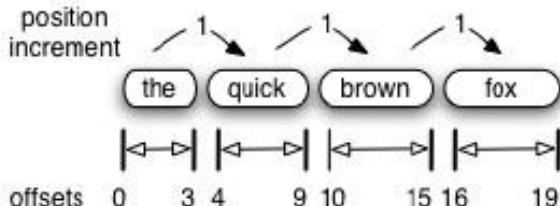


Figura 4.1  
Fluxo de token com posicional  
e informações de deslocamento

O início de deslocamento é a posição do caractere no texto original, onde o texto token começa, eo deslocamento final é a posição logo após o último caractere do token texto. O tipo de token é um `Corda`, Padronizando "Palavra", Que você pode controlar e uso no processo de filtragem de token, se desejar. Como o texto é tokenized, a posição relative o token anterior é registrado como o valor de incremento de posição. Todos os built-in tokenizers deixar o incremento de posição com o valor padrão de 1, indicating que todas as fichas estão em posições sucessivas, um após o outro.

### Tokens em termos

Após o texto ser analisados durante a indexação, cada ficha é enviada para o índice como um prazo. O incremento de posição é a apenas adicionais meta-dados associados com o token realizada para o índice. Início e fim offset bem como o tipo token são descartados, estes são apenas utilizados durante o processo de análise.

### Incrementos de posição

O valor de incremento token posição relaciona o token atual para a anterior.

Geralmente, os incrementos de posição são 1, indicando que cada palavra é em um único e posição sucessivas no campo. Posição incrementos fator diretamente no desem-consultas frase ing (ver secção 3.4.5) e consultas span (ver secção 5.4), que confiar em saber até onde os termos são um do outro dentro de um campo.

Incrementos posição superior a 1 permitem falhas e pode ser usado para indicar onde as palavras foram removidas. Consulte a seção 4.7.1 para um exemplo de stop-palavra remoção que deixa lacunas usando incrementos posição.

Um token com um incremento de posição zero coloca o token na mesma posição como o token anterior. Analisadores que injetam aliases palavra pode usar um incre-posição mento de zero para os aliases. O efeito é que as consultas frase trabalho independentemente do que alias foi usado na consulta. Consulte as nossas `SynonymAnalyzer` na seção 4.6 para uma exemplo que usa incrementos posição de zero.

## 4.2.2 TokenStreams sem censura

Há dois estilos diferentes de `TokenStream`s: `Tokenizer` e `TokenFilter`. A boa generalização para explicar a diferença é que `Tokenizers` lidar com indi-caracteres individuais, e `TokenFilters` lidar com as palavras. Figura 4.2 mostra este arqui-tura graficamente.

A `Tokenizer` é uma `TokenStream` tokeniza que a entrada de um `Leitor`. Quando você está a indexação de um `Corda` através `Field.Text (String, String)` ou De campo.

`UnStored (String, String)` (Isto é, o campo indexado construtores que aceitam um `Corda`), Envolve o Lucene `Corda` em um `StringReader` para tokenization.

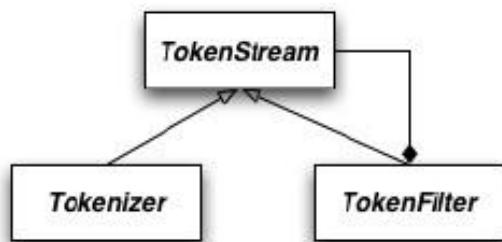


Figura 4.2  
TokenStream arquitetura:  
TokenFilters um filtro  
TokenStream.

O segundo estilo de `TokenStream`, `TokenFilter`, Permite cadeia `TokenStream` s juntos. Este poderoso mecanismo faz jus ao seu homônimo como um filtro de fluxo. A `TokenStream` é alimentado em um `TokenFilter`, Dando o filtro a chance de adicionar, remover, ou alterar o fluxo que passa através.

A Figura 4.3 mostra o pleno `TokenStream` hierarquia de herança dentro Lucene. Observe o padrão composto usado por `TokenFilter` para encapsular uma outra `TokenCórrego` (O que poderia, é claro, ser outra `TokenFilter`). Tabela 4.1 fornece descrições detalhadas para cada uma das classes mostrado na figura 4.3.

Tabela 4.1 Analisador de blocos de construção previsto no core API do Lucene

Nome da classe	Descrição
<code>TokenStream</code>	Classe base com <code>next ()</code> e <code>close ()</code> métodos.
<code>Tokenizer</code>	<code>TokenStream</code> cuja entrada é um Leitor.
<code>CharTokenizer</code>	Classe pai de tokenizers baseados em caracteres, com resumo <code>isTokenChar ()</code> método. Emite tokens de blocos contíguos, quando <code>isTokenChar == true</code> . Também fornece a capacidade para normalizar (por exemplo, em letras minúsculas) caracteres. Tokens estão limitados a um tamanho máximo de 255 caracteres.
<code>WhitespaceTokenizer</code>	<code>CharTokenizer</code> com <code>isTokenChar ()</code> verdadeiro para todos os caracteres nonwhitespace.
<code>LetterTokenizer</code>	<code>CharTokenizer</code> com <code>isTokenChar ()</code> verdadeiro quando <code>Character.isLetter</code> é verdade.
<code>LowerCaseTokenizer</code>	<code>LetterTokenizer</code> que normaliza todos os caracteres para minúsculas.
<code>StandardTokenizer</code>	Sofisticadas gramática baseada tokenizer, emitindo tokens de alto nível como os tipos endereços de correio electrónico (ver secção 4.3.2 para mais detalhes). Cada token é emitido marcado com um tipo especial, alguns dos quais são tratados de forma especial por <code>StandardFilter</code> .
<code>TokenFilter</code>	<code>TokenStream</code> cuja entrada é outra <code>TokenStream</code> .
<code>LowerCaseFilter</code>	Minúsculas texto token.

continua na página seguinte

Tabela 4.1 Analisador de blocos de construção previsto no core API do Lucene (Continuação)

Nome da classe	Descrição
StopFilter	Remove as palavras que existem em um conjunto de palavras fornecida.
PorterStemFilter	Haste cada token usando o algoritmo Porter decorrentes. Por exemplo, país e países tronco para ambos os countri.
StandardFilter	Projetado para ser alimentado por um StandardTokenizer. Remove pontos de siglas e 'S (Apóstrofo seguido por S) de palavras com apóstrofos.

Aproveitando-se da TokenFilter padrão de encadeamento, você pode construir complexo analisadores de simples Tokenizer/TokenFilter blocos de construção. Tokenizers começam o processo de análise agitando a entrada de caracteres em tokens (Principalmente essas correspondem às palavras do texto original). TokenFilters, em seguida, assumir o permanecem da análise, inicialmente envolvendo uma Tokenizer e sucessivamente embrulho aninhadas TokenFilters. Para ilustrar isso no código, aqui é o coração do StopAnalyzer:

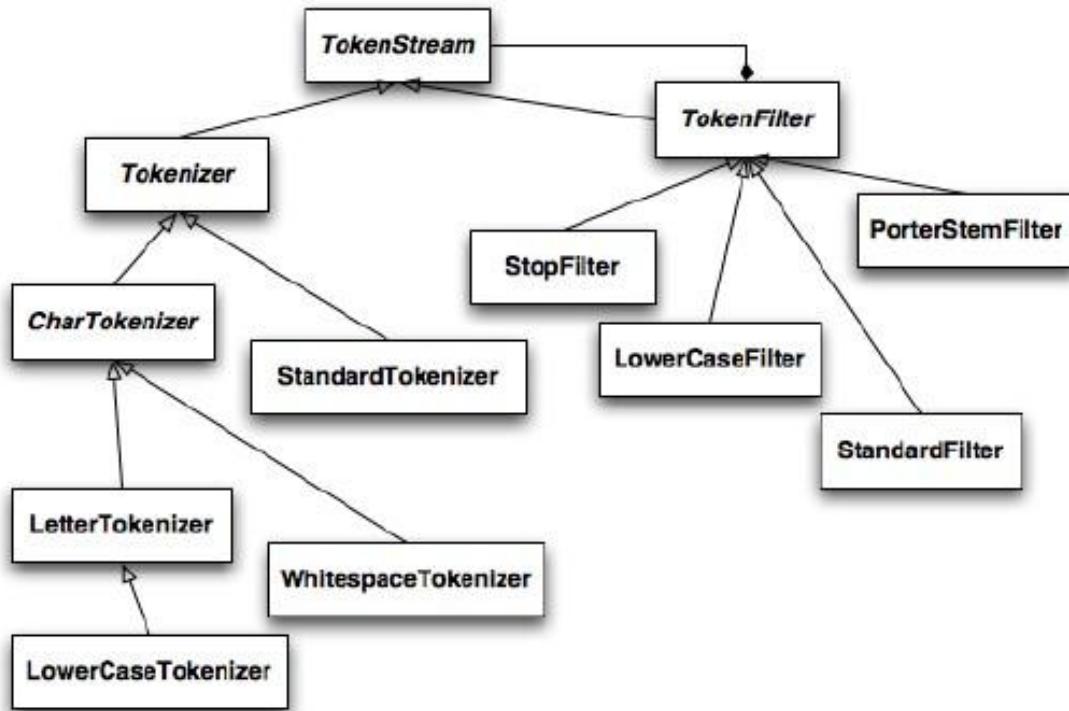


Figura 4.3 TokenStream hierarquia de classe

```
pública TokenStream tokenStream (String fieldName, Reader leitor) {
    return new StopFilter (
        , nova LowerCaseTokenizer (leitor)
        stopTable);
}
```

Em `StopAnalyzer`, Um `LowerCaseTokenizer` alimenta um `StopFilter`. O `LowerCaseTokenizer` emite tokens que são letras adjacentes no texto original, cada lowercasing dos personagens no processo. Caracteres Nonletter forma fronteiras token e não são incluídos em qualquer token emitido. Seguindo esta palavra `tokenizer` e de menor `CASER`, `StopFilter` remove as palavras em uma lista de stop word-(ver secção 4.3.1).

Buffer é um recurso que é comumente necessária na `TokenStream` implementações. De baixo nível `Tokenizers` fazer isso para buffer de caracteres para formar tokens no limites, tais como caracteres em branco ou nonletter. `TokenFilters` que emitem tokens adicionais para o fluxo que está filtrando deve fila de um token de entrada e os adicionais e emitem-los um de cada vez, a nossa `SynonymFilter` no secção 4.6 é um exemplo de um filtro de filas.

#### 4.2.3 analisadores Visualizing

É importante entender o que vários analisadores de fazer com seu texto. Vendo o efeito de um analisador é uma ajuda poderosa e imediata a este entendimento. -Listaing 4.2 fornece uma maneira rápida e fácil de obter feedback visual sobre os quatro primary built-in analisadores em um par de exemplos de texto. `AnalyzerDemo` inclui dois frases pré-definidos e uma série de quatro analisadores estamos focando neste seção. Cada frase é analisada por todos os analisadores, com saída entre parênteses para indicar os termos que seriam indexados.

##### Listagem 4.2 AnalyzerDemo: ver análise em ação

```
/ **
 * Adaptado a partir do código que apareceu pela primeira vez em um artigo
java.net
 * Escrito por Erik
 */
public class {AnalyzerDemo
    String private static final [] = {exemplos
        "A ligeira raposa marrom saltou sobre o cão preguiçoso",
        "XY & Z Corporation - xyz@example.com"
    };

    Analyzer private static final [] = new analisadores Analyzer [] {
        nova WhitespaceAnalyzer (),
        nova SimpleAnalyzer (),
        nova StopAnalyzer (),
        nova StandardAnalyzer ()
    };
}
```

```
public static void (String [] args) throws IOException {principal
    / / Use as cordas exemplo embutido, a menos
    Argumentos de linha de comando / / são especificadas, então usá-los.
    String [] strings = exemplos;
    if (args.length> 0) {
        strings = args;
    }

    for (int i = 0; i strings.length <; i + +) {
        analisar (strings [i]);
    }
}

static void privada analisar (String texto) throws IOException {
    System.out.println ("Analizando \\" "+ texto +" \\ \"");
    for (int i = 0; i analyzers.length <; i + +) {
        Analisador analisador analisadores = [i];
        String nome = analyzer.getClass () getName ();
        name.substring nome = (name.lastIndexOf (".") + 1);
        System.out.println ("\" + nome + ":" );
        System.out.print ("");
        AnalyzerUtils.displayTokens (analisador, texto);
        System.out.println ("\n");
    }
}

}

A verdadeira diversão acontece em AnalyzerUtils (Listagem 4.3), onde o analisador é aplicada ao texto e os sinais são extraídos. AnalyzerUtils passa o texto a uma analisador sem indexá-lo e puxa os resultados de uma forma semelhante ao que acontece durante o processo de indexação sob a capa de IndexWriter.
```

#### Listagem 4.3 AnalyzerUtils: aprofundar em um analisador

```
pública AnalyzerUtils classe {
    Token public static [tokensFromAnalysis]
        (Analisador analisador, String texto) throws IOException {
    Stream = TokenStream
        analyzer.tokenStream ("conteúdo", new StringReader (texto));
    ArrayList tokenList = new ArrayList ();Invoke análise
    while (true) {processo
        Token token = stream.next ();
        if (token == null) break;

        tokenList.add (token);
    }

    retorno (Token []) tokenList.toArray (novo Token [0]);
}
```

```

public static void displayTokens
    (Analizador analisador, String texto) throws IOException {
    Token [] tokens = tokensFromAnalysis (texto analisador);

    for (int i = 0; i < tokens.length; i + +) {
        Token token = tokens [i];

        System.out.println ("[" + token.termText () + "]");
    }
}

// ... outros métodos introduzidos mais tarde ...
}

```

Tokens de saída cercado por suportes

Geralmente você não iria invocar o analisador de `tokenStream` método explicitamente exceto para este tipo de fins de diagnóstico ou informativos (o nome do campo conteúdo é arbitrária na `tokensFromAnalysis ()` método). Temos, no entanto, cobrir uma utilização deste método de produção para destacar consulta na seção 8.7, página 300.

`AnalyzerDemo` produziu a saída mostrada na listagem 4.1. Alguns pontos-chave para nota são os seguintes:

- `WhitespaceAnalyzer` não minúsculas, à esquerda no painel de instrumentos, e fez o bare mínimo de tokenizing nos limites de espaço em branco.
- `SimpleAnalyzer` à esquerda no que pode ser considerada irrelevante (stop) palavras, fez minúsculas e tokenize nos limites de caracteres não-alfabéticos.
- Ambos `SimpleAnalyzer` e `StopAnalyzer` mangled o nome da corporação por divisão XY e Z e remover o comercial.
- `StopAnalyzer` e `StandardAnalyzer` jogou fora as ocorrências da palavra da.
- `StandardAnalyzer` manteve o nome da corporação intacta e lowercased-lo, removido o traço, e manteve o endereço de e-mail juntos. Nenhum outro built-in analisador é esta completa.

Recomendamos manter um utilitário como este útil para ver o que emitem tokens de seu analisadores de escolha. Na verdade, ao invés de escrever isso mesmo, você pode usar o nosso

`AnalyzerUtils` ou o `AnalyzerDemo` código para a experimentação. O `AnalyzerDemo` aplicativo permite que você especifique uma ou mais strings a partir da linha de comando a ser analisados em vez de os exemplo incorporado:

```
% Java lia.analysis.AnalyzerDemo "No Fluff, Just Stuff"
Analizando "No Fluff, Just Stuff"
```

```

org.apache.lucene.analysis.WhitespaceAnalyzer:
[No] [Fluff] [Just] [Coisas]

org.apache.lucene.analysis.SimpleAnalyzer:
[No] [fluff] [apenas] [o material]

org.apache.lucene.analysis.StopAnalyzer:
[Fluff] [apenas] [o material]

org.apache.lucene.analysis.standard.StandardAnalyzer:
[Fluff] [apenas] [o material]

```

Vamos agora olhar mais profundo sobre o que compõe um Símbolo.

### Olhar para dentro de tokens

`TokenStreams` pode criar Símbolos, e `TokenFilters` podem acessar seus meta-dados. Para demonstrar acesso token meta-dados, acrescentou o `displayTokensWithFullDetails` método de utilitário no `AnalyzerUtils`:

```

public static void displayTokensWithFullDetails
    (Analizador analisador, String texto) throws IOException {
    Token [] tokens = tokensFromAnalysis (texto analisador);

    int posição = 0;

    for (int i = 0; i < tokens.length; i + +)
        Token token = tokens [i];

    int = incremento token.getPositionIncrement ();

    if (incremento > 0) {
        posição = posição + incremento;
        System.out.println ();
        System.out.print (posição + ":");

    }

    System.out.println ("[" + token.termText () + ":" +
        token.startOffset () + "-" +
        token.endOffset () + ":" +
        token.type () + "]");
}

}

```

Nós exibir todas as informações token na frase exemplo usando `SimpleAnalyzer`:

```

public static void (String [] args) throws IOException {principal
    displayTokensWithFullDetails (novo SimpleAnalyzer (),
        "A ligeira raposa marrom ....");
}

```

Aqui está a saída:

```

1: [A: 0 -> 3: Palavra]
2: [Rápida: 4 -> 9: Palavra]
3: [Marrom: 10 -> 15: Palavra]
4: [Raposa: 16 -> 19: Palavra]

```

Cada token está em uma posição sucessivas em relação ao anterior (observado pelo incrementando os números 1, 2, 3 e 4). A palavra o começa em 0 e termina antes offset 3 no texto original. Cada um dos tokens tem um tipo de palavra. Nós apresentar um similar, mas mais simples de visualização, de incrementos posição token no seção 4.6.1, e nós fornecemos uma visualização de tokens partilha a mesma posição.

Para que servem os offsets de início e fim? Os valores de início e fim offset não são usadas no núcleo do Lucene. Eles são de uso menos? Não inteiramente. O highlighter termo discutido na seção 8.7 usa um Token-Córrego e os consequentes Símbolos fora da indexação para determinar onde em um bloco de texto para começar e terminar destaque, permitindo que as palavras que busca para usuários para se destacar nos resultados de busca.

Token tipo de utilidade  
Você pode usar o token tipo valor especial para denotar tipos de tokens lexical. Nos bastidores da StandardAnalyzer é uma StandardTokenizer que analisa os de texto recebidas em diferentes tipos com base em uma gramática. Analisando a frase "Eu vou e-mail você em xyz@example.com" com StandardAnalyzer produz este saída interessante:

```

1: [Eu vou: 0 -> 4: <APOSTROPHE>]
2: [E: 5 -> 6: <ALPHANUM>]
3: [Mail: 7 -> 11: <ALPHANUM>]
4: [Você: 12 -> 15: <ALPHANUM>]
5: [Xyz@example.com: 19 -> 34: <EMAIL>]

```

Observe o tipo de cada token token. O token Eu vou tem um apóstrofo, que StandardTokenizer avisos, a fim de mantê-lo em conjunto como uma unidade, e também para o endereço de e-mail. Nós cobrimos o outro StandardAnalyzer efeitos na seção 4.3.2. StandardAnalyzer é o único analisador de built-in que utiliza os dados do tipo token. Analisadores nosso Metaphone e sinônimo, nas seções 4.5 e 4.6, fornecem um outro exemplo de uso do tipo token.

#### 4.2.4 Filtragem de ordem pode ser importante

A ordem dos eventos pode ser muito importante durante a análise. Cada passo pode contar com o trabalho de uma etapa anterior. Um bom exemplo é o de parar de palavra remoção. StopFilter faz um case-sensitive look-up de cada token em um conjunto de stop

palavras. Ele se baseia em ser alimentado lowercased tokens. Como exemplo, podemos escrever uma primeira funcionalmente equivalente `StopAnalyzer` variante; nós vamos segui-lo com uma variante defeituosa que inverte a ordem dos passos:

```
public class StopAnalyzer2 extends Analyzer {
    private Set<String> stopwords;

    public StopAnalyzer2() {
        stopwords = StopFilter.makeStopSet(StopAnalyzer.ENGLISH_STOP_WORDS);
    }

    public StopAnalyzer2(String[] stopwords) {
        this.stopWords = StopFilter.makeStopSet(stopwords);
    }

    public TokenStream tokenStream(String fieldName, Reader leitor) {
        return new StopFilter(
            new LowerCaseFilter(new LetterTokenizer(leitor)),
            stopwords);
    }
}
```

`StopAnalyzer2` usa um `LetterTokenizer` alimentando uma `LowerCaseFilter`, ao invés de apenas um `LowerCaseTokenizer`. A `LowerCaseTokenizer` no entanto, tem um desempenho vantagem, uma vez que minúsculas como tokeniza, ao invés de dividir o processo em duas etapas. Este caso de teste prova que a nossa `StopAnalyzer2` funciona como esperado, por utilização `AnalyzerUtils.tokensFromAnalysis` e afirmando que a palavra `stop` o foi removido:

```
testStopAnalyzer2 public void() throws Exception {
    Token[] tokens =
        AnalyzerUtils.tokensFromAnalysis(
            novo StopAnalyzer2(), "O marrom rápido ...");

    assertTrue(
        AnalyzerUtils.tokensEqual(tokens,
            new String[] {"rápido", "Marrom"}));
}
```

Nós adicionamos um método auxiliar da unidade-teste para a nossa `AnalyzerUtils` que afirma `tokens` coincidir com uma lista de espera:

```
assertTokensEqual public static void(
    Token[] tokens, String[] strings) {
    Assert.assertEquals(tokens.length, strings.length);

    for (int i = 0; i < tokens.length; i++) {
        Assert.assertEquals("index" + i,
```

```
        strings [i] tokens, [i] termText ());
    }
}
```

Para ilustrar a importância que a ordem pode fazer com a filtragem de token, nós escrita de um analisador de falhas que troca a ordem das `StopFilter` e o `InferiorCaseFilter`:

```
/ **
 * Pare de palavras não necessariamente removido devido a filtragem fim
 */
public class StopAnalyzerFlawed estende Analyzer {
    privada stopwords Set;

    pública StopAnalyzerFlawed () {
        stopwords =
            StopFilter.makeStopSet (StopAnalyzer.ENGLISH_STOP_WORDS);
    }

    pública StopAnalyzerFlawed (String [] stopwords) {
        this.stopWords = StopFilter.makeStopSet (stopwords);
    }

    / **
     * Ordenação erro aqui
     */
    pública TokenStream tokenStream (String fieldName, Reader leitor) {
        retorno nova LowerCaseFilter (
            nova StopFilter (novo leitor LetterTokenizer (),
                stopwords));
    }

}
```

O `StopFilter` presume todos os tokens já foram lowercased e faz um case-sensitive de pesquisa. Outro caso de teste mostra que O não foi removido (é o primeiro símbolo da saída do analisador), mas foi em minúsculo:

```
testStopAnalyzerFlawed public void () throws Exception {
    Token [] tokens =
        AnalyzerUtils.tokensFromAnalysis (
            nova StopAnalyzerFlawed (), "O marrom rápido ...");

    assertEquals ("a", tokens [0] termText ());
}
```

Lowercasing é apenas um exemplo onde a ordem pode importar. Os filtros podem assumir processamento anterior foi feito. Por exemplo, o `StandardFilter` é projetado para ser usado em conjunto com `StandardTokenizer` e não faria sentido com qualquer outro `TokenStream` alimentá-lo. Também pode haver considerações sobre o desempenho

quando você requisita o processo de filtragem. Considere um analisador que remove parar palavras e também injeta sinônimos para o token de fluxo seria mais eficientes para remover as palavras primeira parada para que o filtro de injeção seria sinônimo têm menos termos a considerar (ver secção 4.6 para um exemplo detalhado).

## 4.3 Utilizar os analisadores de built-in

Lucene inclui vários built-in analisadores. As principais são mostrados na tabela 4.2. Vamos deixar a discussão dos dois analisadores específicos de idiomas, `RussianAnalyzer` e `GermanAnalyzer`, A seção 4.8.2 eo wrapper analisador especial por campo, `PerFieldAnalyzerWrapper`, A seção 4.4.

Tabela 4.2 Analisadores primárias disponíveis no Lucene

Analisador	Medidas tomadas
<code>WhitespaceAnalyzer</code>	Divide em tokens whitespace
<code>SimpleAnalyzer</code>	Divide o texto em caracteres nonletter e minúsculas
<code>StopAnalyzer</code>	Divide o texto em caracteres nonletter, minúsculas, e remove palavras de parada
<code>StandardAnalyzer</code>	Tokeniza com base em uma gramática sofisticada, que reconhece e-mail endereços, os acrônimos, chinês-japonês-coreano caracteres, alfanuméricos e muito mais; minúsculas; e remove palavras de parada

A built-in analisadores discutimos nesta seção-`WhitespaceAnalyzer`,`SimpleAnalyzer`,`StopAnalyzer`E `StandardAnalyzer`São projetados para trabalhar com texto em quase todos os idiomas ocidentais (European-based). Você pode ver o efeito de cada um estes analisadores na saída do ponto 4.2.3. `WhitespaceAnalyzer` e `SimpleAnalyzer` são triviais e não cobri-los com mais detalhes aqui. Nós exploramos o `StopAnalyzer` e `StandardAnalyzer` com mais profundidade, porque eles têm não efeitos trivial.

### 4.3.1 StopAnalyzer

`StopAnalyzer`, Além de fazer a divisão de texto básico e lowercasing, também remove stop palavras. Embutidas em `StopAnalyzer` é uma lista de palavras comuns parar Inglês; essa lista é usada, a menos que especificado de outra forma:

```
String public static final [] = {ENGLISH_STOP_WORDS
    "A", "an", "and", "são", "como", "at", "ser", "mas", "by",
    "For", "se", "in", "em", "é", "ele",
```

```
"Não", "não", "de", "on", "ou", "s", "such",
"T", "que", "a", "seu", "então", "lá", "estes",
"Eles", "isto", "a", "era", "irá", "com"
};


```

O `StopAnalyzer` tem um segundo construtor que permite que você passe a sua própria lista como um `String []` em seu lugar. De nota são dois itens na lista padrão: "S" e "T". Contrações são comumente usados em Inglês, como não, não pode, e ele é. Antes de remoção de palavras de parada, o `StopAnalyzer` mantém personagens sucessivos juntos, divisão em caracteres nonletter incluindo o apóstrofo e deixando o se t personagens como tokens autônomos, já que esses personagens são sem sentido em suas própria, faz sentido para removê-los.

Stop palavra-remoção traz à tona uma outra questão interessante: O que aconteceu com os buracos deixados pelas palavras removido? Suponha que você index "um não é suficiente". Os tokens emitidos a partir de `StopAnalyzer` será um e suficiente, Com é e não jogado fora. `StopAnalyzer` atualmente não faz contabilidade de palavras removidas, de modo o resultado é exatamente como se você indexados "um basta". Se você fosse usar Consulta-Analisador juntamente com `StopAnalyzer`, Este documento seria corresponder consultas frase para "Um basta", "um é suficiente", "um, mas não o suficiente", eo original "é uma não o suficiente ". Lembre-se, `QueryParser` também analisa as frases, e cada um desses reduz-se a "um basta" e coincide com os termos indexados. Há um "buraco" muito mais a este tópico, que cobrimos no ponto 4.7.3 (depois de fornecer mais detalhes sobre as posições token).

Tendo as palavras stop removido apresenta uma questão interessante semântica. Você perde algum significado em potencial? A resposta a esta pergunta é: "É depende. "Depende do seu uso do Lucene e se pesquisar sobre essas palavras é significativo para a sua aplicação. Brevemente rever esta um pouco rhetori-questão cal depois, na seção 4.7.3. Para enfatizar e reiterar um importante ponto, apenas os tokens emitidos pelo analisador (ou indexado como `Field.Keyword`) estão disponíveis para pesquisa.

### 4.3.2 StandardAnalyzer

`StandardAnalyzer` detém a honra como o analisador geralmente mais úteis built-in. A JavaCC-based<sup>2</sup> subjaz gramática que, tokenizing com inteligência para o seguinte tipos lexical: alfanuméricos, siglas, nomes de empresas, endereços de correio electrónico, computer nomes de host, números, palavras com apóstrofe interior, números de série, IP endereços, e CJK (Chinês Japonês Coreano) caracteres. `StandardAnalyzer` também

---

<sup>2</sup> Java Compiler-Compiler (JavaCC) é um sofisticado analisador lexical. Ver <http://javacc.dev.java.net>.

incluir stop palavra-remoção, usando o mesmo mecanismo como o `StopAnalyzer` (Lista padrão idêntico Inglês, e um opcional `String []` construtor para substituir). `StandardAnalyzer` faz uma ótima escolha em primeiro lugar.

Utilização `StandardAnalyzer` não é diferente do que usando qualquer um dos outros analisadores,

como você pode ver a partir de seu uso na seção 4.1.1 e `AnalyzerDemo` (Listagem 4.2). Sua efeito único, porém, é aparente no tratamento de texto diferente. Por exemplo, olhar para listagem 4.1, e comparar os diferentes analisadores na frase "XY e Z Corporation - xyz@example.com ". `StandardAnalyzer` é o único que manteve XY e Z juntos, bem como a xyz@example.com endereço de email; ambos os mostrar o processo de análise muito mais sofisticados.

## Lidar com 4,4 campos de palavra-chave

---

É fácil de índice utilizando uma palavra-chave `Field.Keyword`, Que é um único token adicionada a um campo que ultrapassa tokenization e está indexada exatamente como é como um único termo.

É também simples de consulta para um prazo através `TermQuery`. Um dilema pode surgem, no entanto, se você expor `QueryParser` aos usuários e tentativa de consulta em `Field.Keyword` criados campos. A "palavra-chave"-ness de um campo só é conhecido durante a indexação. Não há nada especial sobre campos de palavras-chave, uma vez que está indexada; eles são apenas termos.

Vamos ver a questão exposta com um caso de teste simples que os índices de um doc que este documento com um campo de palavra-chave e, em seguida, tenta encontrar esse documento novamente:

```
public class KeywordAnalyzerTest estende TestCase {
    Diretório RAMDirectory;
    privada IndexSearcher pesquisador;

    setUp public void () throws Exception {
        diretório = new RAMDirectory ();
        Escritor IndexWriter IndexWriter = new (diretório,
            nova SimpleAnalyzer (),
            true);

        Documento doc = new Document ();
        Campo não
        analisado
        doc.add (Field.Keyword ("partNum", "P36"));
        doc.add (Field.Text ("descrição", "Space Illidium Modulator"));
        writer.addDocument (doc);

        writer.Close ();

        searcher = new IndexSearcher (diretório);
    }

    testTermQuery public void () throws Exception {
```

```

Query = nova TermQuery (novo Termo ("partNum", "P36"));
Hits hits = searcher.search (query);
assertEquals (1, hits.length ());
}
}

```

Nenhuma análise aqui

Documento encontrado como esperado

Tão longe, tão boa-nós indexados um documento e pode recuperá-lo usando um `TermQuery`. Mas o que acontece se gerar uma consulta utilizando `QueryParser`?

```

testBasicQueryParser public void () throws Exception {
    Query = QueryParser.parse ("partNum: Q36 e espaço",
                               "Descrição",
                               nova SimpleAnalyzer ());
}

Hits hits = searcher.search (query);
assertEquals ("nota Q36 -> q",
             "+ PartNum: q + espaço", query.toString ("description"));
assertEquals ("doc não encontrado:", 0, hits.length ());
}

```

b    QueryParser analisa cada prazo e frase

c    toString () método

- b `QueryParser` analisa cada termo e expressão da expressão de consulta. Ambos Q36 e SPACE são analisados separadamente. `SimpleAnalyzer` tira de personagens nonletter e minúsculas, de modo Q36 torna-se q. Mas no momento da indexação, Q36 foi deixado como está.
- c Observe, também, que este é o mesmo analisador utilizado durante a indexação.

Consulta tem um bom `toString ()` método (ver secção 3.5.1) para retornar a consulta como um `QueryParser`-Como expressão. Note-se que Q36 se foi.

Esta edição da `QueryParser` analisar um campo de palavra-chave enfatiza um ponto-chave: index-

ing e análise estão intimamente ligadas à pesquisa. O `testBasicQueryParser` mostra teste que a procura por termos criado usando `Field.Keyword` quando uma expressão de consulta é analisadas pode ser problemático. É problemático porque `QueryParser` Analisamos a `partNum` campo, mas não deve ter. Há algumas soluções possíveis para este tipo de dilema:

- Separar a interface do usuário de tal forma que um usuário seleciona um número de peça separadamente de forma livre consultas. Geralmente, os usuários não querem saber (e não deve precisar de saber) sobre os nomes dos campos no índice.
- Explorar o uso do campo de análise específica.
- Se os números parte ou outras construções textuais são lexical comum ocorrências no texto que você está analisando, considere criar um domínio personalizado-analisador específico que reconhece números de peças, e assim por diante, e deixa-los como é.
- Subclasse `QueryParser` e substituir uma ou ambas as `getFieldQuery` métodos de ods para fornecer campo específico de manipulação.

Projetar uma interface de usuário de pesquisa é muito dependente da aplicação; `BooleanQuery` (Seção 3.4.4) e filtros (seção 5.5) fornecer o suporte necessário para combinar peças de consulta de forma sofisticada. Seção 8.5 aborda maneiras de usar JavaScript em um browser web para consultas edifício. As informações contidas neste capítulo fornece as base para a construção de domínio centrada analisadores. Nós vamos aprofundar usando campo específico de análise para o restante desta seção. Nós cobrimos subclasse `QueryParser` na seção 6.3, no entanto, não há vantagem em fazê-lo neste cenário sobre o `PerFieldAnalyzerWrapper` solução que apresentamos aqui.

Um `IndexWriter` trata apenas de uma escolha analisador em uma instância per-ou per-Documento base. Internamente, porém, analisadores pode agir sobre o nome do campo a ser analisados. Os analisadores built-in não aproveitar essa capacidade, porque eles são projetado para uso geral, independentemente do nome do campo. Quando você está confrontados com uma situação que requer uma análise única para diferentes campos, uma opção é o `PerFieldAnalyzerWrapper`.

Nós desenvolvemos um `KeywordAnalyzer` tokeniza todo o fluxo como um único token, imitando como `Field.Keyword` é tratada durante a indexação. Nós só queremos um campo para ser "analisado" dessa maneira, por isso, alavancar o `PerFieldAnalyzer` Invólucro para aplicá-lo apenas para o `partNum` de campo. Primeiro vamos olhar para o `KeywordAnalyzer` na ação em que fixa a situação:

```
public void testPerFieldAnalyzer () throws Exception {
    Analisador PerFieldAnalyzerWrapper PerFieldAnalyzerWrapper = new (
        nova SimpleAnalyzer ());
    analyzer.addAnalyzer ("partNum", new KeywordAnalyzer ());

    Query = QueryParser.parse ("partNum: Q36 e espaço",
        "Descrição",
        analisador);

    Hits hits = searcher.search (query);
    assertEquals ("Q36 mantido como está",
        "+ PartNum: Q36 + espaço", query.toString ("description"));
    assertEquals (hits.length "doc encontrado!", 1, ());
}
```

b      Aplicar  
          KeywordAnalyzer  
          apenas para partNum

é encontrado

- b Nós aplicamos o `KeywordAnalyzer` apenas para o `partNum` campo, e usamos o `SimpleAnalíador` para todos os outros campos. Este é o mesmo resultado eficaz quanto durante a indexação.
- c Observe que a consulta tem agora o termo apropriado para o `partNum` campo, e os doc que este documento é encontrado como esperado.

A built-in `PerFieldAnalyzerWrapper` construtor requer o analisador padrão como um parâmetro. Para atribuir um analisador de diferente para um campo, use a `addAnalyzer`

método. Durante tokenization, o analisador específico para o nome do campo é usado, o padrão é usado se não analisador de campo específico foi atribuído.

Os detalhes do `KeywordAnalyzer` ilustrar buffering personagem. Listagem 4,4 mostra a implementação analisador inteiro.

#### Listagem 4,4 KeywordAnalyzer: emulando Field.Keyword

```

    / **
     * "Tokeniza" todo o fluxo como um símbolo único.
     */
    public class KeywordAnalyzer estende Analyzer {
        pública TokenStream tokenStream (String fieldName,
                                         final Reader leitor) {
            retorno TokenStream new () {
                private boolean feito;
                private final char [] buffer = new char [1024];
                pública Token next () throws IOException {
                    if (! feito) {
                        done = true;
                        StringBuffer buffer = new StringBuffer ();
                        int length = 0;
                        while (true) {
                            = comprimento reader.Read (this.buffer);
                            if (tamanho == -1) break;

                                buffer.append (this.buffer, 0, comprimento);
                        }
                        String texto buffer.toString = ();
                        return new Token (texto, 0, texto.length ());
                    }
                    return null;
                }
            };
        }
    }
}

```

Dado `KeywordAnalyzer`, Poderíamos agilizar o nosso código (em `KeywordAnalyzer-Test.setUp`) E usar o mesmo `PerFieldAnalyzerWrapper` usado em `testPerField-Analisador` durante a indexação. Usando um `KeywordAnalyzer` em campos especiais durante indexação seria eliminar o uso de `Field.Keyword` durante a indexação e substituir com `Field.Text`. Esteticamente, pode ser agradável de ver o mesmo analisador utilizado durante a indexação e consulta, e usando `PerFieldAnalyzerWrapper` faz isso possível.

#### 4.4.1 analisador palavra-chave Alternate

Tome nota das `TokenStream` infra-estrutura (figura 4.2 e Tabela 4.1). A mais simples analisador de palavra-chave é possível se você tiver certeza de suas palavras-chave são 255 caracteres ou menos. Subclassing `CharTokenizer` e dizendo que cada personagem é um token caracter dá esta implementação muito mais limpa:

```
public class SimpleKeywordAnalyzer estende Analyzer {

    pública TokenStream tokenStream (String fieldName,
                                     Reader leitor) {
        return new CharTokenizer (leitor) {
            protegidos boolean isTokenChar (char c) {
                return true;
            }
        };
    }

}
```

No nosso exemplo, poderíamos substituir `KeywordAnalyzer` com `SimpleKeyword-Analisador` desde que os números são definitivamente nossa parte menos de 255 caracteres. Você certamente não querem usuário enterable campos a serem qualquer lugar perto de 255 caracteres de comprimento!

#### 4.5 "Soa como" consulta

Alguma vez você já jogou o game Charades, colocando a mão ao ouvido para indicam que seus gestos próximo referem-se às palavras que "soa como" as palavras real você está tentando transmitir? Não há para nós. Suponhamos, porém, que um high-paying cliente lhe pediu para implementar um motor de busca acessível por J2ME-enabled dispositivos, tais como um telefone celular, para ajudar durante as partidas charada difícil. Em Nesta seção, vamos implementar um analisador para converter palavras para uma raiz fonética usando uma implementação do algoritmo Metaphone do Jakarta Commons projeto Codec. Nós escolhemos o algoritmo Metaphone como um exemplo, mas outros algoritmos estão disponíveis, como o Soundex.

Sendo os caras test-driven estamos, começamos com um teste para ilustrar a alta objetivo nível de nossa experiência de pesquisa:

```
testKoolKat public void () throws Exception {
    Diretório RAMDirectory = new RAMDirectory ();
    Analisador analisador = new MetaphoneReplacementAnalyzer ();
```

```
Escritor IndexWriter IndexWriter = new (analisador, diretório, true);
```

```

Documento doc = new Document ();
doc.add (Field.Text ("conteúdo", "Cat cool"));
writer.addDocument (doc);
writer.Close ();

IndexSearcher searcher = new IndexSearcher (diretório);
Query = QueryParser.parse ("kool kat",
    Usuário digitou
    "Conteúdo",
    analisador);
Hits hits = searcher.search (query);
assertEquals (1, hits.length ());
assertEquals ("gato cool", hits.doc (0).get ("conteúdo"));
searcher.close ();
}

```

Original documento

Usuário digitou

Consulta hip jogos!

Consulta Hip jogos!

Valor original ainda disponíveis

Parece mágica! O usuário busca para "kool kat". Nenhum desses termos estavam em nosso documento original, ainda que a pesquisa encontrou o jogo desejado.

**Pesquisas**

no texto original seria também retornar as partidas que o esperado. O truque está sob  
O MetaphoneReplacementAnalyzer:

```

public class MetaphoneReplacementAnalyzer estende Analyzer {
    pública TokenStream tokenStream (String fieldName, Reader leitor) {
        retorno MetaphoneReplacementFilter novo (
            nova LetterTokenizer (leitor));
    }
}

```

Porque o algoritmo Metaphone espera palavras que incluem apenas as letras, o LetterTokenizer é usado para alimentar nosso filtro metaphone. O LetterTokenizer não minúsculas, no entanto. Os tokens emitidos são substituído por seus metaphone equivalente, de modo lowercasing é desnecessário. Vamos agora escavar o MetaphoneReplacement-

Filtro, Onde o trabalho real é feito:

```

public class MetaphoneReplacementFilter estende TokenFilter {
    metaphone String public static final = "metaphone";

    privada Metaphone metaphoner = new Metaphone ();
    MetaphoneReplacementFilter público (entrada TokenStream) {
        super (entrada);
    }

    pública Token next () throws IOException {
        Token t = input.next ();
    }
}

pública Token next () throws IOException {
    Token t = input.next ();
}

```

org.apache.commons  
.Codec.language.  
Metaphone

Puxe próxima  
símbolo

```

    if (t == null) return null;           ← Quando nulo final,
                                         foi alcançado

    try {
        return new Token (metaphoner.encode (t.termText ()),
                           t.startOffset (),
                           t.endOffset (),
                           Metaphone); Definir o tipo de token
    } Catch (EncoderException e) {
        / / Se não é possível codificar, basta devolver token original
        retorno t;
    }

}
}

```

Converter token para Metaphone codificação; deixar cargo info como é

O token emitido pelos nossos `MetaphoneReplacementFilter`, Como o próprio nome indica, litros aliado substitui o token de entrada (a menos que por algum motivo, a codificação falhou, e o original é emitido). Este novo token é definido com os deslocamentos mesma posição que o original, porque é uma substituição na mesma posição. O último argumento para `o Símbolo` indica o construtor tipo de token. Cada token pode ser associado a um `Corda` indicando seu tipo, dando meta-dados para posterior filtragem na análise processo. O `StandardTokenizer`, Conforme discutido em "utilidade tipo Token" sob seção 4.2.3, tags tokens com um tipo que é usado mais tarde pelo `StandardFilter`. O `Metaphone` tipo não é usado em nossos exemplos, mas demonstra que um filtro mais tarde poderia ser Metaphone-token ciente chamando `Símbolo'S tipo ()` método.

**NOTA** Símbolo tipos, tais como a `Metaphone` tipos utilizados em `MetaphoneReplacement-Analisador`, São realizadas através da fase de análise, mas não são codificados para o índice. Salvo disposição em contrário, digite o `palavra` é usado para tokens por padrão. Seção 4.2.3 discute tipos de token mais.

Como sempre, é bom para ver o que é um analisador está fazendo com o texto. Usando nosso `Analyzer-Utils`, Duas frases que soam semelhantes ainda estão escritas de forma completamente diferente são

**tokenized e exibida:**

```

public static void (String [] args) throws IOException {principal
    MetaphoneReplacementAnalyzer analisador =
        nova MetaphoneReplacementAnalyzer ();
    AnalyzerUtils.displayTokens (analisador,
        "A ligeira raposa marrom saltou sobre o cão preguiçoso");

    System.out.println ("");
    AnalyzerUtils.displayTokens (analisador,
        "Tha quik marrom phox jumpd ovvar tha Lazi dogz");
}

```

Temos uma amostra do codificador Metaphone, mostrado aqui:

```
[0] [KK] [BRN] [FKS] [JMPT] [RLO] [0] [LS] [TKS]
[0] [KK] [BRN] [FKS] [JMPT] [RLO] [0] [LS] [TKS]
```

Wow-uma correspondência exata!

Na prática, é improvável que você quiser que sons semelhantes, exceto em partidas especiais lugares, caso contrário, muitos jogos indesejada pode ser returned.<sup>3</sup> Na

"O que o Google faça?" Sentido, uma característica soa-como seria ótimo para situações de ções em que um usuário com erros ortográficos de cada palavra e nenhum documento foi encontrado, mas

palavras alternativas poderia ser sugerido. Uma abordagem de implementação para esta idéia poderia ser para executar todo o texto através de uma análise sons semelhantes e construir uma referência cruzada

de pesquisa para consultar quando uma correção é necessária.

## 4,6 Sinônimos, aliases, e palavras que significam a mesma coisa

Nosso analisador próximo personalizado injeta sinônimos das palavras para o token de saída stream, mas coloca os sinônimos na mesma posição como a palavra original. Por adding sinônimos durante a indexação, você faz buscas encontrar documentos que não podem contêm os termos de pesquisa original, mas combinar os sinônimos dessas palavras. Teste primeiro, claro:

```
testJumps public void () throws Exception {
    Token [] tokens =
        AnalyzerUtils.tokensFromAnalysis (synonymAnalyzer4, "Saltos");
    AnalyzerUtils.assertTokensEqual (tokens,
        new String [] {"saltos", "Hops", "saltos"});
    // Assegura que sinônimos são na mesma posição que o original
    assertEquals ("pula", 1, tokens [0].getPositionIncrement ());
    assertEquals ("hops", 0, tokens [1].getPositionIncrement ());
    assertEquals ("saltos", 0, tokens [2].getPositionIncrement ());
}
```

Analisar uma palavra

Três palavras  
sair

<sup>3</sup> Enquanto trabalhava neste capítulo, Erik pediu a seu filho de 5 anos de idade, brilhante, Jakob, como ele significaria legal gato. Jakob respondeu: "c-o-l c-a-t". O que é um idioma Inglês maravilhosamente confuso é. Erik imagina que um "soa-como" recurso em motores de busca concebido para crianças seria muito útil. Metaphone en-códigos cool, kool, e col todos como KL.

A construção de `SynonymAnalyzer` é mostrado brevemente.

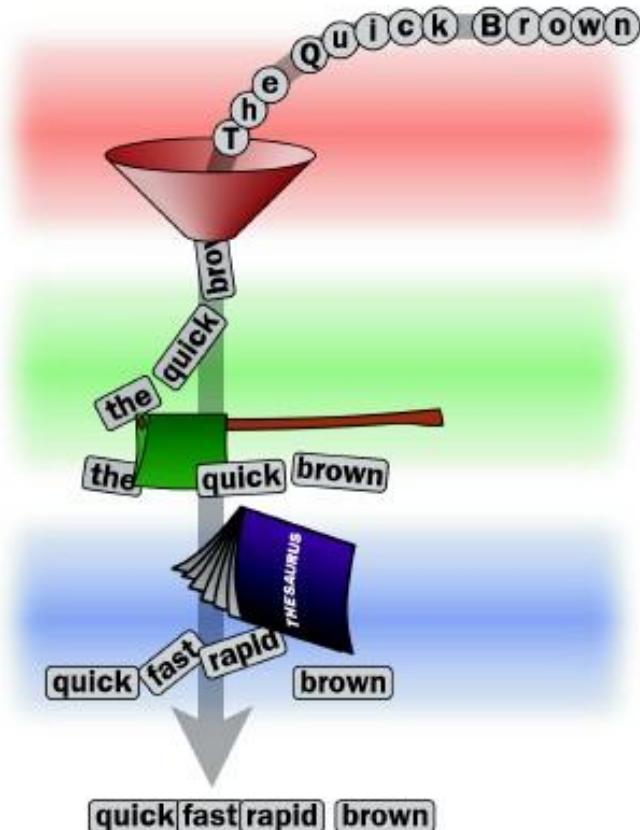


Figura 4.4  
SynonymAnalyzer  
visualizado como fábrica  
automação

Observe que o nosso teste de unidade mostra não só que sinônimos da palavra saltos são emitido a partir da `SynonymAnalyzer` mas também que os sinônimos são colocados na mesma posição (incremento de zero) como a palavra original.

Vamos ver o que o `SynonymAnalyzer` está fazendo, em seguida, vamos explorar as implicações

incrementos de posição. Figura 4.4 mostra graficamente o que os nossos `SynonymAnalyzer` faz a entrada de texto e de listagem 4.5 é a implementação.

#### Listagem 4.5 SynonymAnalyzer implementação

```
public class SynonymAnalyzer estende Analyzer {
    motor SynonymEngine privado;

    pública SynonymAnalyzer (SynonymEngine motor) {
        this.engine = motor;
    }

    pública TokenStream tokenStream (String fieldName, Reader leitor) {
        Resultado TokenStream = nova SynonymFilter (
```

```

        nova StopFilter (
            nova LowerCaseFilter (
                nova StandardFilter (
                    nova StandardTokenizer (leitor))),
                StandardAnalyzer.STOP_WORDS),
            motor
        );

        resultado de retorno;
    }
}

```

Mais uma vez, o código do analisador é mímina e apenas um cadeias `Tokenizer` juntamente com uma série de `TokenFilter`'s, na verdade, este é o `StandardAnalyzer` envolvido com um filtro adicional. (Ver tabela 4.1 para saber mais sobre esses analisador básico blocos de construção.) O final `TokenFilter` na cadeia é o novo `SynonymFilter` (Listagem 4.6), que chega ao coração da discussão atual. Quando você está termos injetáveis, buffer é necessário. Este filtro usa um `Pilha` como o buffer.

#### Listagem 4.6 SynonymFilter: buffering tokens e emitindo um de cada vez

```

public class SynonymFilter estende TokenFilter {
    TOKEN TYPE SÝNONÝM String public static final = "SINÔNIMO";

    synonymStack Stack privado;
    motor SynonymEngine privado;

    pública SynonymFilter (TokenStream in, SynonymEngine motor) {
        super (in);
        synonymStack = new Stack ();
        this.engine = motor;
    }

    pública Token next () throws IOException {
        if (synonymStack.size ()> 0) {
            retorno (Token) synonymStack.pop ();
        }
        Token token = input.next ();
        if (token == null) {
            return null;
        }
        addAliasesToStack (token);
        retorno token;
    }
}

```

Sinônimo  
amortecedor

Pop buffered  
sinônimos

C    Leia próximo token

d    Push sinônimos de token atual na pilha

e    Retorno token atual

```

addAliasesToStack private void (Token token) throws IOException {
    String [] sinônimos = engine.getSynonyms (token.termText ());
    f          Recuperar sinônimos

    if (sinônimos == null) return;

    for (int i = 0; i < synonyms.length; i++) {
        Token synToken = new Token (sinônimos [i],
            token.startOffset (),
            token.endOffset (),
            TOKEN_TYPE_SYNONYM);
        g          Definir a
        posiçãosynToken.setPositionIncrement (0);
        h          h      incremento
                    a zero
        synonymStack.push (synToken);
    }
}

```

g Push sinônimos na pilha

- b O código sucessivamente aparece a pilha de sinônimos buffered dos últimos transmitido em token até que ele é vazio.
- c Depois de todos os sinônimos token anterior ter sido emitido, lemos o próximo token.
- d Nós empurramos todos os sinônimos do token atual na pilha.
- e Agora vamos retornar o atual (e original) antes de token seus sinônimos associados.
- f Sinônimos são obtidos a partir do `SynonymEngine`.
- g Nós empurramos cada sinônimo para a pilha.
- h O incremento de posição é definido para zero, permitindo sinônimos para ser virtualmente na mesmo lugar que o termo original.

O projeto de `SynonymAnalyzer` permite pluggable `SynonymEngine` implementações. `SynonymEngine` é uma interface de um método:

```

public interface SynonymEngine {
    String [] getSynonyms (String s) throws IOException;
}

```

Usando uma interface para este projeto permite facilmente mock-objeto implementações para testes purposes.<sup>5</sup> Nós deixá-lo como um exercício para que você crie qualidade de produção `SynonymEngine` implementations.<sup>6</sup> Para nossos exemplos, nós usamos um mock simples que é codificados com uma sinônimos alguns:

<sup>5</sup> Se os objetos mock são novos para você, consulte o "sobre este livro", secção no início do livro para um descrição e referências você pode consultar para mais informações.

<sup>6</sup> É cruel para deixá-lo pendurado com uma execução simulada, não é? Na verdade, temos implementado um poderoso `SynonymEngine` usando o banco de dados WordNet. É tratados na secção 8.6.2.

```

public class MockSynonymEngine implementa {SynonymEngine
    private static HashMap map = new HashMap ();

    static {
        map.put ("quick", new String [] {"fast", "speedy"});
        map.put ("saltos", new String [] {"saltos", "hops"});
        map.put ("over", new String [] {"acima"});
        map.put ("preguiçoso", new String [] {"apáticos", "lento"});
        map.put ("cães", new String [] {"caninos", "pooches"});
    }

    public String [] getSynonyms (String s) {
        retorno (String []) map.get (s);
    }
}

```

Os sinônimos gerados por `MockSynonymEngine` são uma forma: Por exemplo, rápido tem os sinônimos rápido e speedy, mas rápido não tem sinônimos. Este é, por definição, um objeto fictício usado para testes em um ambiente controlado, de modo que não precisa se preocupar com a natureza de uma forma de sua implementação.

Alavancar o incremento posição parece poderosa, e de fato é. Você só deve modificar incrementos sabendo de alguns casos estranhos que surgem na pesquisa ing, no entanto. Desde sinônimos são indexados como outros termos, `TermQuery` funciona como esperado. Além disso, `PhraseQuery` funciona como esperado quando usamos um sinônimo no lugar de uma palavra original. O `SynonymAnalyzerTest` caso de teste na listagem de 4,7 demonstra as coisas funcionando bem, usando API-criado consultas.

#### Listagem 4,7 SynonymAnalyzerTest: mostrando que as consultas sinônimo de trabalho

```

public class SynonymAnalyzerTest estende TestCase {
    RAMDirectory diretório privado;
    privada IndexSearcher pesquisador;
    private static SynonymAnalyzer synonymAnalyzer =
        nova SynonymAnalyzer (novo MockSynonymEngine ());

    setUp public void () throws Exception {
        diretório = new RAMDirectory ();

        Escritor IndexWriter IndexWriter = new (diretório,
            synonymAnalyzer,
            true);
        b
        Analisar com
        Documento doc = new Document ();
        SynonymAnalyzer
        doc.add (Field.Text ("conteúdo",
            "O ligeira raposa marrom saltou sobre o cão preguiçoso "));
        writer.addDocument (doc); Único índice
        writer.Close (); documento
        searcher = new IndexSearcher (diretório);
    }
}

```

```

}

tearDown public void () throws Exception {
    searcher.close ();
}

testSearchByAPI public void () throws Exception {

    TermQuery tq = nova TermQuery (novo Termo ("conteúdo", "hops"));
    Hits hits = searcher.search (tq);
    assertEquals (1, hits.length ());

    PhraseQuery pq = PhraseQuery new ();
    pq.add (novo prazo ("conteúdo", "raposa"));
    pq.add (Termo de novo ("conteúdo", "hops"));
    hits = searcher.search (pq);
    assertEquals (1, hits.length ());

}
}

```

c Procurar "Saltos"

d Procurar "Raposa saltos"

- b Realizamos a análise com um costume `SynonymAnalyzer`, Usando `MockSynonymMotor`.
- c Uma busca pela palavra lúpulo corresponde ao documento.
- c A busca pela frase "raposa saltos" também corresponde.
- d

A frase "... raposa saltos ..." foi indexada, e nossa `SynonymAnalyzer` injetado lúpulo na mesma posição como saltos. A `TermQuery` para lúpulo conseguiu, como fez uma exata `PhraseQuery` para "fox lúpulo". Excelente!

Tudo está bem, até que decide usar `QueryParser` para criar consultas, em vez de fazê-lo diretamente com a API. Mais uma vez, um teste aponta a estranheza explicitamente:

```

testWithQueryParser public void () throws Exception {
    Query = QueryParser.parse ("\\" saltos fox \ """",
                                "Conteúdo",
                                synonymAnalyzer);

    Hits hits = searcher.search (query);
    ? assertEquals ("!!!! o que ", 0 hits.length (), 0);

    query = QueryParser.parse ("\\" raposa saltos \ """",
                                "Conteúdo",
                                nova StandardAnalyzer ());
    hits = searcher.search (query);
    assertEquals ("* ufa *", uma hits.length (), 0);
}

```

Analisador não consegue encontrar documento usando frase do documento original

StandardAnalyzer ainda documento encontra

A primeira parte `testWithQueryParser` usa o `SynonymAnalyzer` também para analisar a seqüência de consulta em si. Estranhamente, a consulta não corresponde, mesmo usando o mesmo analisador

utilizada para a indexação. Mas, se usarmos o `StandardAnalyzer` (Lembrar que `Sinônimo Analisador` tem o mesmo núcleo, exceto para injetar os sinônimos), o esperado correspondência for encontrada. Por que isso? Um dos primeiros passos de diagnóstico recomendado quando

utilização `QueryParser` é para despejar os `toString ()` valor do Pergunta exemplo:

```
public static void main (String [] args) throws Exception {
    Query = QueryParser.parse ("\" saltos fox \\"",
        "Conteúdo",
        synonymAnalyzer);

    System.out.println ("\" saltos fox \\" analisa a" +
        query.toString ("conteúdo"));

    System.out.println ("De AnalyzerUtils.tokensFromAnalysis:");
    AnalyzerUtils.displayTokens (synonymAnalyzer,
        "\" Saltos fox \\"");
}
```

Aqui está a saída:

```
"Saltos raposa" analisa a "raposa saltos saltos saltos"

De AnalyzerUtils.tokensFromAnalysis:
[Fox] [salta] [hops] [salta]
```

`QueryParser` funciona de forma semelhante ao nosso `AnalyzerUtils.tokensFromAnalysis`, Significando colas que todos os termos da análise em conjunto para formar um `PhraseQuery` e ignora token informações incremento posição. A busca por "saltos fox" não funciona usando `QueryParser` eo `SynonymAnalyzer` porque internamente a consulta é para o frase "saltos saltos saltos da raposa". Por ter um processo de análise um pouco diferente para `QueryParser` do que para a indexação, o problema está resolvido. Não há necessidade de injetar sinônimos ao consultar qualquer maneira, desde que o índice já contém os sinônimos.

Você tem outra opção com sinônimos: ampliá-las em cada consulta ao invés de indexação. Nós não implementar essa abordagem, mas as técnicas e as ferramentas fornecidas neste capítulo seria essencial para implementá-lo efetivamente. O chamado desajeitadamente `PhrasePrefixQuery` (Ver secção 5.2) é uma opção de consider, talvez criado através de uma substituído `QueryParser.getFieldQuery` método, esta é uma opção possível para explorar se você deseja implementar sinônimo injeção no momento da consulta.

#### 4.6.1 Visualizando posições token

Nosso `AnalyzerUtils.tokensFromAnalysis` não nos mostram todas as informações quando se lida com analisadores que definir incrementos posição diferente de 1. Em ordem para obter uma melhor visualização destes tipos de analisadores, nós adicionamos um utilitário adicional método, `displayTokensWithPositions`, Para `AnalyzerUtils`:

```

public static void displayTokensWithPositions
    (Analizador analisador, String texto) throws IOException {
    Token [] tokens = tokensFromAnalysis (texto analisador);

    int posição = 0;

    for (int i = 0; i < tokens.length; i + +) {
        Token token = tokens [i];

        int = incremento token.getPositionIncrement ();

        if (incremento > 0) {
            posição = posição + incremento;
            System.out.println ();
            System.out.print (posição + ":");

        }

        System.out.println ("[" + token.termText () + "]");
    }
}

```

Nós escrevemos um pedaço rápido de código para ver o que os nossos `SynonymAnalyzer` está realmente fazendo:

```

SynonymAnalyzerViewer public class {
    public static void (String [] args) throws IOException {principal
        AnalyzerUtils.displayTokensWithPositions (
            nova SynonymAnalyzer (novo MockSynonymEngine ()),
            "A ligeira raposa marrom ataca o cão preguiçoso");
    }
}

```

E agora podemos visualizar os sinônimos colocados nas mesmas posições que o original palavras:

```

1: [Rápida] [rápida] [fast]
2: [Marrom]
3: [Fox]
4: [Salta] [hops] [salta]
5: [Mais] [acima]
6: [Preguiça] [lento] [apático]
7: [Cães] [pooches] [caninos]

```

Cada número à esquerda representa a posição token. Os números aqui são contínuo, mas eles não seria se o analisador deixou buracos (como você verá com o analisador personalizada ao lado). Vários termos mostrado para uma única posição ilustra onde foram acrescentados sinônimos.

## 4.7 análise Stemming

---

Nosso analisador final retira todos os batentes. Ele tem um nome ridículo, ainda descritivo: `PositionalPorterStopAnalyzer`. Este analisador remove palavras de parada, deixando possíveis buracos internacionais onde as palavras são removidas, e também utiliza um filtro decorrente.

O `PorterStemFilter` é mostrado na figura 4.3, mas não é usado por qualquer built-in analisador. Ele caulea palavras usando o algoritmo de Porter decorrente criado por Dr. Martinus Porter, e é melhor definida em suas próprias palavras:

O algoritmo de Porter decorrentes (*'Porter stemmer'*) é um processo para a remoção das terminações morfológicas e lexeiras flexionais das palavras em Inglês.  
Seu principal uso é como parte de um processo de normalização termo que normalmente é feito quando  
criação de Recuperação de Informação systems.<sup>7</sup>

Em outras palavras, as várias formas de uma palavra são reduzidos a uma forma raiz comum. Por exemplo, as palavras respira, respira, respiração, e respirava, através da Porter stemmer, reduzir a respiração.

O lematizador Porter é um dos muitos algoritmos resultantes. Consulte a seção 8.3.1, página 283, para a colheratura de uma extensão do Lucene que implementa o Snowball algoritmo (também criado por Dr. Porter). KStem é outro algoritmo decorrente que foi adaptado para Lucene (pesquisa no Google por KStem e Lucene).

### 4.7.1 Deixando buracos

Lacunas são deixadas em que palavras de parada são removidos, ajustando o incremento posição dos tokens (ver também "Olhando para dentro tokens" na seção 4.2.3). Isto é ilustrado a partir da saída de `AnalyzerUtils.displayTokensWithPositions`:

```
2: [Rápida]
3: [Marron]
4: [Fox]
5: [Salto]
6: [Mais]
8: [Lazi]
9: [Cão]
```

Posições 1 e 7 estão em falta devido à remoção de da. Stop palavra-remoção que lacunas folhas é realizada usando-se um costume `PositionalStopFilter`:

```
public class PositionalStopFilter estende TokenFilter {
    privada stopwords Set;
```

<sup>7</sup> A partir do site Dr. Porter: <http://www.tartarus.org/~martin/PorterStemmer/index.html>.

```

pública PositionalStopFilter (TokenStream em, Set stopwords) {
    super (in);
    this.stopWords = stopwords;
}

public final Token next () throws IOException {
    int incremento = 0;
    for (Token token = input.next ();
        token != null; token = input.next ()) {

        if (! stopWords.contains (token.termText ())) {
            token.setPositionIncrement (
                token.getPositionIncrement () incremento +);
            retorno token;
        }

        incremento + +;
    }

    return null;
}
}

```

← Deixe espaço para parar ignorada palavras

O analisador, `PositionalPorterStopAnalyzer` (Mostrado na listagem 4.8), fornece a lista de palavras de parada para remover.

#### 4.7.2 Colocando juntos

Este analisador personalizado usa nosso filtro de remoção personalizada stop-palavra, que é alimentado a partir `umLowerCaseTokenizer`. Os resultados do filtro de parada são alimentados à Porter-tronco mer. Listagem 4.8 mostra a implementação completa deste analisador sofisticado. `LowerCaseTokenizer` começa o processo de análise, alimentando tokens através dos nossos filtro de remoção personalizada stop-palavra e, finalmente, decorrentes das palavras usando o built-em Porter stemmer.

Listagem 4.8 `PositionalPorterStopAnalyzer`: remove palavras de parada (deixando lacunas e caules palavras)

```

public class PositionalPorterStopAnalyzer estende Analyzer {
    privada stopwords Set;

    pública PositionalPorterStopAnalyzer () {
        este (StopAnalyzer.ENGLISH_STOP_WORDS);
    }

    pública PositionalPorterStopAnalyzer (String [] lista de palavras
        irrelevantes) {
        stopwords = StopFilter.makeStopSet (lista de palavras irrelevantes);
    }
}

```

```

pública TokenStream tokenStream (String fieldName, Reader leitor) {
    return new PorterStemFilter (
        nova PositionalStopFilter (
            , nova LowerCaseTokenizer (leitor)
            stopwords)
        );
}
}

```

Deixando lacunas quando palavras de parada são removidos faz sentido lógico, mas introduz novas questões que vamos explorar a seguir.

#### 4.7.3 monte de problemas Hole

Como você viu com o `SynonymAnalyzer`, Mexer com informações sobre a posição token pode causar problemas durante a pesquisa. `PhraseQuery` e `QueryParser` são os dois encenqueiros. Jogos frase exata agora falhar, como ilustrado no nosso caso de teste:

```

public class PositionalPorterStopAnalyzerTest estende TestCase {
    private static PositionalPorterStopAnalyzer porterAnalyzer =
        nova PositionalPorterStopAnalyzer ();

    RAMDirectory diretório privado;

    setUp public void () throws Exception {
        diretório = new RAMDirectory ();
        Escritor IndexWriter =
            IndexWriter nova (diretório, porterAnalyzer, true);

        Documento doc = new Document ();
        doc.add (Field.Text ("conteúdo",
            "A ligeira raposa marrom ataca o cão preguiçoso"));
        writer.addDocument (doc);
        writer.Close ();

    }

    testExactPhrase public void () throws Exception {
        IndexSearcher searcher = new IndexSearcher (diretório);
        Query = QueryParser.parse ("\\" sobre o preguiçoso \\ """",
            "Conteúdo",
            porterAnalyzer);

        Hits hits = searcher.search (query);
        assertEquals ("correspondência exata não encontrado!", 0, hits.length
        ());
    }
}

```

Como mostrado, uma consulta frase exata não foram encontrados. Isso é perturbador, é claro. Ao contrário da situação analisador sinônimo, usando um analisador diferentes não vai resolver o problema. A dificuldade reside no interior mais profundo `PhraseQuery` e sua incapacidade atual para fazer face às lacunas posicionais. Todos os termos de uma `PhraseQuery` devem estar lado a lado, e em nosso caso de teste, a frase é procurando é "over Lazi" (stop palavra removido com palavras restantes stemmed).

`PhraseQuery` não permite uma folga pequena, chamada slop. Isto é coberto em maior detalhes na seção 3.4.5, no entanto, seria indelicado deixar sem mostrar um consulta frase de trabalho. Definir o slop a 1 permite a consulta para a ignorar a lacuna:

```
testWithSlop public void () throws Exception {
    IndexSearcher searcher = new IndexSearcher (diretório);

    Parser QueryParser = new ("conteúdo",
                               porterAnalyzer);
    parser.setPhraseSlop (1);

    Query = parser.parse ("\\" sobre o preguiçoso \\ "");

    Hits hits = searcher.search (query);
    assertEquals ("buraco contabilizados", 1, hits.length ());
}
```

O valor do fator de slop frase, em uma definição simplificada para este caso, representa quantas palavras de parada pode estar presente no texto original entre indexados palavras. Introdução de um factor slop maior que zero, no entanto, permite ainda mais inexata frases para corresponder. Neste exemplo, em busca de "mais lenta" também corresponde.

Com parada palavra-remoção em análise, fazendo exato correspondências de frase é, por definição,

não é possível: As palavras removidos não estão lá, então você não pode saber o que eram.

O fator de slop aborda o problema principal com a pesquisa usando palavras-stop remoção que deixa buracos, você pode agora ver a beneficiar nossos analisador fornece, graças ao decorrentes:

```
testStems public void () throws Exception {
    IndexSearcher searcher = new IndexSearcher (diretório);
    Query = QueryParser.parse ("preguiça",
                               "Conteúdo",
                               porterAnalyzer);
    Hits hits = searcher.search (query);
    assertEquals ("Lazi", 1, hits.length ());

    query = QueryParser.parse ("\\" fox saltou\\ """",
                               "Conteúdo",
                               porterAnalyzer);
```

```
    hits = searcher.search (query);
    assertEquals ("salto saltos saltou jumping", 1, hits.length ());
}
```

Ambos preguiça ea frase "raposa saltou" matched nosso documento indexado, permitindo que os usuários um pouco de flexibilidade nas palavras utilizadas durante a pesquisa.

## 4.8 análise de questões de Língua

---

Lidando com línguas em Lucene é uma questão interessante e multifacetado. Como pode texto em várias línguas ser indexados e recuperados posteriormente? Como um desen-  
oper construção I18N-friendly aplicações em todo Lucene, quais as questões que você  
necessidade de considerar?

Você deve lidar com várias questões ao analisar o texto em várias lan-  
guas. O primeiro obstáculo é garantir que conjunto de caracteres de codificação é feito  
corretamente

tais que os dados externos, como arquivos, são lidos em Java corretamente. Durante o  
processo de análise, línguas diferentes têm conjuntos diferentes de palavras de parada e único  
decorrentes algoritmos. Talvez acentos devem ser removidas a partir de personagens como  
bem, o que seria dependentes do idioma. Finalmente, você pode exigir linguagem  
detecção se você não tem certeza de que língua está sendo usada. Cada uma dessas questões é  
em última análise, até que o desenvolvedor endereço, com apenas suporte de blocos de  
construção básicos

fornecidas pelo Lucene. No entanto, uma série de analisadores e construções adicionais  
blocos, como `Tokenizers` e `TokenStreams` estão disponíveis no Sandbox (dis-  
discutidos em seção 8.3) e em outros lugares online.

Esta seção discute Lucene embutido no manuseio para não-Inglês línguas,  
mas começamos pela primeira vez com uma breve introdução ao Unicode e codificação de  
caracteres.

### 4.8.1 Unicode e codificações

Internamente, Lucene armazena todos os personagens na codificação UTF-8 padrão. Java  
nos liberta de muitas lutas pela manipulação de Unicode automaticamente dentro `Cordas`  
e fornecimento de instalações para a leitura de dados externos nas codificações muitos. Você,  
no entanto, são responsáveis pela obtenção de texto externo em Java e Lucene. Se você está  
arquivos em um sistema de indexação de arquivos, você precisa saber o que codificar os  
arquivos foram

salvos como, a fim de lê-los corretamente. Se você está lendo HTML ou XML a partir de um  
HTTP servidor, problemas de codificação de ficar um pouco mais complexa. Codificações podem ser  
especifi-

cados em um cabeçalho HTTP do tipo de conteúdo ou especificado no próprio documento em  
o cabeçalho do XML ou um HTML `<meta>` tag.

Nós não vamos elaborar sobre esses detalhes de codificação, não porque eles não são impor-  
tante, mas porque são questões separadas Lucene. Consulte o apêndice C

para várias fontes de informações mais detalhadas sobre temas de codificação. Em particular, se você é novo para as questões I18N, leia o excelente artigo de Joel Spolsky, "O Absoluto Todo Software Developer mínimo absolutamente, positivamente deve saber sobre a Unicode e Conjuntos de Caracteres (Sem Desculpas!) "(<http://www.joelonsoftware.com/articles/Unicode.html>) e a linguagem Java Internacionalização tutorial (<http://java.sun.com/docs/books/tutorial/i18n/intro/>). Além disso, a próxima versão do Java (codinome Tiger) transições para Unicode 4.0 suporte para caracteres suplementares.

Vamos prosseguir com o pressuposto de que você tem o seu texto disponível como Unicode, e passar para as preocupações específicas da linguagem Lucene-.

#### 4.8.2 Análise não-Inglês línguas

Todos os detalhes do processo de análise se aplicam quando você está lidando com um texto não-

Idiomas Inglês. Extração de termos do texto é o objetivo. Com a Western linguagens, onde os espaços em branco e pontuação são usados para separar as palavras, você deve

ajustar parar de listas de palavras e algoritmos decorrentes de ser específico para a linguagem de o texto está sendo analisado.

Além do built-in analisadores nós discutimos, distribuição Lucene o núcleo fornece dois analisadores específicos do idioma: GermanAnalyzer e RussianAnalyzer. Ambos estes empregam idioma específico decorrentes e parar palavra de remoção. Também livremente disponível é o SnowballAnalyzer família de derivações, que suporta vários idiomas europeus. Discutimos SnowballAnalyzer na secção 8.3.1.

O GermanAnalyzer começa com uma StandardTokenizer e StandardFilter (Como o StandardAnalyzer) E, em seguida, alimenta o fluxo através de um StopFilter e umGermanStemFilter. A built-in conjunto comum de palavras de parada alemã é usado por padrão, você pode substituí-lo usando o mecanismo discutido na secção 4.3.1. O GermanStemFilter hastas com base em palavras de língua alemã regras e também provides um mecanismo para fornecer um conjunto de exclusão de palavras que não devem ser stemmed (que está vazio por padrão).

O RussianAnalyzer começa com uma RussianLetterTokenizer, Que suporta vários conjuntos de caracteres Unicode e como CP1251 e minúsculas em um forma de conjunto de caracteres específicos usando um RussianLowerCaseFilter. O StopFilter remove stop palavras usando um conjunto padrão de palavras russas, mas também permite que você forneça uma conjunto personalizado. Por fim, o RussianStemFilter hastas palavras usando o Snowball algo-seado em regras (ver secção 8.3.1 para mais detalhes).

### 4.8.3 Analisando idiomas asiáticos

Idiomas asiáticos, como o chinês, japonês e coreano (também indicado como CJK), geralmente usam ideogramas ao invés de um alfabeto para representar palavras. Esses-pictorial palavras podem ou não podem ser separados por espaços em branco e, portanto, exigem uma di-

tipo diferente de análise que reconhece quando tokens devem ser divididos. A única built-in analisador capaz de fazer algo útil com texto asiático é o `StandardAnalyzer`, Que reconhece alguns intervalos do espaço como caracteres Unicode CJK e tokeniza-los individualmente.

No entanto, dois analisadores no Sandbox Lucene são adequados para idiomas asiáticos análise (ver seção 8.1 para mais detalhes sobre o Sandbox). Em nossos dados livro da amostra, os caracteres chineses para o livro Tao Te Ching foram adicionados ao título. Porque nossos dados se origina em arquivos de propriedades Java, seqüências de escape Unicode são usados: 8

```
title = Tao Te Ching \ u9053 \ u5FB7 \ u7D93
```

Usamos `StandardAnalyzer` para todos os campos tokenized em nosso índice, que tokeniza cada palavra Inglês como esperado (tao, te, e ching) assim como cada um dos chineses personagens como termos separados (tao Te Ching) mesmo que não há espaço entre -los. Nosso `ChineseTest` demonstra que a procura pela palavra tao usando seu Chinese representação funciona como desejado:

```
public class ChineseTest estende LiaTestCase {
    testChinese public void () throws Exception {
        IndexSearcher searcher = new IndexSearcher (diretório);
        Hits hits = searcher.search (
            nova TermQuery (novo Termo ("conteúdo", "道")));
        assertEquals ("tao", 1, hits.length ());
    }
}
```

Note-se que o nosso arquivo `ChineseTest.java` foi salvo em formato UTF-8 e compilados utilizando o interruptor de codificação `UTF8` para o compilador `javac`. Tivemos que garantir que a representações dos caracteres chineses são codificados e ler corretamente, e usar um analisador de CJK-aware.

Semelhante ao `AnalyzerDemo` na listagem 4.2, criamos um `ChineseDemo` (Listagem 4.9) programa para ilustrar como analisadores de vários trabalhar com texto em chinês. Esta demo usa AWT Rótulos para exibir corretamente os caracteres, independentemente de sua localidade e console ambiente.

---

<sup>8</sup> `java.util.Properties` arquivos de cargas propriedades usando a codificação ISO-8859-1, mas permite que personagens de ser codificados usando o padrão Java Unicode. Java inclui um `native2ascii` programa que pode converter arquivos codificados em nativamente o formato apropriado.

## Listagem 4,9 ChineseDemo: ilustra o que fazer com analisadores de texto em chinês

```

public class ChineseDemo {
    String private static [] strings = {"道德经"};
    Analyzer private static [] = {analisadores
        nova SimpleAnalyzer (),
        nova StandardAnalyzer (),
        nova ChineseAnalyzer (), Analisadores de
        nova CJKAnalyzer () Sandbox
    };

    public static void main (String args []) throws Exception {

        for (int i = 0; i < strings.length; i++) {
            String string = strings [i];
            for (int j = 0; j < analyzers.length; j++) {
                Analisador analisador analisadores = [j];
                analisar (string, analisador);
            }
        }
    }

    static void privada analisar (String String, Analyzer analisador)
        throws IOException {
        StringBuffer buffer = new StringBuffer ();
        Token [] tokens =
            AnalyzerUtils.tokensFromAnalysis (analisador, string);
        for (int i = 0; i < tokens.length; i++) {Recuperar tokens
            buffer.append ("[");da análise usando
            buffer.append (tokens [i] termText ());
            buffer.append ("]");
        }
    }

    Saída de String buffer.toString = ();

    Frame Frame f = new ();
    String nome = analyzer.getClass () getName ();
    f.setTitle nome.substring ((name.lastIndexOf ('.') + 1)
        + ":" + String);
    f.setResizable (false);

    Font font = novo Font (null, Font.PLAIN, 36);
    int width = getWidth (f.getFontMetrics (fonte), output);

    f.setFontSize ((largura <250) 250: largura + 50, 75);

    Label Label = new Label (buffer.toString ());
    label.setSize (largura, 75);
    label.setAlignment (Label.CENTER);
    label.setFont (fonte);
    f.add (label);
}

```

Textos chinês para ser analisados

Recuperar tokens da análise usando AnalyzerUtils

Label exibe AWT análise

```

        f.setVisible (true);
    }

    private static int getWidth (FontMetrics métricas, String s) {
        int size = 0;
        for (int i = 0; i <s.length (); i + +) {
            tamanho + = metrics.charWidth (s.charAt (i));
        }

        retorno de tamanho;
    }
}

```

CJKAnalyzer e ChineseAnalyzer analisadores são encontrados no Sandbox Lucene; eles não são incluídos na distribuição Lucene core. ChineseDemo mostra o ou colocar usando um componente Label AWT para evitar qualquer confusão que possa surgir a partir de console codificação de saída ou limitada fontes mangling coisas, você pode ver os ou colocar na figura 4.5.

The screenshot shows an IDE interface with the title bar "lia.ipr - [/Users/erik/dev/LuceneInAction] - /Users/erik/dev/LuceneInAction/Manuscript/Analysis/src/lia/analysis/i18n/ChineseDe". The menu bar includes Edit, Search, View, Go To, Code, Refactor, Build, Run, Tools, CVS, Window, Help. The toolbar has icons for file operations like Open, Save, Print, etc. Below the toolbar is a tab bar with tabs for SpanQuery.java, TestBasics.java, SpanQueryTest.java, ChineseDemo.java (which is the active tab), SpanFirstQuery.java, and SpanNearQuery.java. The code editor displays the following Java code:

```

public class ChineseDemo {

    private static String[] strings = {"道德經"};

    private static Analyzer[] analyzers = {
        new SimpleAnalyzer(),
        new StandardAnalyzer(),
        new ChineseAnalyzer(),
        new CJKAnalyzer()
    };
}

```

Overlaid on the code editor are four small windows, each containing a different analysis result for the string "道德經":

- SimpleAnalyzer:** 道德經
- StandardAnalyzer:** 道 [德] 經
- ChineseAnalyzer:** [道] [德] 經
- CJKAnalyzer:** [道德] [德經]

Figura 4.5 ChineseDemo ilustrando a análise do título Tao Te Ching

O CJKAnalyzer pares de caracteres em janelas sobrepostas de dois caracteres cada. Muitas palavras CJK são dois personagens. Ao emparelhar caracteres desta forma, as palavras é provável que sejam mantidos juntos (assim como personagens desconexos, aumentando o tamanho do índice). O ChineseAnalyzer assume uma abordagem mais simples e, no nosso exemplo, espelhos os resultados do built-in StandardAnalyzer por tokenizing cada Chinese personagem. Palavras que consistem de múltiplos caracteres chineses são divididos em termos para cada personagem componente.

#### 4.8.4 Zaijian<sup>9</sup>

Um grande obstáculo (não relacionados com Lucene) permanece quando você está lidando com vários

idiomas: manipulação de codificação de texto. O StandardAnalyzer ainda melhor o built-in analisador de propósito geral, mesmo tendo em conta caracteres CJK, no entanto, o Sandbox CJKAnalyzer parece mais adequado para a análise de línguas asiáticas.

Quando você está indexação de documentos em vários idiomas em um único índice, usando um porDocumento analisador é apropriado. Você também pode querer adicionar um campo a

documentos indicando a sua língua; este campo pode ser usado para filtrar os resultados da pesquisa

ou para fins de exibição durante a recuperação. Em "locale Controlar data de análise" em seção 3.5.5, que mostram como recuperar a localidade a partir do browser do usuário web, o que pode ser automaticamente usado em consultas.

Um tópico final é a detecção de idioma. Isto, como codificações de caracteres, está fora lado âmbito do Lucene, mas pode ser importante para a sua aplicação. Nós não cobrir linguagem de detecção de técnicas neste livro, mas é uma área activa de pesquisa com várias implementações para escolher (ver anexo C).

### 4.9 análise Nutch

---

Nós não temos o código fonte do Google, mas temos o projeto open-source Nutch, criado pela Corte do Lucene Doug criador. Nosso estudo de caso na seção Nutch 10,1 discute os detalhes da arquitetura Nutch. Há um outro interessante faceta Nutch: como ele analisa o texto. Nutch faz algo muito interessante, com palavras de parada, que chama termos comuns. Se todas as palavras estão indexados, um enorme

número de documentos tornar-se associado a cada termo comum, como da. Consultando o é praticamente uma consulta sem sentido, dado que a maioria dos documents contenham esse termo. Quando os termos comuns são usados em uma consulta, mas não dentro uma frase, como o marrom rápida sem outros adornos ou citações, comum

---

<sup>9</sup> Zaijian meios adeus em chinês.

termos são descartados. No entanto, se uma série de termos é cercada por aspas duplas, como "o marrom rápido", um apreciador truque é jogado, o que detalhamos nesta seção.

Nutch combina uma análise do índice de tempo bigram (Agrupamento dois anos consecutivos palavras como um token único) com uma técnica de otimização de consultas em tempo de frases. Isso resulta em um espaço muito menor documento considerado durante a busca, por exemplo, os documentos têm muito menos o rápido lado a lado, que contêm da. Utilização as partes internas do Nutch, criamos um exemplo simples para demonstrar a Nutch trickery análise. Listagem 4,10 primeiro analisa a frase "O marrom rápida ..." usando o `NutchDocumentAnalyzer` e em seguida, analisa uma consulta de "o marrom rápido" para demonstrar a consulta Lucene criado.

Listagem 4,10 NutchExample: demonstrando a análise Nutch e consulta para análise de técnicas

```
public class {NutchExample
    public static void (String [] args) throws IOException {principal
        NutchDocumentAnalyzer analisador = new NutchDocumentAnalyzer ();
        displayTokensWithDetail (analisador, "A ligeira raposa marrom ...");
    }
}
```

**a** Analisador de costume

**b** método displayTokensWithDetail

**c** Use nomes de classe totalmente qualificado

**d** Traduzir Nutch Query

**e**

- b** Nutch usa um analisador de costume, `NutchDocumentAnalyzer`.
- c** `displayTokensWithDetail` é semelhante ao nosso anterior `AnalyzerUtils` métodos, exceto Nutch exige o nome do campo conteúdo. Então, criamos uma versão personalizada one-off de este utilitário para inspecionar Nutch.
- d** Nutch confrontos com alguns dos nomes Lucene de classe, nomes de classe totalmente qualificado para
- e** são necessárias. O `net.nutch.searcher.Query` classe não está relacionado com Lucene Pergunta classe.
- f** A Nutch Pergunta é traduzido em um Lucene Pergunta exemplo.

A saída do analisador mostra como "o rápido" torna-se um bigram, mas a palavra o não é descartada. O bigram reside na mesma posição como token a:

```
1: [o: <word>] [a-rápido: gram]
2: [rápida: <word>]
```

```
3: [brown: <word>]
4: [raposa: <word>]
```

Porque tokens adicionais são criados durante a análise, o índice é maior, mas o benefício deste trade-off é que procura por frase exata-consultas são muito mais rápidos. E há um bônus: Nenhum termo foram descartados durante a indexação.

Durante a consulta, as frases são também analisados e otimizados. A saída da consulta (Lembre-se do ponto 3.5.1 que `Pergunta'S toString ()` é útil) do Lucene `Pergunta` instância para a expressão de consulta "O marrom rápido" é

```
query = (+ url: "o marrom rápido" ^ 4.0)
        => (+ Anchor: "o marrom rápido" ^ 2.0) (+ de conteúdo: "o marrom-rápido rápido")
```

Uma consulta Nutch expande para pesquisa na `url` e âncora `campos`, bem como, com maior aumento para esses campos, usando a frase exata. O `conteúdo` cláusula de campo é optimizada a incluir apenas os bigram de uma posição que contém um adicional `<word>` token tipo.

Esta foi uma visão rápida do que Nutch faz com análise de indexação e consulta construção. Nutch continua a evoluir, otimizar e ajustar os vários técnicas de indexação e consulta. Os bi-gramas não são levados em consideração exceto na `conteúdo` campo, mas como o documento base cresce, se otimização deções são necessárias em outros campos serão reavaliados.

## Resumo 4.10

---

Análise, enquanto que apenas uma faceta única de usar o Lucene, é o aspecto que merece o mais atenção e esforço. As palavras que podem ser pesquisados são aqueles emitidos during análise de indexação. Claro, usando `StandardAnalyzer` pode fazer o truque para a sua necessidades, e é suficiente para muitas aplicações. No entanto, é importante sub-suportar o processo de análise. Usuários que fazem análise para concessão muitas vezes correr em

confusão mais tarde, quando eles tentam entender por que procura "ser ou não ser" não retornou resultados (talvez devido a palavra-stop remoção).

Demora menos de uma linha de código para incorporar um analisador durante a indexação. Muitos processos sofisticados podem ocorrer debaixo das cobertas, como parar de palavra remoção e decorrentes de palavras. Removendo palavras diminui o seu tamanho índice, mas pode ter um impacto negativo sobre a precisão consulta.

Porque um tamanho não cabe tudo quando se trata de análise, você pode precisar ajustar o processo de análise para o seu domínio de aplicação. Analisador elegante do Lucene arquitetura separa cada um dos processos internos para a análise textual, deixando você reutilizar blocos de construção fundamentais para a construção de analisadores personalizados. Quando

você está trabalhando com analisadores, certifique-se de usar o nosso `AnalyzerUtils`, Ou algo similar, para ver em primeira mão como o texto é indexado. Se você está mudando analisadores, você deve recriar o índice usando o novo analisador de modo que todos os documentos são analisados da mesma maneira.

# Avançado técnicas de pesquisa

Este capítulo aborda

- Classificando resultados de
- Pesquisa Consultas abrangendo
- Filtragem
- Pesquisar índice de múltiplas e remotas
- Aproveitando vetores prazo

Muitas aplicações que implementam busca com Lucene pode fazer isso usando a API introduzido no capítulo 3. Alguns projetos, porém, precisa mais do que o básico pesquisar mecanismos. Neste capítulo, vamos explorar o mais sofisticado busca recursos incorporados no Lucene.

Um casal de probabilidades e extremidades, `PhrasePrefixQuery` e `MultiFieldQueryParser`, completar a nossa cobertura adicional de Lucene capacidades embutidas. Se você Lucene utilizado por um tempo, você pode não reconhecer algumas dessas características. Classificação, consultas span, e vetores de termos são todos novos na Lucene 1.4, aumentando dramaticamente Lucene poder e flexibilidade.

## 5.1 Classificando resultados de pesquisa

Até Lucene 1.4, os resultados da pesquisa só foram devolvidos em ordem decrescente de pontuação, com os documentos mais relevantes aparecendo primeiro. BookScene, O nosso hipotético livraria, precisa de exibir os resultados agrupados em categorias, e dentro a categoria de resultados os livros devem ser ordenados por relevância para a consulta. Cola seleção de todos os resultados e classificá-los programaticamente fora do Lucene é um maneira de conseguir isso, no entanto, isso introduz um desempenho possível gargalo se o número de resultados é enorme. Felizmente, programador especializado Tim Jones contribuiu com um aumento altamente eficiente para Lucene, acrescentando sofisticadas capacidades de classificação para resultados de busca. Nesta seção, vamos explorar a vari-ous maneiras para classificar os resultados da pesquisa, incluindo a separação de um ou mais valores de campo em ascendente ou descendente.

### 5.1.1 Usando uma espécie

`IndexSearcher` contém vários sobreescarregado pesquisa métodos. Até agora temos cov-Ered apenas o básico pesquisa (`Query`) método, que retorna os resultados ordenados por decrescente de relevância. A versão de classificação deste método tem a assinatura de busca (`Query Ordenar`). Listagem 5,1 demonstra o uso da classificação pesquisa método. O `displayHits` método utiliza a classificação pesquisa método e exibe o `Hits`. Os seguintes exemplos irão utilizar o `displayHits` método para ilustrar como vários tipos de trabalho.

#### Listagem 5,1 exemplo classificação

```
public class {SortingExample  
    Directory diretório privado;  
  
    pública SortingExample (diretório Diretório) {  
        this.directório = diretório;  
    }  
}
```

```

pública displayHits void (Query, tipo Sort) Ordenar objeto encapsula
    classificação infothrows IOException {
        IndexSearcher searcher = new IndexSearcher (diretório);

        Hits hits = searcher.search (espécie de consulta); C Busca sobrecarregado
        System.out.println ("\ nresults para:" + c método
            query.toString () + "ordenados por" sorte +);
            d saída toString

        System.out.println (StringUtils.rightPad ("Título", 30) +
            StringUtils.rightPad ("pubmonth", 10) +
            StringUtils.center ("id", 4) +
            StringUtils.center ("score", 15));

        DecimalFormat scoreFormatter = new DecimalFormat ("0 .#####");
        for (int i = 0; i <hits.length (); i + +) {
            Documento doc = hits.doc (i); e StringUtils fornece
            System.out.println (
                StringUtils.rightPad
                    StringUtils.abbreviate (doc.get ("title"), 29), 30) +
                StringUtils.rightPad (doc.get ("pubmonth"), 10) +
                StringUtils.center (" " + hits.id (i), 4) +
                StringUtils.leftPad (
                    scoreFormatter.format (hits.score (i)), 12));
                System.out.println (" " + doc.get ("categoria"));
                System.out.println (searcher.explain (consulta, hits.id (i)));
            }
            f Explicação comentou
            / / fora por enquanto
            searcher.close ();
        }
    }

```

- b** Ordenar o objeto encapsula uma coleção ordenada de informações de classificação de campo.
- c** Que chamamos de sobrecarga pesquisa método com o Espécie objeto.
- d** O Espécie classe tem informativa `toString ()` saída.
- e** Usamos `StringUtils` de Jakarta Commons Lang para boa saída para colunar-matting.
- f** Mais tarde você vai ver uma razão para olhar para a explicação de pontuação. Por agora, é comentadas para fora.

Uma vez que nossa amostra conjunto de dados consiste em apenas um punhado de documentos, a classificação exemplos usam uma consulta que retorna todos os documentos:

```

Termo mais antigo = novo Termo ("pubmonth", "190001");
Último termo Term = new ("pubmonth", "201012");
AllBooks RangeQuery RangeQuery = new (o mais antigo mais recente, true);

```

Todos os livros em nossa coleção são nesta faixa mês de publicação. Em seguida, a exemplo runner soas é construído com base no caminho índice fornecido como uma propriedade do sistema:

```

String indexDir = System.getProperty ("index.dir");

Diretório FSDirectory =
    FSDirectory.getDirectory (indexDir, false);
Exemplo SortingExample = new SortingExample (diretório);

```

Agora que você já viu como usar a classificação, vamos explorar várias formas de pesquisa resultados podem ser classificadas.

### 5.1.2 A classificação por relevância

Tipos Lucene por relevância decrescente, também chamado partitura por padrão. Ordenar por relevância pontuação funciona por qualquer passagem nulo como o Espécie objeto ou usando o padrão Espécie comportamento. Cada uma das variantes a seguir retorna os resultados no padrão ordem de pontuação. `example.displayHits (allBooks, Sort.RELEVANCE)` é um atalho para o uso Ordenar new ():

```

example.displayHits (allBooks, Sort.RELEVANCE);
example.displayHits (allBooks, Sort.new ());

```

Há sobrecarga envolvida no uso de uma Espécie objeto, porém, de modo a manter a usar pesquisa (Query) ou de busca (Query, null) se você deseja classificar por relevância. O out-posto de usar `Sort.RELEVANCE` é a seguinte (note a coluna pontuação decrescente):

```

Resultados para: pubmonth: [190001 TO 201012] classificadas por <score>, <doc>
Titlepubmonthidscore
A Arte Moderna de Education19810600.086743
    / Educação / pedagogia
Segredos de Saúde Imperial ... 19940110.086743
    / Saúde / alternativas / chinês
Tao Te Ching 道德 經19881020.086743
    / Filosofia / Leste
Gödel, Escher, Bach: um Et ... 19790330.086743
    / Tecnologia / computadores / ai
Mindstorms19800140.086743
    / Tecnologia / programação de computadores / / educação
Java Development com Ant20020850.086743
    / Tecnologia / computadores / programação
JUnit em Action20031060.086743
    / Tecnologia / computadores / programação
Lucene em Action20040670.086743
    / Tecnologia / computadores / programação
Tapeçaria em Action20040390.086743
    / Tecnologia / computadores / programação
Extreme Programming Explained 19991080.062685
    / Tecnologia / computadores / programação / metodologia
O Programmer199910100.062685 Pragmatic
    / Tecnologia / computadores / programação

```

A saída de `Espécie's toString ()` mostra `<score>, <doc>`. Pontuação e ordem do índice são tipos especiais de classificação: Os resultados são retornados primeiro na diminuição da pontuação

ordem e, quando as notas são idênticas, por subsorted ID documento crescente da ordem. Fim ID documento é a ordem em que os documentos foram indexados. Em nosso caso, a ordem do índice não é relevante, ea ordem é não especificado (ver secção 8.4 na a formiga <index> tarefa, que é como nós indexamos os nossos dados da amostra).

Como um aparte, você pode se perguntar por que a pontuação dos dois últimos livros é diferente

do resto. Nossa consulta estava em um intervalo de datas de publicação. Ambos os livros tem o mês mesma publicação. A `RangeQuery` se expande, debaixo das cobertas, em `umBooleanQuery` combinando qualquer um dos termos no intervalo. O documento freqüência do termo 199910 no `pubmonth` campo é 2, o que reduz o inverso documento de freqüência factor (IDF) para os documentos, diminuindo assim o pontuação. Tivemos a mesma curiosidade ao desenvolver este exemplo, e pouco comum execução do Explicação saída em `displayHits` deu-nos os detalhes para substand este efeito. Consulte a seção 3.3. para mais informações sobre os fatores de pontuação.

### 5.1.3 Ordenação por ordem do índice

Se a ordem documentos foram indexados é relevante, você pode usar `Sort.INDEXORDER`. Nota da coluna ID aumentando documento:

```
example.displayHits (allBooks, Sort.INDEXORDER);

Resultados para: pubmonth: [190001 TO 201012] classificadas por <doc>
Titlepubmonthidscore
A Arte Moderna de Education19810600.086743
    / Educação / pedagogia
Segredos de Saúde Imperial ... 19940110.086743
    / Saúde / alternativas / chinês
Tao Te Ching 道德 经19881020.086743
    / Filosofia / Leste
Gödel, Escher, Bach: um Et ... 19790330.086743
    / Tecnologia / computadores / ai
Mindstorms19800140.086743
    / Tecnologia / programação de computadores / / educação
Java Development com Ant20020850.086743
    / Tecnologia / computadores / programação
JUnit em Action20031060.086743
    / Tecnologia / computadores / programação
Lucene em Action20040670.086743
    / Tecnologia / computadores / programação
Extreme Programming Explained 19991080.062685
    / Tecnologia / computadores / programação / metodologia
Tapeçaria em Action20040390.086743
    / Tecnologia / computadores / programação
O Programmer199910100.062685 Pragmatic
    / Tecnologia / computadores / programação
```

Até agora nós apenas classificadas por pontuação, que já estava acontecendo sem usar o classificação das instalações e ordem do documento, que é provavelmente apenas marginalmente útil em melhores. Ordenar por um de nossos próprios campos é realmente o que estamos procurando.

#### 5.1.4 Ordenação por um campo

Ordenar por um campo primeiro requer que você siga as regras para a indexação de um sortable campo, conforme detalhado na seção 2.6. Nosso categoria campo foi indexado como um único Field.Keyword por documento, permitindo que seja usado para classificação. Para classificar por um campo, você deve criar um novo Espécie objeto, fornecendo o nome do campo:

```
example.displayHits (allBooks, nova espécie ("categoria"));
```

```
Resultados para: pubmonth: [190001 TO 201012] classificadas por "categoria", <doc>
Titlepubmonthidscore
A Arte Moderna de Education19810600.086743
    / Educação / pedagogia
Segredos de Saúde Imperial ... 19940110.086743
    / Saúde / alternativas / chinês
Tao Te Ching 道德 經19881020.086743
    / Filosofia / Leste
Gödel, Escher, Bach: um Et ... 19790330.086743
    / Tecnologia / computadores / ai
Java Development com Ant20020850.086743
    / Tecnologia / computadores / programação
JUnit em Action20031060.086743
    / Tecnologia / computadores / programação
Lucene em Action20040670.086743
    / Tecnologia / computadores / programação
Tapeçaria em Action20040390.086743
    / Tecnologia / computadores / programação
O Programmer199910100.062685 Pragmatic
    / Tecnologia / computadores / programação
Mindstorms19800140.086743
    / Tecnologia / programação de computadores / / educação
Extreme Programming Explained 19991080.062685
    / Tecnologia / computadores / programação / metodologia
```

Os resultados aparecem agora ordenados por nossa categoria campo em ordem alfabética crescente da ordem. Observe a saída ordenados por: A Espécie própria classe adiciona automaticamente documento de identificação como o campo de classificação final quando um nome único campo é especificada, de modo que o sec-

#### 5.1.5 Inverso da ordem padrão da categoria é pela ID do documento.

classificação

A direção de classificação padrão para campos de classificação (incluindo relevância e ID do documento) ordenação é natural. Ordem natural é descendente de relevância, mas crescente de

todos os outros campos. A ordem natural pode ser revertida por campo. Por exemplo, aqui que lista os livros com as mais recentes publicações em primeiro lugar:

```
example.displayHits (allBooks, nova espécie ("pubmonth", true));
```

Resultados para: pubmonth: [190001 TO 201012] classificadas por "pubmonth", <doc>

Titlepubmonthidscore

Lucene em Action20040670.086743  
 / Tecnologia / computadores / programação

Tapeçaria em Action20040390.086743  
 / Tecnologia / computadores / programação

JUnit em Action20031060.086743  
 / Tecnologia / computadores / programação

Java Development com Ant20020850.086743  
 / Tecnologia / computadores / programação

Extreme Programming Explained 19991080.062685  
 / Tecnologia / computadores / programação / metodologia

O Programmer199910100.062685 Pragmatic  
 / Tecnologia / computadores / programação

Segredos de Saúde Imperial ... 19940110.086743  
 / Saúde / alternativas / chinês

Tao Te Ching 道德 經19881020.086743  
 / Filosofia / Leste

A Arte Moderna de Education19810600.086743  
 / Educação / pedagogia

Mindstorms19800140.086743  
 / Tecnologia / programação de computadores / / educação

Gödel, Escher, Bach: um Et ... 19790330.086743  
 / Tecnologia / computadores / ai

O ponto de exclamação em `classificadas por "pubmonth"!`, <doc> indica que o `pubmonth` campo está sendo classificados em ordem natural reverso (descendente meses de publicação, mais recente primeiro). Note que os dois livros com o mês mesma publicação são classificados a fim id documento.

### 5.1.6 Ordenação por vários campos

Implicitamente temos ordenar por vários campos, desde a `Espécie` anexa um objeto classificar por ID documento em casos apropriados. Você pode controlar os campos de classificação explicitamente usando uma matriz de `SortFields`. Este exemplo usa como uma categoria principal de alfa-diabéticos tipo, com resultados dentro de categoria, ordenados por pontuação e, finalmente, livros com igual pontuação dentro de uma categoria são classificados por diminuindo mês de publicação:

```
example.displayHits (allBooks,
  Ordenar novo (novo SortField [][]
    nova SortField ("categoria"),
    SortField.FIELD_SCORE,
    nova SortField ("pubmonth", SortField.INT, true)
 )));

```

```

Resultados para: pubmonth: [190001 TO 201012]
    classificadas por "categoria", <score> ", pubmonth"!
Titlepubmonthidscore
A Arte Moderna de Education19810600.086743
    / Educação / pedagogia
Segredos de Saúde Imperial ... 19940110.086743
    / Saúde / alternativas / chinês
Tao Te Ching 道德 经19881020.086743
    / Filosofia / Leste
Gödel, Escher, Bach: um Et ... 19790330.086743
    / Tecnologia / computadores / ai
Lucene em Action20040670.086743
    / Tecnologia / computadores / programação
Tapecaria em Action20040390.086743
    / Tecnologia / computadores / programação
JUnit em Action20031060.086743
    / Tecnologia / computadores / programação
Java Development com Ant20020350.086743
    / Tecnologia / computadores / programação
O Programmer199910100.062685 Pragmatic
    / Tecnologia / computadores / programação
Mindstorms19800140.086743
    / Tecnologia / programação de computadores / / educação
Extreme Programming Explained 19991001080.062685
    / Tecnologia / computadores / programação / metodologia

```

## TECNOLOGIA

O exemplo, mantém internamente um array de `SortFields`, mas só nesta-exame soas você já viu isso explicitamente; os outros exemplos usados atalhos para criar o `SortField array`. A `SortField` mantém o nome do campo, um tipo de campo, e vice-versa bandeira ordem. `SortField` contém constantes para os tipos de campos diversos, incluindo SCORE,

`DOC,AUTO,STRING,INT,E FLOAT,SCORE` e `DOC` são tipos especiais de classificação na relevância e ID documento. `AUTO` é o tipo usado por cada um dos nossos outros exemplos, que classificar por um nome de campo.

O tipo de campo é automaticamente detectado como `Corda,int` ou `flutuar` baseado sobre o valor do primeiro termo no campo. Se você estiver usando cordas que podem aparecerem como numéricos em alguns campos, não se esqueça de especificar o tipo explicitamente como `SortField.STRING`.

### 5.1.7 Selecionando um tipo de campo de classificação

Por tempo de pesquisa, os campos que podem ser classificados em tipos e suas correspondentes são já está definido. Tempo de indexação é quando a decisão sobre recursos de classificação deve ser feita, no entanto, o costume implementações de classificação podem fazê-lo em tempo de pesquisa, como você verá na secção 6.1. Seção 2.6 discute índice de tempo de design de classificação. Pelo índice-  
ing um `Integer.toString` ou `Float.toString`, A classificação pode ser baseada em numérico val-  
ues. Em dados de nosso exemplo, `pubmonth` foi indexada como uma `Corda` mas é um válido,

parsable Número inteiro, Assim ela é tratada como tal para fins de classificação, a menos que especificado como

SortField.STRING explicitamente. Ordenar por um tipo numérico consome menos memória recursos do que por STRING; Seção 5.1.9 discute problemas de desempenho ainda mais.

É importante entender que os valores de índice numérico desta forma a facilitate a classificação desses campos, não para restringir uma busca em um intervalo de valores. O recurso de consulta numérico gama é coberto na seção 6.3.3, o padding-tech nique será necessário durante a indexação e pesquisa a fim de utilizar numéricos campos para a pesquisa. Todos os termos de um índice são Cordas, o recurso de classificação usa o padrão Número inteiro e Flutuar construtores para analisar as representações de string.

### 5.1.8 Usando um local não padrão para classificar

Quando você está em uma classificação SortField.STRING ordem do tipo, é feita pelo sobre o uso String.CompareTo por padrão. No entanto, se você precisa de um colabo-diferentes fim ção, SortField permite que você especifique uma localidade. A Collator é determinada para o locale fornecido usando Collator.getInstance (Locale), E os Collator.compare método determina a ordem de classificação. Há dois sobrecarregado SortField constructors para uso quando você precisa especificar locale:

```
pública SortField (campo String, Locale locale)
pública SortField (campo String, Locale locale, inverter boolean)
```

Ambos os construtores implica a SortField.STRING tipo porque locale se aplica apenas a string do tipo de classificação, para não numéricos.

### 5.1.9 Performance efeito de classificação

Classificação vem à custa de recursos. Mais memória é necessária para manter os campos utilizados para a classificação disponível. Para tipos numéricos, cada campo que está sendo classificada para cada documento no índice exige que quatro bytes armazenados em cache. Para Corda tipos, cada termo também é única em cache para cada documento. Apenas os campos reais utilizados para classificação são armazenadas em cache desta maneira.

Plano de recursos do seu sistema de acordo, se você quiser usar a classificação capacidades, sabendo que a classificação por um Corda é o tipo mais caro em termos de recursos.

## Usando 5,2 PhrasePrefixQuery

---

A built-in PhrasePrefixQuery é definitivamente uma consulta de nicho, mas é potencialmente útil. O nome é um pouco confuso porque esta consulta não é de forma alguma relacionado com PrefixQuery. É, no entanto, estreitamente relacionadas com PhraseQuery.

`PhrasePrefixQuery` permite que vários termos por posição, efetivamente o mesmo que um `BooleanQuery` nonrequired em múltiplas `PhraseQuery` cláusulas. Por exemplo, supose queremos encontrar todos os documentos sobre raposas rápidas, com rápido ou rápido seguido por raposa. Uma abordagem é fazer uma "Fox rápida" ou "fast fox" consulta. Outra opção é usar `PhrasePrefixQuery`. No nosso exemplo, dois documentos são indexados com frases semelhantes. Um documento com usos "A ligeira raposa marrom saltou sobre o cão preguiçoso", e os outros usos "a raposa rápido pulou sobre o cão", como mostrado na nosso teste `setUp ()` método:

```
public class PhrasePrefixQueryTest extends TestCase {
    privada IndexSearcher pesquisador;

    setUp protected void () throws Exception {
        Diretório RAMDirectory = new RAMDirectory ();
        Escritor IndexWriter = new (diretório,
            nova WhitespaceAnalyzer (), true);
        Doc1 documento = new Document ();
        doc1.add (Field.Text ("campo",
            "O ligeira raposa marrom saltou sobre o cão preguiçoso "));
        writer.addDocument (doc1);
        Doc2 documento = new Document ();
        doc2.add (Field.Text ("campo",
            "O rápido fox pulou sobre o cão "));
        writer.addDocument (doc2);
        writer.Close ();

        searcher = new IndexSearcher (diretório);
    }
}
```

Sabendo que queremos encontrar documentos sobre raposas rápida, `PhrasePrefixQuery` permite-nos corresponder frases muito parecido `PhraseQuery`, Mas com uma diferença: cada posição do termo da consulta pode ter vários termos. Isto tem o mesmo conjunto de hits como um `BooleanQuery` consistindo de múltiplos `PhraseQuery`s combinada com uma OR operador. O método de teste a seguir demonstra a mecânica de usar o `PhrasePrefixQuery` API, adicionando um ou mais termos a um `PhrasePrefixQuery` exemplo, em ordem:

```
testBasic public void () throws Exception {
    PhrasePrefixQuery query = new PhrasePrefixQuery ();
    query.add (Termo de novo [] {
        novo prazo ("campo", "rápido"),
        novo prazo ("campo", "fast")
    });
    query.add (Termo de novo ("campo", "raposa"));
}
```

Qualquer um destes termos pode ser  
na primeira posição para corresponder  
Apenas um em cada  
segunda posição

```

        Hits hits = searcher.search (query);
        assertEquals ("match fox rápido", 1, hits.length ());

        query.setSlop (1);
        hits = searcher.search (query);
        assertEquals ("match tanto", 2, hits.length ());
    }
}

```

Assim como com `PhraseQuery`, O fator de slop é suportado. Em `testBasic ()`, O fator é usados para corresponder "quick brown fox" na segunda pesquisa, com o padrão de slop zero, ela não corresponde. Para completar, aqui é um teste que ilustra a descrita `BooleanQuery`, Com um conjunto de slop para a frase "fox rápido":

```

testAgainstOR public void () throws Exception {
    PhraseQuery QuickFox PhraseQuery = new ();
    quickFox.setSlop (1);
    quickFox.add (Termo de novo ("campo", "rápido"));
    quickFox.add (Termo de novo ("campo", "raposa"));

    PhraseQuery FastFox PhraseQuery = new ();
    fastFox.add (Termo de novo ("campo", "fast"));
    fastFox.add (Termo de novo ("campo", "raposa"));

    Query = BooleanQuery BooleanQuery new ();
    query.add (QuickFox, false, false);
    query.add (FastFox, false, false);
    Hits hits = searcher.search (query);
    assertEquals (2, hits.length ());

}

```

Uma diferença entre `PhrasePrefixQuery` eo `BooleanQuery` de Frase PerguntaAbordagem é de que o fator slop é aplicada globalmente com `PhrasePrefixQuery`. É aplicado numa base frase-per com `PhraseQuery`.

Claro, hard-coding os termos não seria realista, de modo geral. Um possível uso de um `PhrasePrefixQuery` seria injetar sinônimos Dynamicamente em posições frase, permitindo a correspondência menos precisos. Por exemplo, você pode amarrar no código WordNet-based (ver secção 8.6 para mais informações sobre WordNet e Lucene).

**NOTA** Lucene QueryParser atualmente não suporta `PhrasePrefixQuery`.

## 5.3 Consultando em vários campos de uma só vez

---

Em dados de nosso livro, vários campos foram indexadas. Os usuários podem querer consulta para termos independentemente de qual campo que estão dentro Uma maneira de lidar com isso é com Com diversos-

QueryParser, Que se baseia QueryParser. Nos bastidores, ele analisa uma consulta expressão usando QueryParser's estática analisar método para cada campo como o padrão campo e os combina em uma BooleanQuery. O operador por defeito ou é usado em o mais simples analisar método ao adicionar as cláusulas para o BooleanQuery. Para mais fino controle, o operador pode ser especificado para cada campo, conforme necessário (REQUIRED\_FIELD), proibida (PROHIBITED\_FIELD) Ou normal (NORMAL\_FIELD), Usando as constantes a partir de MultiFieldQueryParser.

Listagem 5.2 mostra isso mais pesado QueryParser variante em uso. O testDefault-Operador () primeiro método analisa a consulta "Desenvolvimento" usando tanto o título e assuntos campos. O teste mostra que os documentos jogo com base em qualquer uma das esses campos. O segundo teste, testSpecifiedOperator (), Define a análise para man-

data que os documentos devem coincidir com a expressão em todos os campos especificados.

#### Listagem 5.2 MultiFieldQueryParser em ação

```
public class MultiFieldQueryParserTest estende LiaTestCase {
    testDefaultOperator public void () throws Exception {
        Query = MultiFieldQueryParser.parse ("desenvolvimento",
            new String [] {"title", "sujeitos"},
            nova SimpleAnalyzer ());
        IndexSearcher searcher = new IndexSearcher (diretório);
        Hits hits = searcher.search (query);

        assertHitsIncludeTitle (hits, "Desenvolvimento Java com Ant");

        / / Tem "desenvolvimento" no campo de assuntos
        assertHitsIncludeTitle (hits, "Extreme Programming Explained");
    }

    testSpecifiedOperator public void () throws Exception {
        Query = MultiFieldQueryParser.parse ("desenvolvimento",
            new String [] {"title", "sujeitos"},
            new int [] {MultiFieldQueryParser.REQUIRED_FIELD,
                MultiFieldQueryParser.REQUIRED_FIELD},
            nova SimpleAnalyzer ());
        IndexSearcher searcher = new IndexSearcher (diretório);
        Hits hits = searcher.search (query);

        assertHitsIncludeTitle (hits, "Desenvolvimento Java com Ant");
        assertEquals ("um e apenas um", 1, hits.length ());
    }
}
```

`MultiFieldQueryParser` tem algumas limitações devido à forma como ele usa `QueryParser's` estático analisar método. Você não pode controlar qualquer uma das configurações que `QueryParser` sup<sup>porta</sup>s, e você está preso com os padrões, tais como padrão de análise de localidade e data zero slop-frase consultas padrão.

**NOTA** De um modo geral, a consulta em vários campos, não é a melhor prática introduzidos pelo utilizador para consultas. Mais comumente, todas as palavras que você deseja que pesquisou são indexados em um conteúdo ou palavras-chave campo através da combinação de vários campos. A sintético conteúdo campo em nosso ambiente de teste usa essa esquema para colocar autor e assuntos em conjunto:  
`doc.add(Field.Unstored("conteudo", autor + " " + temas));`

Usamos um espaço (" ") Entre o autor e temas para separar as palavras para analisador. Permitindo que os usuários insiram texto da maneira mais simples possível sem a necessidade de qualificar os nomes de campo, geralmente contribui para uma menor experiência do usuário confuso.

Se você optar por usar `MultiFieldQueryParser`, Certifique-se suas consultas são fabricados adequadamente com o `QueryParser` e Analisador técnicas de diagnóstico mostrado nos capítulos 3 e 4. Abundância de interações estranho com a análise ocorrer usando Consulta-Analisador, E estes são agravados com `MultiFieldQueryParser`.

## 5.4 consultas Span: nova jóia Lucene está escondido

---

Lucene 1.4 inclui uma nova família de consultas, todos baseados em `SpanQuery`. A vão em Neste contexto é uma posição inicial e final em um campo. Lembre-se do ponto 4.2.1 que tokens emitidos durante o processo de análise incluem um incremento de posição do token anterior. Esta informação da posição, em conjunto com o novo `SpanQuery` subclasses, permitir a consulta a discriminação ainda mais sofisticação e-ção, tais como todos os documentos onde "rápida raposa" está perto "cão preguiçoso".

Usando os tipos de consulta que discutimos até agora, não é possível formular como uma consulta. Consultas frase poderia chegar perto com algo como "Fox rápido" E "Cão preguiçoso", Mas estas frases podem ser muito distantes uns dos outros para ser relevante para os nossos propósitos a pesquisa. Felizmente, Doug Cutting nos agraciou com seu brilho mais uma vez e acrescentou consultas span ao núcleo do Lucene.

Consultas span faixa mais do que os documentos que combinam: O indivíduo vão, talvez mais do que um por campo, são rastreadas. Contrastando com `TermQuery`, que simplesmente jogos documentos, por exemplo, `SpanTermQuery` mantém o controle do posições de cada um dos termos que correspondem.

Há cinco subclasses da base `SpanQuery`, Mostrado na tabela 5.1.

Tabela 5.1 SpanQuery família

SpanQuery tipo	Descrição
<code>SpanTermQuery</code>	Utilizado em conjunto com o outros tipos de consulta span. Por si só, é funcionalmente equivalente a <code>TermQuery</code> .
<code>SpanFirstQuery</code>	Jogos vãos que ocorrem dentro da primeira parte de um campo.
<code>SpanNearQuery</code>	Jogos vãos que ocorrem perto um do outro.
<code>SpanNotQuery</code>	Abrange jogos que não se sobrepõem uns aos outros.
<code>SpanOrQuery</code>	Agregados partidas de consultas span.

Vamos discutir cada um destes `SpanQuery` tipos dentro do contexto de um caso de teste JUnit, `SpanQueryTest`. A fim de demonstrar cada um desses tipos, um pouco de configuração é necessários, bem como alguns auxiliares afirmam métodos para tornar o nosso código mais claro mais tarde, como mostrado na listagem 5.3. Nós índice de duas frases semelhantes em um campo `fcomo` separada documentos e criar `SpanTermQuerys` para vários dos termos para uso posterior em nosso teste métodos. Além disso, nós adicionamos três conveniência afirmar métodos para agilizar nossos exemplos.

#### Listagem 5.3 SpanQuery infra-estrutura de demonstração

```
public class SpanQueryTest estende TestCase {
    RAMDirectory diretório privado;
    privada IndexSearcher pesquisador;
    privada IndexReader leitor;

    privado SpanTermQuery rápida;
    privado SpanTermQuery marron;
    privado SpanTermQuery vermelho;
    privado SpanTermQuery raposa;
    privado SpanTermQuery preguiçoso;
    privado SpanTermQuery sonolento;
    privado SpanTermQuery cão;
    privado Cat SpanTermQuery;
    privado Analisador analisador;

    setUp protected void () throws Exception {
        diretório = new RAMDirectory ();

        analisador = new WhitespaceAnalyzer ();
        Escritor IndexWriter IndexWriter = new (diretório,
            analisador, true);
```

```

Documento doc = new Document ();
doc.add (Field.Text ("f",
    "A ligeira raposa marrom ataca o cão preguiçoso"));
writer.addDocument (doc);

doc = new Document ();
doc.add (Field.Text ("f",
    "A rápida raposa vermelha saltos sobre o gato sonolento"));
writer.addDocument (doc);

writer.Close ();

searcher = new IndexSearcher (diretório);
reader = IndexReader.open (diretório);

rápida = new SpanTermQuery (Termo de novo ("f", "rápido"));
marrom = new SpanTermQuery (Termo de novo ("f", "marrom"));
vermelho = new SpanTermQuery (Termo de novo ("f", "vermelho"));
fox = new SpanTermQuery (Termo de novo ("f", "fox"));
lazy = new SpanTermQuery (Termo de novo ("f", "preguiçoso"));
sleepy = new SpanTermQuery (novo Termo ("f", "sonolento"));
dog = new SpanTermQuery (Termo de novo ("f", "cachorro"));
cat = new SpanTermQuery (Termo de novo ("f", "gato"));

}

private void assertOnlyBrownFox (Query) throws Exception {
    Hits hits = searcher.search (query);
    assertEquals (1, hits.length ());
    assertEquals ("doc errado", 0, hits.id (0));
}

assertBothFoxes private void (Query) throws Exception {
    Hits hits = searcher.search (query);
    assertEquals (2, hits.length ());
}

assertNoMatches private void (Query) throws Exception {
    Hits hits = searcher.search (query);
    assertEquals (0 hits.length, ());
}

}

```

Com este bit necessária de instalação para fora do caminho, podemos começar a explorar span queries. Primeiro vamos chão-nos com SpanTermQuery.

#### 5.4.1 Construção de bloco abrangendo SpanTermQuery,

Consultas span precisa de um ponto de alavancagem inicial, e SpanTermQuery é apenas isso. Internamente, um SpanQuery se mantém informado dos seus jogos: uma série de início / fim posições para



Figura 5.1 SpanTermQuery para marrom

cada documento correspondente. Por si só, um `SpanTermQuery` documentos partidas como `TermQuery` faz, mas também mantém o controle da posição dos mesmos termos que aparecem dentro de cada documento.

Figura 5.1 ilustra o `SpanTermQuery` resultados para este código:

```
testSpanTermQuery public void () throws Exception {
    assertOnlyBrownFox (marrom);
    dumpSpans (marrom);
}
```

O `marrom` `SpanTermQuery` foi criado em `setUp ()` porque ele vai ser usado em outros testes que se seguem. Nós desenvolvemos um método, `dumpSpans`, Para visualizar spans. O `dumpSpans` método utiliza alguns de nível inferior `SpanQuery` API para navegar as extensões, o que API de nível inferior, provavelmente, não é de grande interesse para você que não para de diagnóstico fins, por isso não elaborar mais sobre ele. Cada `SpanQuery` esporte uma subclasse útil `toString ()` para fins de diagnóstico, que `dumpSpans` usos:

```
dumpSpans private void (consulta SpanQuery) throws IOException {
    Abrange abrange = query.getSpans (leitor);
    System.out.println (consulta + ":" );
    int numSpans = 0;

    Hits hits = searcher.search (query);
    float [] = new float notas [2];
    for (int i = 0; i <hits.length (); i + +) {
        pontuações [hits.id (i)] = hits.score (i);
    }

    while (spans.next ()) {
        numSpans + +;

        int id = spans.doc ();
        Documento doc = reader.document (id);

        / / Para simplicidade - assume tokens são seqüenciais,
        / / Posições, a partir de 0
        Token [] = AnalyzerUtils.tokensFromAnalysis tokens (
            analisador, doc.get ("f"));
        StringBuffer buffer = new StringBuffer ();
        buffer.append ("");
        for (int i = 0; i <tokens.length <; i + +) {
```

```

        if (i == spans.start ()) {
            buffer.append ("<");
        }
        buffer.append (tokens [i] termText ());
        if (i + 1 == spans.end ()) {
            buffer.append (">");
        }
        buffer.append ("");
    }

    buffer.append ("(" + notas [id] + ")");
    System.out.println (buffer);
    System.out.println (searcher.explain (consulta, id));
    /
}
}

if (numSpans == 0) {
    System.out.println ("      Não abrange ");
}
System.out.println ();
}
}

```

A saída de dumpSpans (marrom) é

```
f: marrom:
  a raposa <brown> rápida salta sobre o cão preguiçoso (0.22097087)
```

Mais interessante é a dumpSpans saída de um SpanTermQuery para a:

```
dumpSpans (novo SpanTermQuery (Termo de novo ("f", "o")));
f: o:
<the> ligeira raposa marrom salta sobre o preguiçoso (0.18579213)
A ligeira raposa marrom salta sobre <the>preguiçoso (0.18579213)
<the> rápida raposa vermelha saltos sobre o sono gato (0.18579213)
a rápida raposa vermelha saltos mais <the>sonolento (0.18579213)
```

Não foram só os documentos encontrados, mas também cada documento tinha duas span partidas destaque para os suportes. O básico SpanTermQuery é usado como um building bloco de outro SpanQuery tipos.

#### 5.4.2 abrange Encontrar no início de um campo

Para consultar se estende por que ocorrem dentro dos primeiros nposições de um campo, use SpanFirstQuery. Figura 5.2 ilustra um SpanFirstQuery.

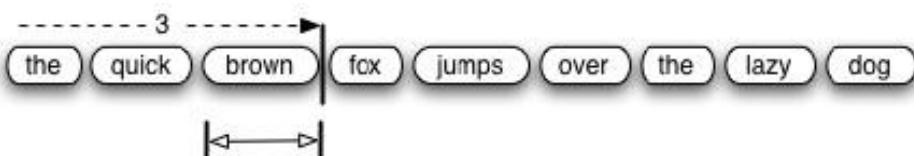


Figura 5.2 SpanFirstQuery

Este teste mostra consultas não correlação e correspondência:

```
testSpanFirstQuery public void () throws Exception {
    SpanFirstQuery sfq = new SpanFirstQuery (marrom, 2);
    assertNoMatches (sfq);

    sfq = new SpanFirstQuery (marrom, 3);
    assertOnlyBrownFox (sfq);
}
```

Nenhuma correspondência foi encontrada na primeira consulta porque o intervalo de 2 é muito curto para encontrar marrom, mas 3 é apenas o tempo suficiente para causar uma correspondência na segunda consulta (veja figura 5.2). Qualquer `SpanQuery` pode ser usado dentro de um `SpanFirstQuery`, Com resultados que correspondem à vãos que têm uma posição final na primeira n(2 e 3, neste caso) posições.

As partidas span resultantes são o mesmo que o original `SpanQuery` abrange, neste caso o mesmo `dumpSpans ()` saída para marrom como visto no ponto 5.4.1.

#### 5.4.3 Spans perto um do outro

A `PhraseQuery` (Ver seção 3.4.5) corresponde a documentos que tenham termos perto de um outro, com um fator de slop para permitir termos intermediários ou revertida. `SpanNearQuery` opera de forma semelhante ao `PhraseQuery`, Com algumas diferenças importantes. `SpanNearQuery` abrange jogos que estão dentro de um certo número de posições de um outro, com uma bandeira em separado indicando se os vãos devem estar na ordem especificada ou pode ser revertida. Os vãos correspondentes resultantes vão desde a posição inicial da primeira span seqüencialmente para a posição final dos últimos span. Um exemplo de uma `SpanNearQuery` dado três `SpanTermQuery` objetos é mostrado na figura 5.3.

Utilização `SpanTermQuery` objetos como o `SpanQuery`s em um `SpanNearQuery` é muito como um `PhraseQuery`. No entanto, o `SpanNearQuery` fator de slop é um pouco menos confusa do que o `PhraseQuery` fator de slop, porque não requer, pelo menos, mais dois posições para explicar uma extensão invertida. Para reverter uma `SpanNearQuery`, Defina o `inorder` flag (terceiro argumento para o construtor) para falso. Demônio listagem 5.4 siveis algumas variações de `SpanNearQuery` e mostra que em relação ao `PhraseQuery`.

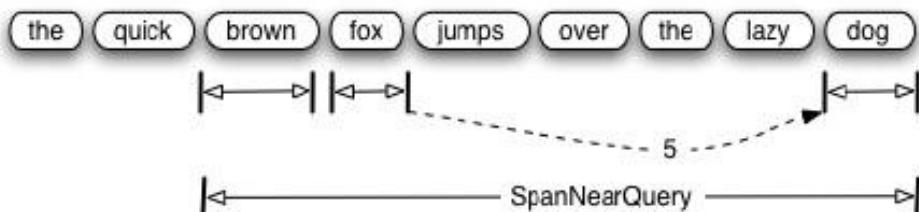


Figura 5.3 SpanNearQuery

## Listagem 5.4 SpanNearQuery

```

testSpanNearQuery public void () throws Exception {
    SpanQuery [] = quick_brown_dog
        nova SpanQuery [] {rápida, marrom, cão};
    SpanNearQuery SNQ =
        nova SpanNearQuery (quick_brown_dog, 0, true);
    assertNoMatches (SNQ);                                b Consulta por três
                                                        mandatos sucessivos

    SNQ = new SpanNearQuery (quick_brown_dog, 4, true);   c Mesmos termos,
                                                        slop de 4

    SNQ = new SpanNearQuery (quick_brown_dog, 5, true);   d SpanNearQuery
                                                        fósforos

    / / Interessante - até mesmo uma consulta frase descuidada exigiria
    / / Mais slop para combinar
    SNQ = new SpanNearQuery (novo SpanQuery [] {preguiçoso, fox}, 3, false);
    assertOnlyBrownFox (SNQ);                            e

PhraseQuery pq = PhraseQuery new ();
pq.add (Termo de novo ("f", "preguiçoso"));
pq.add (Termo de novo ("f", "fox"));
pq.setSlop (4);
assertNoMatches (pq);                                f Comparável
                                                        PhraseQuery

pq.setSlop (5);
assertOnlyBrownFox (pq);                            g PhraseQuery,
                                                        slop de 5
}

```

- b Consultando para estes três termos em posições sucessivas não corresponde nem doc que este documento.
- c Usando os mesmos termos com um slop de 4 posições ainda não resulta em uma partida.
- d A nested `SpanTermQuery` objetos estão em ordem inversa, de modo que o `inorder` Sinalizar é definido
- e para `falso`. A slop de apenas 3 é necessário para uma partida.
- f Aqui usamos um nível comparável `PhraseQuery`, Embora um slop de 4 ainda não corresponde.
- g A slop de 5 é necessário para um `PhraseQuery` para combinar.

Nós só mostrado `SpanNearQuery` com aninhadas `SpanTermQuery`s, mas `SpanNearQuery` permite que qualquer `SpanQuery` tipo. A mais sofisticada `SpanNearQuery` é demonstram mais tarde na listagem 5.5 em conjunto com `SpanOrQuery`.

#### 5.4.4 Excluindo span sobreposição de jogos

O `SpanNotQuery` exclui partidas onde um `SpanQuery` sobrepõe-se outro. O código a seguir demonstra:

```
testSpanNotQuery public void () throws Exception {
    SpanNearQuery quick_fox =
        nova SpanNearQuery (novo SpanQuery [] {rápida, fox}, 1, true);
    assertBothFoxes (quick_fox);
    dumpSpans (quick_fox);

    SpanNotQuery quick_fox_dog = new SpanNotQuery (quick_fox, cão);
    assertBothFoxes (quick_fox_dog);
    dumpSpans (quick_fox_dog);

    SpanNotQuery no_quick_red_fox =
        nova SpanNotQuery (quick_fox, vermelho);
    assertOnlyBrownFox (no_quick_red_fox);
    dumpSpans (no_quick_red_fox);
}
```

O primeiro argumento para o `SpanNotQuery` construtor é um espaço para incluir, e a segunda argumento é o span para excluir. Nós estrategicamente adicionada `dumpSpans` para esclarecer o que está acontecendo. Aqui está a saída com a consulta Java anotado acima de cada:

```
SpanNearQuery quick_fox =
    nova SpanNearQuery (novo SpanQuery [] {rápida, fox}, 1, true);
spanNear ([f: rápida, f: fox], 1, true):
    o marrom <Marcação fox> salta sobre o cão preguiçoso (0.18579213)
    o vermelho <Marcação saltos fox> sobre o gato sonolento (0.18579213)

SpanNotQuery quick_fox_dog = New SpanNotQuery (quick_fox, cão);
spanNot (spanNear ([f: rápida, f: fox], 1, true), f: cão):
    o marrom <Marcação fox> salta sobre o cão preguiçoso (0.18579213)
    o vermelho <Marcação saltos fox> sobre o gato sonolento (0.18579213)

SpanNotQuery no_quick_red_fox =
    nova SpanNotQuery (quick_fox, vermelho);
spanNot (spanNear ([f: rápida, f: fox], 1, true), f: vermelho):
    o marrom <Marcação fox> salta sobre o cão preguiçoso (0.18579213)
```

O `SpanNear` consulta combinados ambos os documentos, porque ambos têm rápido e raposa dentro de uma posição um do outro. O primeiro `SpanNotQuery`, `quick_fox_dog`, continua para combinar ambos os documentos, porque não há sobreposição com o `quick_fox` vazio e cão. O segundo `SpanNotQuery`, `no_quick_red_fox`, Exclui o segundo documento, porque vermelho coincide com a `quick_fox` span. Observe que o span resultante partidas são a extensão original incluído. O tempo de excluídos é apenas usado para determinar se há uma sobreposição e não fator para os jogos span resultante.

### 5.4.5 em todo o globo

Finalmente, há `SpanOrQuery`, Que agrega um conjunto de `SpanQuerys`. Nossa exemplo consulta plo, em Inglês, é todos os documentos que têm "rápida raposa" perto de "cão preguiçoso" ou que têm "rápida raposa" perto de "gato sonolento". A primeira cláusula desta consulta é mostrado na figura 5.4. Esta cláusula é única `SpanNearQuery` dois ninhos `SpanNearQuerys`, que cada um, dois `SpanTermQuerys`.

Nosso caso de teste torna-se um pouco demorado devido a todos os sub-`SpanQuerys` ser construída em cima (veja a lista 5.5). Utilização `dumpSpans`, Analisamos o código com mais detalhes.

Listagem 5.5 `SpanOrQuery`

```
testSpanOrQuery public void () throws Exception {
    SpanNearQuery quick_fox =
        nova SpanNearQuery (novo SpanQuery [] {rápida, fox}, 1, true);
    SpanNearQuery lazy_dog =
        nova SpanNearQuery (novo SpanQuery [] {cão, preguiçoso}, 0, true);

    SpanNearQuery sleepy_cat =
        nova SpanNearQuery (novo SpanQuery [] {sonolento, cat}, 0, true);

    SpanNearQuery qf_near_ld =
        nova SpanNearQuery (
            nova SpanQuery [] {quick_fox, lazy_dog}, 3, true);
    assertOnlyBrownFox (qf_near_ld);
    dumpSpans (qf_near_ld);

    SpanNearQuery qf_near_sc =
        nova SpanNearQuery (
            nova SpanQuery [] {quick_fox, sleepy_cat}, 3, true);
    dumpSpans (qf_near_sc);

    SpanOrQuery ou = new SpanOrQuery (
        nova SpanQuery [] {qf_near_ld, qf_near_sc});
    assertBothFoxes (ou);
    dumpSpans (ou);
}
```

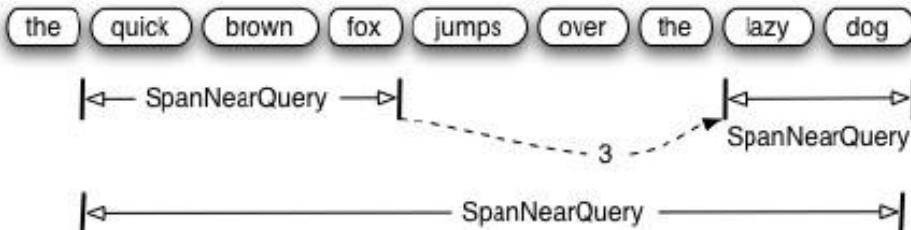


Figura 5.4 Uma cláusula do `SpanOrQuery`

Nós usamos a nossa mão `dumpSpans` algumas vezes para permitir-nos a seguir a progressão como a consulta final OR é construído. Aqui está a saída, seguida por nossa análise dele:

```
SpanNearQuery qf_near_ld =
    nova SpanNearQuery (
        nova SpanQuery [] {quick_fox, lazy_dog}, 3, true);
spanNear ([spanNear ([f: rápida, f: fox], 1, true),
           spanNear ([f: f, preguiçoso: cão], 0, true)], 3, true):
    o <Marcação raposa marrom salta sobre o preguiçoso dog> (0.3321948)

SpanNearQuery qf_near_sc =
    nova SpanNearQuery (
        nova SpanQuery [] {quick_fox, sleepy_cat}, 3, true);
spanNear ([spanNear ([f: rápida, f: fox], 1, true),
           spanNear ([f: f, sonolento: gato], 0, true)], 3, true):
    o <Marcação raposa vermelha salta sobre o cat> sonolento (0.3321948)

SpanOrQuery ou = new SpanOrQuery (
    nova SpanQuery [] {qf_near_ld, qf_near_sc});
spanOr ([spanNear ([spanNear ([f: rápida, f: fox], 1, true),
                     spanNear ([f: f, preguiçoso: cão], 0, true)], 3, true),
           spanNear ([spanNear ([f: rápida, f: fox], 1, true),
                     spanNear ([f: f, sonolento: gato], 0, true)], 3, true)]):
    o <Marcação raposa marrom salta sobre o preguiçoso dog> (0.6643896)
    o <Marcação raposa vermelha salta sobre o cat> sonolento (0.6643896)
```

Dois `SpanNearQuery`s são criados para atender rápida raposa perto cão preguiçoso (`qf_near_ld`) E rápida raposa perto gato sleepy (`qf_near_sc`) Utilizando nested `SpanNearQuery`s feito de `SpanTermQuery`s no nível mais baixo. Finalmente, estes dois `SpanNearQuery` instâncias são combinados dentro de um `SpanOrQuery`, Que agrupa todos os vãos correspondentes. Ufa!

#### 5.4.6 SpanQuery e QueryParser

`QueryParser` atualmente não suporta nenhum dos `SpanQuery` tipos. Talvez, porém, o apoio será eventualmente adicionados. Pelo menos um membro do Lucene comunidade criou um analisador de expressão de consulta projetado para consulta span expressões que podem ser parte da Sandbox Lucene pelo tempo que você ler isto. Consulte os recursos listados no Apêndice C para obter mais detalhes sobre como explorar o Lucene comunidade de usuários.

Lembre-se da seção 3.4.5 que `PhraseQuery` é imparcial a fim termo quando `slop` suficiente é especificado. Curiosamente, você pode facilmente estender `QueryParser` usar um `SpanNearQuery` com `SpanTermQuery` cláusulas em vez disso, e consultas força frase para corresponder apenas a campos com os termos na mesma ordem como especificado. Demonstramos esta técnica na seção 6.3.4.

## 5.5 Filtering uma busca

Filtragem é um mecanismo de estreitar o espaço de busca, permitindo que apenas um subconjunto de os documentos a serem considerados como possíveis hits. Eles podem ser usados para implementar busca-dentro-de-busca recursos para pesquisa, sucessivamente, dentro de um conjunto anterior de hits ou para restringir o espaço de busca de documentos por motivos de segurança de dados externos.

Um filtro de segurança é um poderoso exemplo, permitindo que os usuários só vêem os resultados da pesquisa de documentos que, mesmo que se a sua consulta tecnicamente partidas outros documentos que estão fora dos limites; nós fornecemos um exemplo de um filtro de segurança na seção 5.5.3.

DateFilter estringe o espaço de documentos para documentos apenas com um espe-  
cífico campo de data dentro de um determinado intervalo de datas.

- Você pode filtrar qualquer busca Lucene, usando o sobrecarregado pesquisa método que aceitaria o campo parâmetro da sua implementação:
- `QueryFilter` utiliza os resultados de consulta como o espaço do documento pesquisado para nova consulta.
  - `CachingWrapperFilter` é um decorador em detrimento de outro filtro de caching seus resultados para aumentar o desempenho quando usado novamente.

Antes de chegar preocupados com menções de armazenamento de resultados, a certeza de que é feito com uma estrutura de dados pequeno (a `BitSet`) Onde cada posição de bit representa um documento.

Considere, também, a alternativa ao uso de um filtro: agregar cláusulas necessárias em um `BooleanQuery`. Nesta seção, vamos discutir cada um dos filtros built-in, assim como o `BooleanQuery` alternativa.

### 5.5.1 Usando DateFilter

O tipo de campo data é coberto na seção 2.4, juntamente com a sua ressalvas. Ter um data campo, filtro, como mostrado na `testDateFilter ()` em 5.6 listagem. Nossos dados livros índices a data modificada pela última vez de cada arquivo de dados como um livro modificado campo, indexados como um `Field.Keyword (Date, String)`. Nós testamos o filtro intervalo de datas usando um all-consulta, inclusive, que por si só retorna todos os documentos.

#### Usando 5.6 listagem DateFilter

```
public class FilterTest estende LiaTestCase {  
    privada allBooks Query;  
    privada IndexSearcher pesquisador;  
    privada numAllBooks int;
```

```

setUp protected void () throws Exception {
    super.setUp ();

    allBooks RangeQuery = new (novo Termo ("pubmonth", "190001"),
                                novo prazo ("pubmonth", "200512"),
                                true);
    searcher = new IndexSearcher (diretório);
    Hits hits = searcher.search (allBooks);
    numAllBooks hits.length = ();

}

público testDateFilter void () throws Exception {
    Data jan1 = ParseDate ("2004 Jan 01");
    Data jan31 = ParseDate ("2004 31 de janeiro");
    Data dec31 = ParseDate ("2004 31 dezembro");

    DateFilter filtro = new DateFilter ("modificado", jan1, dec31);

    Hits hits = searcher.search (allBooks, filtro);
    assertEquals ("todos modificada em 2004",
                 numAllBooks hits.length, ());

    filtro = new DateFilter ("modificado", jan1, jan31);
    hits = searcher.search (allBooks, filtro);
    assertEquals (none " modificado em janeiro",
                 0 hits.length, ());
}

```

- b** setUp () estabelece contagem de livro de referência

- b** setUp () estabelece uma contagem inicial de todos os livros em nosso índice, permitindo comparações quando usamos um filtro de data todos os inclusive.

O primeiro parâmetro para ambos os DateFilter construtores é o nome de uma data campo no índice. Em dados de nossa amostra o nome do campo é `modificado`; Neste campo é o data da última modificação do arquivo de dados de origem. Os dois construtores diferem apenas no tipos de segundo e terceiro argumentos: ou `java.util.Date` (Como neste exemplo) ou `longo`, Faça a sua escolha.

Open-ended intervalo de datas de filtragem

DateFilter também suporta varia open-ended data. Para filtrar em datas com um final do intervalo especificado ea outra extremidade aberta, use um dos fábrica estática métodos em DateFilter:

```

filter = DateFilter.Before ("modificado", endDate);
filter = DateFilter.After ("modificado", startDate);

```

**NOTA** `DateFilter` intervalos são inclusivo do início e final datas. O `Antes` e `Depois` nomes de método pode ser enganosa dado este fato. A `DateFilter.Before` faixa é realmente um "em ou antes de" filtro.

Como com a `DateFilter` construtores, `Antes` e `Depois` métodos aceitar nem uma `java.util.Date` ou um longo.

Você pode deixar ambas as extremidades do intervalo de datas em aberto, embora fazê-lo é eficiente o mesmo que usar nenhum filtro, mas com um impacto no desempenho para a comunicação com os filhos. É complicado deixar ambas as extremidades sem restrições, porque os únicos métodos para obter o mínimo de datas especiais e máxima retornam seqüências que devem ser convertidos para uma representação de data, como mostrado aqui:

```
Filtro filtro = new DateFilter ("modificada",
    DateField.stringToDate (DateField.MIN_DATE_STRING ()),
    DateField.stringToDate (DateField.MAX_DATE_STRING ());
```

Não faria muito sentido para embutir tal um open-ended `DateFilter`, Mas essas constantes seria útil como casos especiais, quando você está construindo uma `DataFiltro` dinamicamente.

#### DateFilter e cache

Os filtros são ideais quando estão reutilizado para muitas pesquisas, com a ressalva que o seu trabalho ser armazenada em cache inicialmente. `DateFilter` No entanto, não armazena em cache, e se você usá-lo várias vezes, ele vai tomar a decisão de filtragem de cada vez com um aviso-degradação de desempenho capaz. Quando você reutilizar um `DateFilter` através de múltiplas pesquisas, envolvê-la com um `CachingWrappingFilter` para beneficiar do cache do documento intervalo que corresponde à primeira pesquisa. Consulte a seção 5.5.5 para obter detalhes sobre cache que estabelece um `DateFilter`.

#### 5.5.2 Usando QueryFilter

Mais útil do que genericamente `DateFilter` é `QueryFilter`. `QueryFilter` usa os hits de uma consulta para restringir os documentos disponíveis a partir de uma pesquisa subsequente. O

resultado, uma `BitSet` representando quais documentos foram pareados a partir da filtragem consulta, é armazenada em cache para maximizar o desempenho em futuras pesquisas que usam o mesmo

`QueryFilter` e `IndexSearcher` instâncias. Usando um `QueryFilter`, Restringimos o documentos pesquisados para uma categoria específica:

```
testQueryFilter public void () throws Exception {
    TermQuery categoryQuery =
        nova TermQuery (novo Termo ("categoria", "/ filosofia / leste"));
```

```

Filtro categoryFilter = new QueryFilter (categoryQuery);

Hits hits = searcher.search (allBooks, categoryFilter);
assertEquals ("apenas Tao Te Ching", 1, hits.length ());
assertTrue (hits.score (0) <1.0);

}

```

Aqui estamos à procura de todos os livros (ver `setUp ()` na listagem 5.6), mas restringindo a busca utilizando um filtro para uma categoria que contém um único livro. Explicamos a última afirmação da `testQueryFilter ()` em breve, na seção 5.5.4.

`QueryFilter` pode até mesmo substituir `DateFilter` uso, embora exija alguns mais linhas de código e não é tão elegante. O código a seguir demonstra como filtrar usando um `QueryFilter` em um `RangeQuery` usando a mesma data alcance e de pesquisa como o primeiro `DateFilter` exemplo:

```

testQueryFilterWithRangeQuery public void () throws Exception {
    Data jan1 = ParseDate ("2004 Jan 01");
    Data dec31 = ParseDate ("2004 31 dezembro");
    Começar a prazo = novo Termo ("modificada",
        DateField.dateToString (jan1));
    Pôr termo = novo Termo ("modificada",
        DateField.dateToString (dec31));

    RangeQuery consulta RangeQuery = new (início, fim, true);
    Filtro filtro = new QueryFilter (rangeQuery);

    Hits hits = searcher.search (allBooks, filtro);
    assertEquals ("todos" em ", numAllBooks, hits.length ());
}

```

Método Unshown:  
Data retorna como esperado

Se você vai ser pendurado em uma instância de filtro para várias pesquisas, o cache de `QueryFilter` resultará em pesquisas mais eficiente do que um similar `DateFilter`, que não faz cache.

### 5.5.3 filtros de segurança

Outro exemplo de filtragem documento restringe documentos com segurança em mente. Nossa exemplo assume documentos estão associados com um proprietário, que é conhecido em tempo de indexação. Nós índice de dois documentos, ambos têm o prazo info em seus palavras-chave campo, mas cada documento tem um proprietário diferente:

```

public class SecurityFilterTest estende TestCase {
    RAMDirectory diretório privado;

    setUp protected void () throws Exception {
        Escritor IndexWriter IndexWriter = new (diretório,
            nova WhitespaceAnalyzer (), true);

```

```

    / / Elwood
    Documento documento = new Document ();
    document.add (Field.Keyword ("dono", "Elwood"));
    document.add (Field.Text ("keywords", "info sensíveis elwoods"));
    writer.addDocument (documento);

    / / Jake
    documento = new Document ();
    document.add (Field.Keyword ("dono", "Jake"));
    document.add (Field.Text ("keywords", "info jakes sensíveis"));
    writer.addDocument (documento);

    writer.Close ();
}
}

```

Usando um `TermQuery` para info no palavras-chave resultados de campo em ambos os documentos encontrados, naturalmente. Suponhamos, porém, que Jake está usando o recurso de busca em nosso aplicação, e somente ele possui documentos devem ser pesquisadas por ele. Muito elegante, podemos facilmente usar um `QueryFilter` para restringir o espaço de busca a documentos só ele é o proprietário de, como mostrado na listagem 5.7.

#### Listagem 5.7 Protegendo o espaço de busca com um filtro

```

testSecurityFilter public void () throws Exception {
    diretório = new RAMDirectory ();
    setUp ();

    Query = TermQuery nova TermQuery (novo Termo ("keywords", "info"));

    IndexSearcher searcher = new IndexSearcher (diretório);
    Hits hits = searcher.search (query);
    assertEquals ("Ambos os documentos match", 2, hits.length ());

    QueryFilter jakeFilter = new QueryFilter (
        nova TermQuery (novo Termo ("dono", "jake")));
    hits = searcher.search (consulta, jakeFilter);
    assertEquals (1, hits.length ());
    assertEquals ("elwood é seguro",
        . "Info jakes sensíveis", hits.doc (0) get ("keywords"));

}

```

**b** TermQuery para "info"  
**c** Retorna documentos contendo "Info"  
**d** Filtro  
**e** Mesmo TermQuery, constrangido resultados

- b Este é um general `TermQuery` para info.
- c Todos os documentos que contenham info são retornados.
- d Aqui, o filtro restringe busca documento para apenas documentos de propriedade de "jake".
- e Apenas o documento de Jake é retornado, usando o mesmo info `TermQuery`.

Se seus requisitos de segurança são este simples, onde os documentos podem ser associado com usuários ou funções durante a indexação, utilizando um `QueryFilter` vai trabalhar muito bem. No entanto, este cenário é simplista para a maioria das necessidades; os caminhos que documents estão associados com papéis podem ser um pouco mais dinâmica. `QueryFilter` é útil apenas quando as restrições estão presentes como a filtragem de informação dentro de campo o índice em si. Na seção 6.4, nós desenvolvemos uma implementação mais sofisticados filtros que utiliza informações externas; esta abordagem poderia ser adaptado a uma filtro de segurança mais dinâmico personalizado.

#### 5.5.4 Uma alternativa `QueryFilter`

Você pode restringir uma consulta a um subconjunto de documentos de outra forma, através da combinação restringindo a consulta para a consulta original como um exigido cláusula de um `BooleanQuery`. Existem algumas diferenças importantes, apesar do fato de que o mesmo documentos são retornados de ambos. `QueryFilter` caches o conjunto de documentos permitido, provavelmente acelerar buscas sucessivas usando a mesma instância. Além disso, normalizada `Hits` pontuações não são suscetíveis de ser o mesmo. A diferença de pontuação faz sentido quando você está olhando para a fórmula de pontuação (ver secção 3.3, página 78). O Fator IDF pode ser dramaticamente diferente. Quando você estiver usando `BooleanQuery` agregação, todos os documentos contendo os termos são tidos em conta na equação, ao passo que um filtro reduz os documentos em apreço e impactos o inverso documento fator de freqüência.

Este caso de teste demonstra como "filtro" usando `BooleanQuery` agregação e ilustra a diferença de pontuação em relação ao `testQueryFilter`:

```
testFilterAlternative public void () throws Exception {
    TermQuery categoryQuery =
        nova TermQuery (novo Termo ("categoria", "/ filosofia / leste"));

    BooleanQuery constrainedQuery BooleanQuery = new ();
    constrainedQuery.add (allBooks, true, false);
    constrainedQuery.add (categoryQuery, true, false);

    Hits hits = searcher.search (constrainedQuery);
    assertEquals ("apenas Tao Te Ching", 1, hits.length ());
    assertTrue (hits.score (0) == 1.0);

}
```

A técnica de agregação de uma consulta desta forma funciona bem com `Consulta-Analisador` analisado consultas, permitindo que os usuários insiram de forma livre consultas ainda restringindo o conjunto de documentos pesquisados por uma consulta API-controlada.

### 5.5.5 Caching filtrar resultados

O maior benefício dos filtros vem quando eles cache e são reutilizados. DateFilter não cache, mas QueryFilter faz. Embrulho com um filtro noncaching Cache WrapperFilter cuida de cache automaticamente (internamente usando uma WeakHashMap). De modo que as entradas dereferenced obter coleta de lixo). Filtros de cache usando o IndexReader como a chave, o que significa pesquisar também deve ser feito com o mesmo instância de IndexReader a beneficiar do cache. Se você não está construindo IndexReader si mesmo, mas estão criando um IndexSearcher a partir de um diretório, você deve usar a mesma instância IndexSearcher a beneficiar do cache. Quando as alterações índice precisa ser refletido nas pesquisas, descarte IndexSearcher e IndexReader e reinstantiate.

Estritamente falando, CachingWrapperFilter é uma terceira built-in filtro dentro Lucene, embora o seu objectivo é dissociar a filtragem da cache e não do filtro. CachingWrapperFilter decora um filtro existente e armazena os resultados em um similares a maneira QueryFilter. Para demonstrar seu uso, voltamos para a data-faixa de filtragem exemplo. Queremos usar DateFilter porque as contorções de usando um QueryFilter para datas são feios, mas nós gostaríamos de beneficiar da cache para melhorar o desempenho:

```
testCachingWrapper public void () throws Exception {
    Data jan1 = ParseDate ("2004 Jan 01");
    Data dec31 = ParseDate ("2004 31 dezembro");

    DateFilter dateFilter =
        nova DateFilter("modificado", jan1, dec31);

    cachingFilter =
        nova CachingWrapperFilter (dateFilter);
    Hits hits = searcher.search (allBooks, cachingFilter);
    assertEquals ("todos" em ", numAllBooks, hits.length ());
}
```

Usos sucessivos do mesmo CachingWrapperFilter exemplo, com o mesmo IndexSearcher instância será bypass usando o filtro envolto, em vez de usar o resultados em cache.

### 5.5.6 Além dos filtros built-in

Lucene não é restrito ao uso dos filtros built-in. Um filtro adicional encontrado em Sandbox do Lucene, ChainedFilter, permite o encadeamento complexo de filtros. Nós cobri-lo na seção 8.8, página 304.

Escrever filtros personalizados permite que os dados externos para fator em restrições de pesquisa;

No entanto, um pouco de detalhada API Lucene know-how pode ser obrigado a ser altamente eficientes. Nós cobrimos escrever filtros personalizados na seção 6.4, página 209.

E se estas opções de filtragem não são suficientes, Lucene 1.4 acrescenta outro interesse no uso de um filtro. O `FilteredQuery` filtra de uma consulta, como `IndexSearcher`'s de busca (`Query, Filter`) pode, exceto que é em si uma consulta: Assim ele pode ser usado como um pecado-  
gle dentro de uma cláusula `BooleanQuery`. Utilização `FilteredQuery` parece fazer sentido apenas quando utilizar filtros personalizados, de modo que cobri-lo juntamente com filtros personalizados

## 5.6 Pesquisando em vários índices Lucene

Se sua arquitetura é composta por vários índices Lucene, mas você precisa procurar através deles usando uma única consulta com resultados de pesquisa interleaving documentos de índices diferentes, `MultiSearcher` é para você. Em alto volume de uso de Lucene, sua arquitetura partição pode conjuntos de documentos em diferentes índices.

### 5.6.1 Usando MultiSearcher

Com `MultiSearcher`, Todos os índices podem ser pesquisados com os resultados mesclados em um specified order (ou descendente-score). Utilização `MultiSearcher` é comparável ao uso de `IndexSearcher`, Exceto que você entregá-lo um conjunto de `IndexSearchers` para pesquisar, em vez do que um único diretório (por isso é efetivamente um padrão de decorador e delegados mais do trabalho para o subsearchers).

Listagem 5.8 ilustra como pesquisar dois índices que são divididas por ordem alfabética por palavra-chave. O índice é composto por nomes de animais que começam com cada letra do alfabeto. Metade dos nomes estão em um índice, e metade são do outro. A pesquisa é realizada com um intervalo que abrange ambos os índices, demonstrando que os resultados são fundidos juntos.

#### Listagem 5.8 Protegendo o espaço de busca com um filtro

```
public class MultiSearcherTest estende TestCase {
    privada IndexSearcher [] pesquisadores;

    setUp public void () throws Exception {
        String [] animais = {"aardvark", "castor", "quati",
            "Cachorro", "elefante", "sapo", "monstro de gila",
            "Cavalo", "iguana", "javali", "canguru",
            "Lemur", "alces", "nematóides", "orca",
            "Python", "quokka", "rato", "escorpião",
            "Tarantula", "Uromastyx", "vicuna",
            "Walrus", "xiphias", "yak", "zebra"};
```

```
Analisador analisador = new WhitespaceAnalyzer();
```

```
Directory aTOmDirectory = new RAMDirectory();
Directory nTOzDirectory = new RAMDirectory();
```

b Dois  
índices

```
aTOmWriter IndexWriter IndexWriter = new (aTOmDirectory,
                                            analisador, true);
nTOzWriter IndexWriter IndexWriter = new (nTOzDirectory,
                                            analisador, true);
```

```
for (int i = 0; i < animals.length; i++) {
    Documento doc = new Document();
    String animal = animais [i];
    doc.add (Field.Keyword ("animal", animal));
    if (animal.compareToIgnoreCase ("n") <0) {
        aTOmWriter.addDocument (doc);
    } else {
        nTOzWriter.addDocument (doc);
    }
}
```

c Metades indexação  
do alfabeto

```
aTOmWriter.close ();
nTOzWriter.close ();
```

```
pesquisadores = new IndexSearcher [2];
pesquisadores [0] = new IndexSearcher (aTOmDirectory);
pesquisadores [1] = new IndexSearcher (nTOzDirectory);
```

```
}
```

```
public void testMulti () throws Exception {
```

```
MultiSearcher searcher = new MultiSearcher (pesquisadores);
```

```
Query = new RangeQuery (Termo de novo ("animal", "h"),
                        Termo de novo ("animal", "t"), true);
```

d Pergunta  
abrange tanto  
índices

```
Hits hits = searcher.search (query);
assertEquals ("tarantula não incluída", 12, hits.length());
```

```
}
```

```
}
```

b Esse código usa dois índices.

c A primeira metade do alfabeto é indexado a um índice, e a outra metade é indexados ao índice de outras.

Esta consulta abrange documentos em ambos os índices.

d

O inclusiva RangeQuery animais encontrados, que começou com h através de animais que começou com t, com os documentos correspondentes provenientes de ambos os índices.

### 5.6.2 Multithreaded pesquisar usando ParallelMultiSearcher

A versão multithreaded do MultiSearcher chamado ParallelMultiSearcher foi adicionado ao Lucene 1.4. A operação de busca gira um thread para cada Pesquisável e espera por todos eles para terminar. A pesquisa básica e pesquisa com as opções do filtro são paralelizado, mas procurando com um HitCollector ainda não foi paralelizado.

Se você verá ganhos de desempenho usando ParallelMultiSearcher muito depende de sua arquitetura. Supostamente, se os índices residem em diferentes discos físicos e você é capaz de tirar proveito de múltiplos processadores, pode haver melhor desempenho, mas em nossos testes com um único CPU, disco físico único, e vários índices, o desempenho com MultiSearcher foi ligeiramente melhor que ParallelMultiSearcher.

Usando um ParallelMultiSearcher é idêntico ao uso de MultiSearcher. Um exemplo, usando ParallelMultiSearcher remotamente, é mostrada na listagem 5.9.

#### Pesquisar vários índices remotamente

Lucene inclui recurso de pesquisa remota índice através Remote Method Invocation (RMI). Existem inúmeras outras alternativas para expor pesquisa remotamente, como através de serviços web. Esta seção se concentra exclusivamente em Lucene

built-in capacidades; outras implementações são deixados à sua inovação (você pode também tomar emprestadas idéias de projetos como Nutch; ver secção 10.1).

Um servidor RMI se liga a uma instância de RemoteSearchable, Que é uma implementação do Pesquisável interface como IndexSearcher e MultiSearcher. Lado do servidor RemoteSearchable delegados para um concreto Pesquisável, Como um regular IndexSearcher exemplo.

Cientes para o RemoteSearchable invocar métodos de pesquisa de forma idêntica para pesquisa

ção através de um IndexSearcher ou MultiSearcher, Como mostrado ao longo deste chapter. Figura 5.5 ilustra uma configuração remota, em busca possível.

Outras configurações são possíveis, dependendo de suas necessidades. O cliente poderia instanciar um ParallelMultiSearcher sobre vários remoto (e / ou local) índices, e cada servidor pode procurar apenas um índice único.

A fim de demonstrar RemoteSearchable, Montamos um servidor multi-índice configuração, semelhante à figura 5.5, utilizando tanto MultiSearcher e ParallelMultiSearcher , a fim de comparar o desempenho. Nós dividimos o índice WordNet (a database de cerca de 40.000 palavras e seus sinônimos) em 26 índices que representam A através Z, com cada palavra no índice correspondente à sua primeira carta. O servidor expõe duas RMI cliente acessível RemoteSearchables, permitindo aos clientes o acesso ou o serial MultiSearcher ou o ParallelMultiSearcher.

SearchServer é mostrado na listagem 5.9.

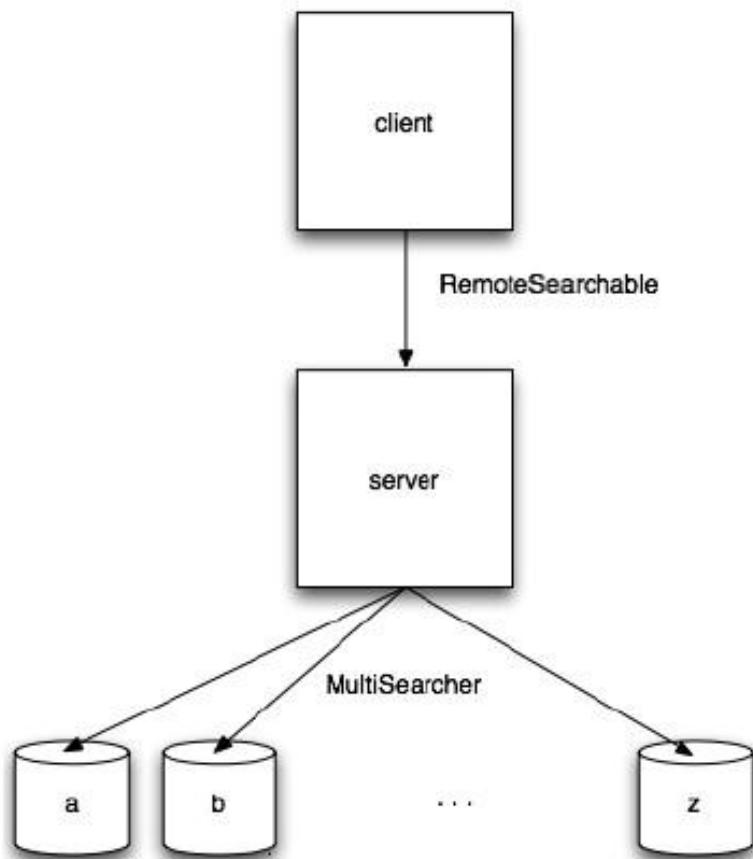


Figura 5.5  
Pesquisa remota  
através de RMI, com a  
pesquisar servidor  
vários índices

#### Listagem 5.9 SearchServer: um servidor remoto usando RMI pesquisa

```

public class {SearchServer
    ALFABETO String private static final =
        "Abcdefghijklmnopqrstuvwxyz";

    public static void main (String [] args) throws Exception {
        if (args.length! = 1) {
            System.err.println ("Uso: <diretório> SearchServer");
            System.exit (-1);
        }

        String basedir = args [0];Índices são basedir
        Pesquisável [] = new Conhecido Searchable [ALPHABET.length ()];
        for (int i = 0; i <ALPHABET.length (); i + +) {
            Conhecido como [i] = new IndexSearcher (
                new File (basedir,
                "" + ALPHABET.charAt (i)) getAbsolutePath ()).;Open IndexSearcher
                para cada índice}
    }
  
```

C

```
LocateRegistry.createRegistry (1099); d Criar RMI Registro
```

```
Searcher multiSearcher = new MultiSearcher (Conhecido);
RemoteSearchable multiImpl =
    nova RemoteSearchable (multiSearcher);
Naming.rebind ("/ / localhost / LIA_Multi", multiImpl); e MultiSearcher
sobre todos os
índices
```

```
Searcher parallelSearcher =
    nova ParallelMultiSearcher (Conhecido);ParallelMultiSearcher
    sobre todos os índicesRemoteSearchable parallelImpl =
        nova RemoteSearchable (parallelSearcher);
Naming.rebind ("/ / localhost / LIA_Parallel", parallelImpl); f
```

```
System.out.println ("Server started");
}
}
```

- b Vinte e seis índices residem sob o `basedir`, Cada um nomeado para uma letra do alfabeto.
- c Uma planície `IndexSearcher` é aberto para cada índice.
- d Um registro RMI é criado.
- e A `MultiSearcher` sobre todos os índices, com o nome `LIA_Multi`, É criado e publicado através de RMI.
- f A `ParallelMultiSearcher` sobre os mesmos índices, com o nome `LIA_Parallel`, É criado e publicado.

Consultando através `SearchServer` remotamente envolve principalmente RMI cola, como mostrado na

`SearchClient` na listagem 5.10. Porque o nosso acesso ao servidor é através de uma `Remoto` Pesquisável, Que é uma API de nível mais baixo do que queremos trabalhar, nós coloca-la dentro `umMultiSearcher`. Por que `MultiSearcher`? Porque é um wrapper sobre `Pesquisáveis`, tornando-o mais amigável para usar como `IndexSearcher`.

Listagem 5.10 `SearchClient`: acessa o RMI-expostos objetos de `SearchServer`

```
SearchClient public class {
    private static HashMap searcherCache = new HashMap ();

    public static void main (String [] args) throws Exception {
        if (args.length! = 1) {
            System.err.println ("Uso: <query> SearchClient");
            System.exit (-1);
        }
    }

    Palavra String = args [0];
```

```

for (int i = 0; i <5; i + +) {
    pesquisa ("LIA_Multi" palavra);
    pesquisa ("LIA_Parallel", palavra);
}

pesquisa private static void (String nome String palavra)
    throws Exception {
    TermQuery query = new TermQuery (Termo de novo ("palavra", palavra));

MultiSearcher searcher =
    (MultiSearcher) searcherCache.get (nome);          c Cache pesquisadores

if (pesquisador == null) {
    Pesquisa =
        nova MultiSearcher (novo Searchable [] {lookupRemote (nome)} );
    searcherCache.put (nome, pesquisador);             d Wrap em Searchable
} MultiSearcher

```

e

```

longo begin = new Date () getTime ();
Hits hits = searcher.search (query);
final longo = new Date () getTime ();
System.out.print ("Procurado" nome + +
    "Para" + palavra + "(" + fim (- começar) + "ms):");

if (hits.length () == 0) {
    System.out.print ("<NONE FOUND>");

for (int i = 0; i <hits.length (); i + +) {
    Documento doc = hits.doc (i);
    String [] valores = doc.getValues ("syn");
    for (int j = 0; j values.length <; j + +) {
        System.out.print (valores [j] + "");
    }
}
System.out.println ();
System.out.println ();

/ / NÃO FECHAR pesquisador!          f Não feche pesquisador
}

private static Searchable lookupRemote (String nome)
    throws Exception {
    Naming.lookup retorno (pesquisável) ("localhost / / /" + name);
}

```

g

RMI pesquisa

- b Realizamos várias buscas idênticos para aquecer a JVM e obter uma boa exemplo do tempo de resposta. O MultiSearcher e ParallelMultiSearcher são cada pesquisados.
- c Os pesquisadores estão em cache, para ser o mais eficiente possível.
- d O controle remoto Pesquisável está localizado e envolto em um MultiSearcher.
- e O processo de busca é cronometrado.
- f Nós não fechamos o pesquisador, porque ele fecha o pesquisador remoto, assim, proibindo pesquisas futuras.
- g Procurar a interface remota.

g

**AVISO** Não `close ()` o `RemoteSearchable` ou embalar seus `MultiSearcher`. Isso irá prevenir futuras pesquisas de trabalhar porque o servidor lado terá fechado seu acesso ao índice.

Vamos ver o nosso pesquisador remoto em ação. Para fins de demonstração, ele correu em uma única máquina em janelas de console separadas. O servidor é iniciado:

```
% Java lia.advsearching.remote.SearchServer caminho / para / índices /
Servidor iniciado
```

O cliente se conecta, pesquisas, produz os resultados várias vezes, e sai:

```
% Java lia.advsearching.remote.SearchClient Olá
Pesquisado LIA_Multi para 'Olá' (259 ms): olá howdy oi

LIA_Parallel procurado 'Olá' (40 ms): olá howdy oi

Pesquisado LIA_Multi para 'Olá' (17 ms): olá howdy oi

LIA_Parallel procurado 'Olá' (83 ms): olá howdy oi

Pesquisado LIA_Multi para 'Olá' (11 ms): olá howdy oi

LIA_Parallel procurado 'Olá' (41 ms): olá howdy oi

Pesquisado LIA_Multi para 'Olá' (30 ms): olá howdy oi

LIA_Parallel procurado 'Olá' (50 ms): olá howdy oi

Pesquisado LIA_Multi para 'Olá' (15 ms): olá howdy oi

LIA_Parallel procurado 'Olá' (47 ms): olá howdy oi
```

É interessante notar a busca vezes relatados por cada tipo de servidor pesquisador. O ParallelMultiSearcher é mais lento do que o MultiSearcher em nosso ambiente (single CPU disco, único). Além disso, você pode ver a razão pela qual optamos para executar a pesquisa múltiplas vezes: A primeira pesquisa demorou muito mais em relação ao

as buscas sucessivas, o que provavelmente é devido a aquecimento JVM. Estes resultados salientar que os testes de desempenho é um negócio complicado, mas é necessário em muitos ambientes. Por causa do forte efeito do ambiente tem sobre desempenho, recomendamos que você realize seus próprios testes com seu próprio ambiente. Per-

testes de desempenho é abordado com mais detalhe na seção 6.5, página 213.

Se você optar por expor pesquisar através de RMI desta maneira, é provável que você quero criar um pouco de infra-estrutura para coordenar e gerir questões como fechando um índice e como o servidor lida com atualizações índice (lembre-se, a pesquisador enxerga um instantâneo do índice e deve ser reaberto para ver as mudanças).

## Aproveitando 5.7 vetores prazo

Vetores prazo são um novo recurso do Lucene 1.4, mas eles não são novos como informe conceito de recuperação ção. A vector prazo é uma coleção de duração-freqüência pares. A maioria

de nós provavelmente não pode imaginar vetores no espaço hyperdimensional, portanto, para visualiza-

fins ção, vamos olhar para dois documentos que contêm apenas os termos gato e cão. Estas palavras aparecem várias vezes em cada documento. Tramando o termo freqüências de cada documento em coordenadas X, Y é algo como a figura 5.6.

O que fica interessante com vetores termo é o ângulo entre eles, como você verá em mais pormenor na secção 5.7.2.

Para habilitar o termo vetor de armazenamento, durante a indexação de habilitar o prazo loja vetores atributo em os campos desejados. Field.Text e Field.Unstored ter adicional métodos sobrecarregados com um boolean storeTermVector bandeira no signatura. Definir esse valor para verdadeiro liga o suporte opcional para o termo do vetor campo, como fizemos para o assunto campo quando indexação de dados nosso livro (ver figura 5.7).

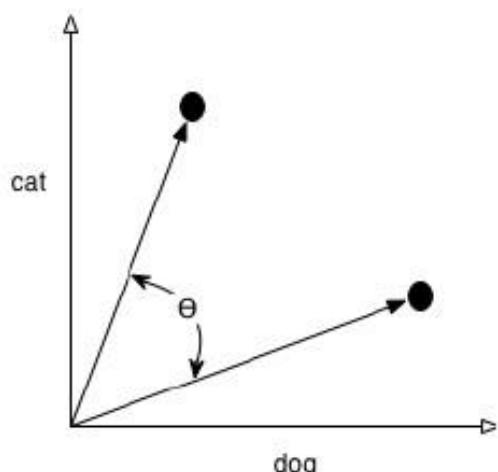


Figura 5.6  
Vetores prazo para dois documentos  
contendo os termos gato e cão

```
doc.add(Field.UnStored("subject", subject, true));
```



Figura 5.7 Permitindo vetores termo durante a indexação

Recuperando vetores prazo para um campo em um determinado documento por ID requer uma chamada para um

IndexReader método:

```
TermFreqVector termFreqVector =  
    reader.termFreqVector(id, "sujeito");
```

ATermFreqVector exemplo, tem vários métodos para recuperar o vetor de informações, principalmente como matrizes de correspondência `Cordas` e `ints` (o valor e prazo freqüência no campo, respectivamente). Você pode usar vetores prazo para algum interessing efeitos, tais como encontrar documentos "como" um documento particular, que é um exemplo de análise semântica latente. Nós construímos uma `BooksLikeThis` recurso, bem como um

prova de conceito categorizador que pode nos dizer a categoria mais adequada para um novo livro, como você verá nas seções seguintes.

### 5.7.1 Livros como este

Seria bom para oferecer outras opções aos clientes de nossa livraria, quando que está vendendo um livro particular. As alternativas devem estar relacionados com o original livro, mas alternativas associando manualmente seria trabalhoso e exigiria esforço contínuo para manter-se atualizado. Em vez disso, usamos Lucene boolean recurso de consulta e as informações de um livro para procurar outros livros que são semelhantes. Listagem 5.11 demonstra uma abordagem básica para encontrar livros como cada um em nossa amostra de dados.

#### Listagem Books 5,11 como esta

```
publica BooksLikeThis classe {  
  
    public static void (String [] args) throws IOException {principal  
        String indexDir = System.getProperty ("index.dir");  
  
        Diretório FSDirectory =  
            FSDirectory.getDirectory (indexDir, false);  
  
        IndexReader reader = IndexReader.open (diretório);  
        int numDocs reader.maxDoc = ();  
  
        BooksLikeThis blt = new BooksLikeThis (leitor);
```

```

for (int i = 0; i < numDocs; i++) {
    System.out.println ();
    Documento doc = reader.document (i);
    System.out.println (doc.get ("title"));

        Busca de livros
    Documento [] = docs blt.docsLike (i, 10); assim C
    if (docs.length == 0) {
        Iterar
        System.out.println ("Nenhum como este");
        todos os livros
    }
    for (int j = 0; j < docs.length; j++) {
        Documento likeThisDoc = docs [j];
        System.out.println ("->" + likeThisDoc.get ("title"));
    }
}

privada IndexReader leitor;
privada IndexSearcher pesquisador;

pública BooksLikeThis (IndexReader leitor) {
    this.reader = leitor;
    searcher = new IndexSearcher (leitor);
}

Documento público [] docsLike (int id, int max) throws IOException {
    Documento doc = reader.document (id);

        Aumenta a livros de
        mesmo autorAutor String = autores [i];
        authorQuery.add (novo TermQuery (Termo de novo ("autor", autor)),
                         false, false);
    }
    authorQuery.setBoost (2.0f);

        Use termos de
        Termo "sujeito"
        vetores
}

BooleanQuery subjectQuery BooleanQuery = new ();
for (int j = 0; j < vector.size (); j++) {
    TermQuery tq = new TermQuery (
        Termo de novo ("sujeito", vector.getTerms () [j]));
    subjectQuery.add (tq, false, false);
}

BooleanQuery likeThisQuery BooleanQuery = new ();
likeThisQuery.add (authorQuery, false, false);
likeThisQuery.add (subjectQuery, false, false);

```

f Criar final pergunta

e Use termos de Termo "sujeito" vetores

d

b

```

    / / Me auto-excluir
    likeThisQuery.add (novo TermQuery (
        Termo de novo ("isbn", doc.get ("isbn "))), false, true); g Excluir
                                                                livro atual

    / / System.out.println ("Query:" +
    / LikeThisQuery.toString / ("Conteúdo"));
    Hits hits = searcher.search (likeThisQuery);
    int size = max;
    if (max > hits.length ()) size = hits.length ();

    Documento [] docs = new Document [tamanho];
    for (int i = 0; i < tamanho; i + +) {
        docs [i] = hits.doc (i);
    }

    retorno docs;
}

}

```

- b Como exemplo, podemos iterar sobre todos os documentos livro no índice e encontrar livros como cada um deles.
- c Aqui olhamos para os livros que são como um presente.
- d Livros do mesmo autor são considerados iguais e são impulsionados para que eles provavelmente comparecer perante livros de outros autores.
- e Usando os termos do `assunto` vetores prazo, nós adicionamos cada um para uma consulta booleana.
- f Nós combinamos as consultas autor e assunto em uma consulta final boolean.
- g Excluímos o livro atual, que certamente seria o melhor jogo, dada a outros critérios, de consideração.

Em d, Usamos uma forma diferente para obter o valor do `autor` de campo. Foi indexada como vários campos, da forma (mostrado com mais detalhes na seção 8.4, página 284), onde o autor original string é uma lista separada por vírgulas de autor (es) de um livro:

```

String [] = autores author.split ",";
for (int i = 0; i < authors.length; i + +) {
    doc.add (Field.Keyword ("autor", os autores [i]));
}

```

A saída é interessante, mostrando como os nossos livros são conectados através de autor e assunto:

A Arte Moderna da Educação  
-> Mindstorms

Segredos Imperial de Saúde e Longevidade  
Nenhum como este

Tao Te Ching 道德经  
Nenhum como este

Gödel, Escher, Bach: um Eternal Golden Braid  
Nenhum como este

Mindstorms  
-> A Arte Moderna da Educação

Java Desenvolvimento com Ant  
-> Lucene in Action  
-> JUnit em Ação  
-> Extreme Programming Explained

JUnit em Ação  
-> Java Development com Ant

Lucene in Action  
-> Java Development com Ant

Extreme Programming Explained  
-> O programador pragmático  
-> Java Development com Ant

Tapeçaria em Ação  
Nenhum como este

The Pragmatic Programmer  
-> Extreme Programming Explained

Se você gostaria de ver a consulta real usado para cada um, descomente as linhas de saída em direção ao final do `docsLike`.

Os livros-como-este exemplo poderia ter sido feito sem vetores prazo, e não estamos realmente usá-los como vetores neste caso. Nós só usou a conveniência educacional de obter os termos para um determinado campo. Sem vetores prazo, o assunto

campo poderia ter sido novamente analisadas ou indexados tais que os termos sujeito individual foram adicionados separadamente a fim de obter a lista de termos para esse campo (ver secção 8.4

para a discussão de como os dados da amostra foi indexado). Nossa próximo exemplo também usa

o componente de freqüência para um vetor termo de uma maneira muito mais sofisticada.

### 5.7.2 O que categoria?

Cada livro no nosso índice é dada uma única categoria principal: Por exemplo, este livro é classificado como "tecnologia / computadores / programação". A melhor categoria colocação de um novo livro pode ser relativamente óbvia, ou (mais provável) várias possíveis categorias responsáveis pode parecer razoável. Você pode usar vetores prazo para automatizar a decisão. Nós escrevemos um pouco de código que cria um vetor assunto representante para

cada categoria existente. Este representante, arquetípica, vetor é a soma de todos os vetores para cada documento é assunto vector campo.

Com estes vetores representante pré-computadas, nosso objetivo final é um cálculo que podem, dadas algumas palavras-chave assunto para um novo livro, nos dizer o que é a categoria melhor ajuste. Nosso caso de teste utiliza duas cordas assunto exemplo:

```
testCategorization public void () throws Exception {
    assertEquals ("metodologia / tecnologia / programação de computadores / /",
                 getCategory ("metodologia ágil extrema"));
    assertEquals ("educação / / pedagogia",
                 getCategory ("montessori filosofia da educação"));

}
```

A primeira afirmação diz que, com base em nossos dados de exemplo, se um novo livro "Extrema metodologia ágil" palavras-chave em seu sujeito, o melhor ajuste categoria é "/ Tecnologia / computadores / programação / metodologia". A melhor categoria é determinada por encontrar a mais próxima categoria ângulo sábio no espaço para o vector assunto novo livro.

O teste `setUp ()` constrói vetores para cada categoria:

```
public class CategorizerTest estende LiaTestCase {
    Mapa categoryMap;

    setUp protected void () throws Exception {
        super.setUp ();

        categoryMap = new TreeMap ();

        buildCategoryVectors ();
        // DumpCategoryVectors ();
    }

    / / . . .
}
```

Nosso código cria vetores categoria a pé todos os documentos no índice e agregação de vetores assunto livro em um único vetor para associados do livro de gatogoria. Vetores categoria são armazenados em um Mapa, Classificados por nome da categoria. O valor de cada item no mapa categoria é outro mapa com chave de prazo, com o valor de um Número inteiro por sua freqüência:

```
buildCategoryVectors private void () throws IOException {
    IndexReader reader = IndexReader.open (diretório);

    int maxDoc reader.maxDoc = ();
    for (int i = 0; i <maxDoc; i + +) {
```

```

        if (! reader.isDeleted (i)) {
            Documento doc = reader.document (i);
            String = doc.get categoria ("categoria");

            Mapa vectorMap = (Mapa) categoryMap.get (categoria);
            if (vectorMap == null) {
                vectorMap = new TreeMap ();
                categoryMap.put (categoria, vectorMap);
            }

            TermFreqVector termFreqVector =
                reader.getTermFreqVector (i, "sujeito");

            addTermFreqToMap (vectorMap, termFreqVector);
        }
    }
}
}

```

Vector Um livro de freqüência do termo é adicionado ao seu vetor categoria em `addTermFreqToMap`. As matrizes retornado por `getTerms ()` e `getTermFrequencies ()` alinhar com um ao outro de tal forma que a mesma posição em cada um refere-se ao mesmo termo:

```

private void addTermFreqToMap (Mapa vectorMap,
                               TermFreqVector termFreqVector) {
    String [] = termos termFreqVector.getTerms ();
    int [] = freqs termFreqVector.getTermFrequencies ();

    for (int i = 0; i < terms.length; i++) {
        Prazo String = termos [i];

        if (vectorMap.containsKey (termo)) {
            Integer valor = (Integer) vectorMap.get (prazo);
            vectorMap.put (prazo,
                           new Integer (value.intValue () + freqs [i]));
        } Else {
            vectorMap.put (prazo, new Integer (freqs [i]));
        }
    }
}

```

Essa foi a parte fácil construção do vetor categoria de mapas, porque ele só Além envolvidos. Ângulos entre os vetores de computação, no entanto, está mais envolvido matematicamente. No mais simples caso bidimensional, conforme mostrado anteriormente na figura-  
ure 5.6, duas categorias (A e B) têm uma única vetores prazo com base na agregação (Como acabamos de fazer). O mais próximo da categoria, o ângulo-sábio, a indivíduos um novo livro é o jogo nós vamos escolher. Figura 5.8 mostra a equação para calcular um ângulo entre dois vetores.

$$\cos \theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Figura 5.8  
Fórmula para calcular o  
ângulo entre dois vetores

Nosso `getCategory` método percorre todas as categorias, o ângulo de computação entre cada categoria eo novo livro. O menor ângulo é o mais próximo jogo, eo nome da categoria é retornado:

```
private String getCategory (sujeito String) {
    String [] palavras = subject.split ("");
    Iterator iterator = categoryIterator categoryMap.keySet () () .;
    dupla bestAngle = Double.MAX_VALUE;
    String bestCategory = null;

    while (categoryIterator.hasNext ()) {
        Categoria String = (String) categoryIterator.next ();
        duplo ângulo = computeAngle (palavras, categorias);
        if (ângulo <bestAngle) {
            bestAngle = ângulo;
            bestCategory = categoria;
        }
    }

    retorno bestCategory;
}
```

Assumimos que o tema é string de forma separadas por espaços brancos e que cada palavra ocorre apenas uma vez. O cálculo do ângulo leva esses pressupostos em conta para simplificar uma parte da computação. Finalmente, o ângulo de computação entre uma variedade de palavras e uma categoria específica é feita em `computeAngle`, mostrado na listagem 5.12.

#### Listagem 5.12 Computing prazo ângulos vetor para um novo livro contra uma determinada categoria

```
computeAngle dupla privado (String [] palavras, categoria String) {
    Mapa vectorMap = (Mapa) categoryMap.get (categoria);

    int dotProduct = 0;
    int sumOfSquares = 0;
    for (int i = 0; i < words.length <; i + +) {
        Palavra String = palavras [i];
        int categoryWordFreq = 0;

        if (vectorMap.containsKey (palavra)) {
            categoryWordFreq =
                ((Integer) vectorMap.get (palavra)) intValue () .;
        }
    }
}
```

```
dotProduct += categoryWordFreq; Supor que cada palavra tem uma freqüência  
sumOfSquares += categoryWordFreq * categoryWordFreq;  
}  
  
dupla denominador;  
if (sumOfSquares > words.length ==) {  
    denominador = sumOfSquares;  
} Else {}  
denominador = Math.sqrt (sumOfSquares) *  
              Math.sqrt (words.length);  
}  
  
relação de casal = dotProduct / denominador;  
  
Math.acos retorno (ratio);  
}
```



Atalho para evitar  
questão de precisão

- b O cálculo é otimizado com o pressuposto de que cada palavra na matriz tem uma frequência de 1.
- c Multiplicamos a raiz quadrada de N pela raiz quadrada de N. Este atalho pré-aberturas de emitir uma precisão onde a taxa poderia ser maior que 1 (que é um ilegal valor para a função cosseno inverso).

Você deve estar ciente de que a computação ângulos termo vetor entre dois documentos ou, neste caso, entre um documento e uma categoria arquetípica, é computação intensivos. Ela exige de raiz quadrada e cálculos inversos e cosseno podem ser proibitivos em alto volume índices.

## 5,8 Resumo

Este capítulo cobriu algum terreno diversificado, destacando adicionais Lucene built-in funções de busca. Classificação é um acessório novo dramático que lhe dá controle sobre a ordenação dos resultados de pesquisa. O novo SpanQuery família aproveita prazo uma situação de informação para uma precisão maior procura. Filtros restringir doc-que este documento espaço de busca, independentemente da consulta. Lucene inclui suporte para múltiplos (Incluindo paralelo) e busca índice remoto, dando aos desenvolvedores um avanço em arquiteturas distribuídas e escaláveis. E, finalmente, o recurso novo vetor prazo permite efeitos interessantes, como "como este" cálculos ângulo termo vetor.

Esse é o fim da história procurando? Não é bem assim. Lucene também inclui vários formas de alargar o seu comportamento em busca, tais como classificação personalizada, filtragem e análise de expressão de consulta, que cobrimos no capítulo seguinte.

# Estendendo pesquisa

---

A large, light gray, stylized number '6' is positioned on the right side of the page, partially overlapping the horizontal line of the title's subtitle.

## Este capítulo aborda

- Criação de uma ordenação personalizada
- Usando um HitCollector
- Personalizando QueryParser
- Testes de desempenho

Apenas quando você pensou que fosse feito com a pesquisa, aqui estamos novamente com o mesmo mais sobre o assunto! Capítulo 3 discutidos os conceitos básicos de Lucene embutido na capacida- laços, eo capítulo 5 foi bem além do básico em mais avançados do Lucene pesquisar características. Nesses dois capítulos, nós exploramos apenas as características built-in.

Lucene também tem vários pontos de extensão bacana.

Nossa primeira extensão personalizada demonstra personalizado Lucene ganchos de classificação, permitindo-nos implementar uma busca que retorna os resultados em ordem crescente geográfica ordem de proximidade da localização atual do usuário. Em seguida, implementar o seu próprio HitCollector ignora Hits, O que é efetivamente um ouvinte de evento, quando as partidas são detectada durante as buscas.

QueryParser é extensível de várias formas úteis, tais como controle para a data análise e formatação numérica, bem como para desabilitar o desempenho potencial degradante, tais como consultas consultas curinga e fuzzy. Os filtros personalizados permitem informações de fora do índice de incluir em restrições de pesquisa, como factoring algumas informações presentes apenas em um banco de dados relacional em busca Lucene.

E, finalmente, nós exploramos o teste de desempenho usando Lucene JUnitPerf. O testes de desempenho exemplo que oferecemos é um exemplo significativo de testes realmente se tornando uma ferramenta de design em vez de um teste de segurança após o fato.

## 6.1 Usando um método de classificação personalizada

---

Se a classificação por pontuação, ID, ou valores de campo é insuficiente para suas necessidades, permite Lucene você implementar um mecanismo de classificação personalizada, fornecendo o seu próprio implementação da SortComparatorSource interface. Custom implementações classificação são mais útil em situações em que os critérios de classificação não pode ser determinado durante a indexação.

Uma idéia interessante para um mecanismo de classificação personalizada é para ordenar os resultados de busca com base na distância geográfica de um dado location.<sup>1</sup> A localização do dado só é conhecido no momento da pesquisa. Nós criamos uma demonstração simplificada deste conceito utilizando a importante questão: "Qual restaurante de comida mexicana está mais próximo de mim?"

Figura 6.1 mostra um exemplo de restaurantes e suas coordenadas na grade fictícia uma amostra de 10x10 grid.<sup>2</sup>

Os dados de teste é indexado como mostrado na listagem 6.1, com cada lugar recebe um nome, localização em coordenadas X e Y, e um tipo. O tipo campo permite que os nossos dados para

<sup>1</sup> Graças a Tim Jones (o contribuinte de recursos Lucene é meio) para a inspiração.

<sup>2</sup> Estes são reais (tasty!) restaurantes em Tucson, Arizona, uma cidade Erik usado para chamar de lar.

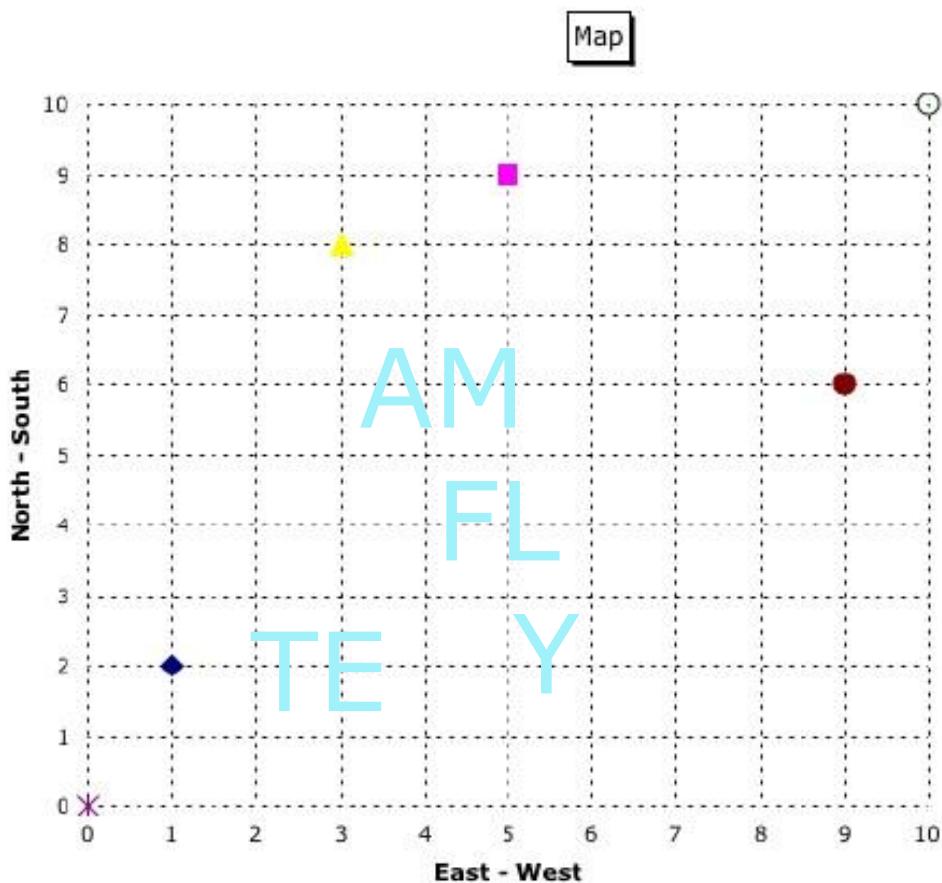


Figura 6.1 Restaurante mexicano que está mais próximo de casa (em 0,0) ou trabalho (em 10,10)?

acomodar outros tipos de empresas e pode permitir-nos para filtrar os resultados da pesquisa a tipos específicos de lugares.

#### Listagem de dados de indexação geográfica 6.1

```
public class DistanceSortingTest estende TestCase {
    RAMDirectory diretório privado;
    privada IndexSearcher pesquisador;
    Query privado;

    setUp protected void () throws Exception {
        diretório = new RAMDirectory ();
        Escritor IndexWriter =
            IndexWriter nova (diretório, novo WhitespaceAnalyzer (), true);
        addPoint (escritor, "El Charro", "restaurante", 1, 2);
        addPoint (escritor, "Cafe Poca Cosa", "restaurante", 5, 9);
```

```
addPoint (escritor, "Los Betos", "restaurante", 9, 6);
addPoint (escritor, "Taco Nico Shop", "restaurante", 3, 8);

writer.Close ();

searcher = new IndexSearcher (diretório);

query = new TermQuery (Termo de novo ("tipo", "restaurante"));
}

addPoint private void (escritor IndexWriter,
                      String nome, String tipo, int x, int y)
throws IOException {
Documento doc = new Document ();
doc.add (Field.Keyword ("name", nome));
doc.add (Field.Keyword ("tipo", tipo));
doc.add (Field.Keyword ("localização", x +","+ y));
writer.addDocument (doc);
}

}
```

As coordenadas são indexados em um campo único local como uma string x, y. O localização poderia ser codificado de várias formas, mas optamos pela abordagem mais simples para este exemplo. Em seguida, escrever um teste que usamos para afirmar que a nossa classificação implementação funciona adequadamente:

```
testNearestRestaurantToHome public void () throws Exception {
    Ordenar Ordenar tipo = new (novo SortField ("localização",
        nova DistanceComparatorSource (0, 0)));

    Hits hits = searcher.search (espécie de consulta);

    assertEquals ("mais próximo",
                  . "El Charro", hits.doc (0) get ("nome"));
    assertEquals ("mais longe",
                  . "Los Betos", hits.doc (3) get ("nome"));

}
```

Home está em coordenadas (0,0). Nossa teste mostrou que o primeiro eo último documentos no Hits retornados são os mais próximos e mais distantes de casa. Muy bien! Se não tivéssemos usado uma espécie, os documentos teriam sido devolvidos em inserfim ção, uma vez que a pontuação de cada hit é equivalente para a consulta restaurante do tipo. O cálculo de distância, usando a fórmula de distância básica, é feita sob o nosso personalizado DistanceComparatorSource, Mostrada na listagem 6.2.

Listagem 6.2 DistanceComparatorSource

```

public class DistanceComparatorSource
    implements SortComparatorSource {
    privada x int;
    private int y;

    pública DistanceComparatorSource (int x, int y) {
        this.x = x;
        this.y = y;
    }

    newComparator ScoreDocComparator público (newComparator
        IndexReader leitor, String nome do campo) throws IOException {
        retorno DistanceScoreDocLookupComparator novo (
            leitor, nome do campo, x, y);
    }

    DistanceScoreDocLookupComparator classe private static
        implements ScoreDocComparator {
        private float [] distâncias; Conjunto de
            f
            distâncias
        DistanceScoreDocLookupComparator público (IndexReader leitor,
            Fieldname String, int x, int y) throws IOException {

            enumerador TermEnum final =
                reader.terms (novo Termo (nome do campo, ""));
            distâncias = new float [reader.maxDoc ()];
            if (distances.length> 0) {
                TermDocs termDocs reader.termDocs = ();
                try {
                    if (enumerator.term () == null) {
                        throw new RuntimeException (
                            "Não termos no campo" + nome do campo);
                }
                do {
                    Prazo prazo enumerator.term = ();
                    se quebrar (term.field () = nome do campo!);
                        termDocs.seek (enumerador);
                        while (termDocs.next ()) {
                            String [] xy = term.text (). Split ("");
                            int deltaX = Integer.parseInt (xy [0]) - x;
                            int deltax = Integer.parseInt (xy [1]) - y;
                            distâncias [termDocs.doc ()] = (float) Math.sqrt (
                                deltaX deltaX * + * deltax deltax);
                        }
                    } While (enumerator.next ());
                    Finally {}
                    termDocs.close ();
                }
            }
        }
    }

```

**b** Implementar SortComparatorSource

**c** Dê construtor localização base

**d**

**e** ScoreDocComparator

**f**

**g** Iterar condições

**h** Iterar documentos contendo mandato em curso

**Eu** Computar e armazenar distância

```

        }
    }

    compare int público (ScoreDoc i, j ScoreDoc) {
        if (distâncias [i.doc] <distâncias [j.doc]) return -1;
        if (distâncias [i.doc]> distâncias [j.doc]) return 1;
        return 0;
    }

    sortValue público comparável (ScoreDoc i) {
        retorno Float novo (distâncias [i.doc]);
    }

    sortType public int () {
        retorno SortField.FLOAT;
    }

}

public String toString () {
    "Distância (" return x + y +"," + +")";
}
}

```

- b Primeiro vamos implementar `SortComparatorSource`.
- c O construtor é entregue a localização base do qual os resultados são classificados por distância.
- d Isto é `SortComparatorSource`'S único método. Lucene se lida com o armazenamento em cache de `ScoreDocComparators`.
  - e Este é nosso costume `ScoreDocComparator` implementação.
  - f Aqui criamos um array de distâncias.
  - g Nós iterar sobre todos os termos no campo especificado.
  - h Em seguida, iterar sobre todos os documentos que contenham o termo atual.
  - i Nós calcular e armazenar a distância.
  - j O `comparar` método é usado pela API em busca de alto nível quando o real dis-  
cia não é necessário.
- Eu O `sortValue` método é usado pela API em busca de nível inferior, quando a dis-  
j valor de importância é desejada.

1)

A infra-estrutura de classificação dentro de caches Lucene (baseado em uma chave combinando o `hashcode` do `IndexReader`, O nome do campo, eo objeto de classificação personalizada) o resultado de `newComparator`. Nosso `DistanceScoreDocLookupComparator` imple-  
cação abre espaço para armazenar uma flutuar para cada documento no índice e calcula a distância do local de base para cada documento que contém o especificado

campo de classificação (`localização` no nosso exemplo). Em um índice homogêneo onde todos os documentos têm os mesmos campos, isso envolveria o cálculo da distância para cada documento. Dadas essas etapas, é imperativo que você está ciente dos recursos utilizados para classificar, este tema é discutido em maiores detalhes na seção 5.1.9, bem como em Javadocs do Lucene.

Ordenar por informações de tempo de execução, como a localização de um usuário é um powerfully incrivelmente característica erful. Neste ponto, porém, ainda temos uma peça que faltava: Qual é a distância de cada um dos restaurantes à nossa localização atual? Ao usar o `Hits`-retornando pesquisa métodos, não podemos chegar à distância calculada. No entanto, um API de baixo nível nos permite acessar os valores utilizados para a classificação.

### 6.1.1 Acessando os valores utilizados na classificação

#### personalizada

Além do `IndexSearcher.search` métodos que você viu até agora, alguns de menor nível são usados internamente para Lucene e não são tão útil para o exterior. A exceção entra com acesso a valores de classificação personalizada, como a distância de cada um dos restaurantes calculado pela nossa fonte comparador personalizado. O signature do método que usamos, em `IndexSearcher`, É

```
busca TopFieldDocs público (Query, Filtro de filtro,
nDocs final int, tipo Sort)
```

`TopFieldDocs` contém o número total de `Hits`, O `SortField` matriz utilizada para classificando, e uma série de `FieldDoc` objetos. A `FieldDoc` encapsula a prima calculado pontuação, ID documento, e uma série de Comparáveis com o valor utilizado para cada `SortField`. `TopFieldDocs` e `FieldDoc` são específicos para a pesquisa com um Espécie, Mas um sim-  
API de baixo nível ILAR existe quando a classificação não está sendo usado: Ele retorna `TopDocs` (Pai classe de `TopFieldDocs`) Contendo uma série de `ScoreDoc` (Classe pai de `FieldDoc`) objetos. Ao invés de nos preocuparmos com os detalhes da API, que você pode começar a partir Javadocs Lucene ou o código-fonte, vamos ver como realmente usá-lo.

Listagem 6.3's caso de teste demonstra o uso de `TopFieldDocs` e `FieldDoc` para recuperar a distância computada durante a classificação, esta classificação de tempo de trabalho em localização (10,10).

#### Listagem 6.3 Acessando personalizado valores de classificação para os resultados

```
testNearestRestaurantToWork public void () throws Exception {
    Ordenar Ordenar tipo = new (novo SortField ("localização", Especificar máximo
    nova DistanceComparatorSource (10, 10))); retornou hits

    TopFieldDocs docs = searcher.search (consulta, null, 3, tipo);
    assertEquals (4, docs.totalHits); b

    assertEquals (4, docs.totalHits); c Número total de hits
```

```
assertEquals (3, docs.Der retorno do número total d
de documentos

FieldDoc fieldDoc = (FieldDoc) docs.scoreDocs [0]; e Obter classificação
valores
assertEquals ("(10,10) -> (9,6) = sqrt (17)", f
nova Float (Math.sqrt (17)),
fieldDoc.fields [0]); Dê valor da primeira
f
computação
Documento documento = searcher.doc (fieldDoc.doc); g
assertEquals ("Los Betos", document.get ("nome"));
}
}■
```

- b Esta API de baixo nível exige que especificar o número máximo de visitas retornado.
- c O número total de visitas ainda é fornecido, porque todos os hits precisa ser determinado para encontrar os três melhores.
- d `docs.scoreDocs (0)` retorna um `ScoreDoc` e deve ser convertido para `FieldDoc` para obter `sort-`
- e valores ing.
- f O valor do primeiro (e único, neste exemplo) `SortField` computação é dis- capaz nos primeiros campos slot.
- g Obter o real `Documento` requer uma outra chamada.

g

Esta API de baixo nível exigido que especificar quantos resultados de pesquisa que desejar, que é diferente do `Hits` Retornando-métodos. Neste caso, limitando nossos resultados aos três restaurantes mais próximos é mais realista de qualquer jeito, porque nada mais longe não é o que os usuários querem.

O baixo nível `pesquisa` métodos não são úteis para os desenvolvedores, exceto, neste par- caso damente, de modo que não discuti-las em outro lugar. No entanto, este é atualmente o única forma de obter valores de classificação personalizada. Se você estiver em qualquer classificação do padrão `Sort-`

Campo opções, os valores estão disponíveis a partir `Hits` eo `Documento` se, portanto, use esta interface de nível inferior só nesse cenário de classificação personalizada.

## 6.2 Criar uma HitCollector personalizado

Na maioria das aplicações com pesquisa de texto completo, os usuários estão procurando o mais relevante documentos vant de uma consulta. O padrão de uso mais comum é tal que só os primeiros hits de pontuação mais alta são visitados. Em alguns cenários, porém, os usuários quero ser mostrado todos os documentos (por ID), que correspondem a uma consulta sem a necessidade de acessar o conteúdo do documento; filtros de pesquisa, discutido na seção 5.5, pode uso `HitCollector`s eficiente desta maneira. Outro uso possível, que nós

demonstrar nesta seção, é acessar o conteúdo a cada documento de uma pesquisa de forma direta.

Usando um `Hits-Voltar` pesquisa método irá trabalhar para recolher todos os documentos se você percorrer todos os hits e processá-los manualmente, mas você está incorrendo o esforço do mecanismo de cache dentro `Hits`. Usando um costume `HitCollector` evita a classe `Hits` coleção.

### 6.2.1 Sobre BookLinkCollector

Nós desenvolvemos um costume `HitCollector`, Chamado `BookLinkCollector`, Que constrói um mapa de todas as URLs únicas e os títulos dos livros correspondentes combinando uma consulta.

O `recolher (int, float)` método deve ser implementado a partir do `HitCollector` interface. `BookLinkCollector` é mostrado na listagem 6.4.

Listagem 6.4 Custom HitCollector: recolhe todos os hiperlinks livro

```
BookLinkCollector public class {HitCollector
    privada IndexSearcher pesquisador;
    documentos particulares = new HashMap HashMap ();

    pública BookLinkCollector (IndexSearcher pesquisador) {
        this.searcher = pesquisador;
    }

    public void recolher (int id pontuação float) {
        try {
            Documento doc = searcher.doc (id);
            documentos.put (doc.get ("url"), doc.get ("title"));
        } Catch (IOException e) {
            / / Ignorado
        }
    }

    pública Mapa getLinks () {
        retorno Collections.unmodifiableMap (documentos);
    }
}
```



Nosso coletor recolhe todos os títulos de livros (por URL) que correspondem a consulta.

### 6.2.2 Usando BookLinkCollector

Usando um `HitCollector` requer o uso de um uso de `IndexSearcher'S` pesquisa método variante como mostrado aqui:

```
testCollecting public void () throws Exception {
    Query = new TermQuery TermQuery (Termo de novo ("conteúdo", "junit"));
    IndexSearcher pesquisador = ();
    BookLinkCollector coletor = new BookLinkCollector (pesquisador);
    searcher.search (consulta, coletor);
    searcher.close ();

    Mapa linkMap collector.getLinks = ();
    assertEquals ("Java Development com Ant",
                 linkMap.get ("http://www.manning.com/antbook "));
}
```

Chamada `IndexSearcher.doc (n)` ou `IndexReader.document (n)` no `coletar` método pode retardar as pesquisas por uma ordem de magnitude, por isso não deixe a sua situação

requer acesso a todos os documentos. No nosso exemplo, temos a certeza que queremos o título e URL de cada documento correspondente. Parar um `HitCollector midstream` é um pouco de um corte, no entanto, porque não há nenhum mecanismo embutido para permitir isso. Para parar um `HitCollector`, Você deve lançar uma exceção de tempo de execução e estar preparado para pegá-lo onde você invocar `pesquisa`.

Filtros (ver ponto 5.5), como `QueryFilter`, Pode usar um `HitCollector` para definir bits em um `BitSet` quando os documentos são correspondidos, e não acessar o doc-base uments diretamente; este é um uso altamente eficiente da `HitCollector`.

A pontuação passado para o `coletar` método é a matéria-prima, pontuação, desnormalizada. Este podem diferir de `Hits.score (int)`, Que será normalizada para estar entre 0 e 1 se o documento com maior pontuação é maior que 1,0.

## Estendendo 6.3 QueryParser

---

Na seção 3.5, introduzimos `QueryParser` e mostrou que ele tem algumas configurações para controlar seu comportamento, como a definição do local para analisar e controlar a data o slop frase padrão. `QueryParser` é também extensível, permitindo que a subclassificação substituir partes do processo de consulta-criação. Nesta seção, demonstramos sub-classing `QueryParser` para não permitir consultas ineficientes curinga e fuzzy, personalizado intervalo de datas, manipulação e consultas morphing frase em `SpanNearQuerys` em vez de `PhraseQuerys`.

### 6.3.1 Personalizando o comportamento do QueryParser

Embora `QueryParser` tem algumas peculiaridades, tais como as interações com um analisador, ele não tem pontos de extensibilidade que permitem a personalização. Tabela 6.1 detalha as métodos concebidos para substituir e porque você pode querer fazê-lo.

Tabela 6.1 QueryParser's pontos de extensibilidade

Método	Por que substituir?
getFieldQuery (campo String, Analisador analisador, String queryText)  ou  getFieldQuery (campo String, Analisador analisador, String queryText, slop int)	Estes métodos são responsáveis pela construção de qualquer uma <code>TermQuery</code> ou um <code>PhraseQuery</code> . Se a análise especial é necessário, ou um único tipo de consulta é desejado, substituir esse método. Por exemplo, um <code>SpanNearQuery</code> pode substituir <code>PhraseQuery</code> para forçar ordenou correspondências de frase.
getFuzzyQuery (campo String, String termStr)	Consultas fuzzy pode afetar negativamente o desempenho. Substituir e jogar um <code>ParseException</code> para não permitir consultas fuzzy.
getPrefixQuery (campo String, String termStr)	Este método é usado para construir uma consulta quando o prazo termina com uma asterisco. A seqüência de termo entregue a este método não inclui o trailing asterisco e não é analisada. Substituir esse método para executar qualquer análise desejada.
getRangeQuery campo String (, Analisador analisador, começar a String seção 3.5.5). Imperativas não poderiam: Final String, inclusive boolean)	Comportamento padrão ampla consulta tem várias peculiaridades observado (ver <ul style="list-style-type: none"> <li>▪ Minúsculas os termos de início e fim</li> <li>▪ Usar um formato de data diferente</li> <li>▪ Lidar com séries de números de padding para combinar como os números foram indexados</li> </ul>
getWildcardQuery (campo String, String termStr)	Consultas curinga podem afetar negativamente o desempenho, então substituído métodos poderiam atirar uma <code>ParseException</code> para não permitir-los. Alternativamente, uma vez que a string termo não é analisado, tratamento especial pode ser desejado.

Todos os métodos listados retornar um `Pergunta`, tornando possível a construção de algumas outra coisa do que o tipo subclasse atual usado pelas implementações original desses métodos. Além disso, cada um desses métodos podem jogar um `ParseException` permitindo a manipulação de erro.

### 6.3.2 Proibir consultas fuzzy e curinga

A subclasse personalizada na listagem 6,5 demonstra uma subclasse personalizada analisador de consulta que desativa consultas fuzzy e curinga, aproveitando o `Analisa-Exceção` opção.

Listagem 6,5 Não permitir consultas curinga e fuzzy

```
CustomQueryParser public class {QueryParser
    CustomQueryParser pública (campo String, analisador Analyzer) {
```

```

        super (campo, analisador);
    }

getWildcardQuery Query protegidos final (
    Campo String, String termStr) throws ParseException {
    throw new ParseException ("Wildcard não permitidos");
}

getFuzzyQuery Query protegidos final (
    Campo String, String termStr) throws ParseException {
    throw new ParseException ("Fuzzy consultas não permitidos");
}

}

```

Para utilizar este parser personalizados e impedir que os usuários executar curinga e fuzzy consultas, construir uma instância de `CustomQueryParser` e usá-lo exatamente como você seria `QueryParser`, Como mostrado no código a seguir. Tenha cuidado para não chamar a estático `analisar` método que usa o built-in `QueryParser` comportamento:

```

testCustomQueryParser public void () {
    Parser CustomQueryParser =
        nova CustomQueryParser ("campo", analisador);
    try {
        parser.parse ("um t?");
        falha ("queries Wildcard não deve ser permitido");
    } Catch (ParseException esperado) {
        / / Espera
        assertTrue (true);
    }

    try {
        parser.parse ("xunit ~");
        falha ("Fuzzy consultas não deve ser permitido");
    } Catch (ParseException esperado) {
        / / Espera
        assertTrue (true);
    }
}

```

Com essa implementação, esses dois tipos de consulta caras são proibidos, dando-lhe alguma paz de espírito em termos de desempenho e erros que podem surgem essas consultas se expandindo em muitos termos.

### 6.3.3 Manuseio campo numérico gama-consultas

Lucene é tudo sobre como lidar com o texto. Você já viu em vários lugares como as datas podem ser tratado, o que equivale a sua conversão em uma representação de texto

que podem ser ordenadas alfabeticamente. Manipulação de números é basicamente o mesmo, exceto implementação de uma conversão para um formato de texto é deixado para você.

Nesta seção, nossos índices cenário de exemplo um inteiro `id` campo para que alcance consultas podem ser realizadas. Se indexados `toString` representações do integers 1 a 10, a ordem no índice seria 1, 10, 2, 3, 4, 5, 6, 7, 8, 9 - não a ordem pretendida em tudo. No entanto, se pad os números com zeros à esquerda para que todos os números têm a mesma largura, a ordem é correto: 01, 02, 03, e assim em. Você terá que decidir sobre a largura máxima seus números precisam, optamos 10 dígitos e implementadas as seguintes `pad (int)` método de utilitário: 3

```

pública NumberUtils classe A M
    formatador DecimalFormat private static final =
        nova DecimalFormat ("0000000000");
    }

    pad public static String (int n) F L
    {
        retorno formatter.format (n)
    }

}

Os números precisam ser preenchidos durante a indexação. Isso é feito em nosso testesetUp
()
método no id campo palavra-chave:
public class AdvancedQueryParserTest estende TestCase {
    analisador Analyzer privado;
    RAMDirectory diretório privado;

    setUp protected void () throws Exception {
        super.setUp ();
        analisador = new WhitespaceAnalyzer ();

        diretório = new RAMDirectory ();
        Escritor IndexWriter IndexWriter = new (diretório, analisador,
            true);
        for (int i = 1; i <= 500; i + +) {
            Documento doc = new Document ();
            doc.add (Field.Keyword ("id", NumberUtils.pad (i)));
            writer.addDocument (doc);
        }
        writer.Close ();
    }

}

```

Com este tempo de preenchimento do índice, estamos apenas no meio do caminho. Uma expressão de consulta para IDs 37 a 346 formulada como `id: [37 a 346]` não vai funcionar como esperado com o

<sup>3</sup> Lojas Lucene termo informação com compactação de prefixo de modo que nenhuma penalidade é pago por grandes compartilhada prefixos como este preenchimento zero.

padrão RangeQuery criado por QueryParser. Os valores são tomadas literalmente e não são acolchoadas como eram quando indexados. Felizmente, podemos corrigir este problema em

nosso CustomQueryParser , substituindo o getRangeQuery () método:

```
getRangeQuery Query protegido (campo String, Analyzer analisador,
                String part1, part2 String,
                inclusive boolean)
    throws ParseException {
    if ("id". equals (campo)) {
        try {
            int num1 = Integer.parseInt (part1);
            int num2 = Integer.parseInt (part2);
            retorno RangeQuery novo (
                novo Termo (campo, NumberUtils.pad (num1)),
                novo Termo (campo, NumberUtils.pad (num2)),
                inclusive);
            } Catch (NumberFormatException e) {
                throw new ParseException (e.getMessage ());
            }
        }
    }

    super.getRangeQuery retorno (campo, analisador, part1, part2,
                                inclusive);
}
```

Esta aplicação é específica para o nosso id campo, você pode querer generalizar-lo para mais campos. Se o campo não é id, Delega para o comportamento padrão. O id campo recebe tratamento especial, ea função pad é chamado apenas como com indexação. Seguindo caso de teste mostra que a consulta ampla funcionou conforme o esperado, e você pode ver

os resultados do preenchimento usando Pergunta'S toString (String) método:

```
testIdRangeQuery public void () throws Exception {
    Parser CustomQueryParser =
        nova CustomQueryParser ("campo", analisador);

    Query = parser.parse ("id: [37 TO 346]");

    assertEquals ("acolchado", "id: [TO 000000037 000000346]",
                  query.toString ("campo"));

    IndexSearcher searcher = new IndexSearcher (diretório);
    Hits hits = searcher.search (query);

    assertEquals (310, hits.length ());
}
```

Nosso teste mostra que temos conseguido permitindo sensata de aparência introduzidos pelo utilizador consultas de intervalo para funcionar como esperado.

### 6.3.4 Permitir que ordenou consultas frase

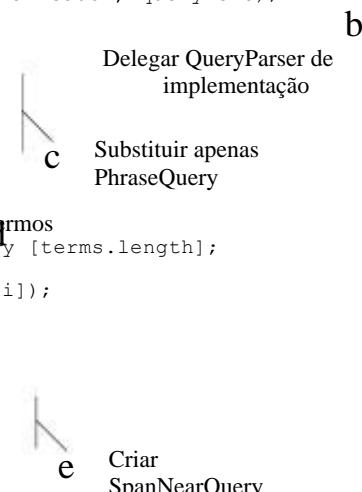
Quando `QueryParser` analisa um único termo, ou termos dentro de aspas duplas, ele DELE-  
portas a construção do Pergunta a um `getFieldQuery` método. A análise em um  
prazo não cotadas chama a `getFieldQuery` método sem a assinatura slop (slop  
só faz sentido na frase MultiTerm consulta); análise de uma frase citada chama a  
`getFieldQuery` assinatura com o fator de slop, que internamente delegados para a  
assinatura nonslop para construir a consulta e, em seguida, define o slop adequadamente. O  
Pergunta devolvidos ou é um `TermQuery` ou um `PhraseQuery`, Por padrão, dependendo  
se um ou mais tokens são retornados do analyzer.<sup>4</sup> Dado o suficiente,  
`PhraseQuery` irá corresponder termos fora de ordem no texto original. Não há maneira de  
uma força `PhraseQuery` para corresponder em ordem (exceto com slop de 0 ou 1). No entanto,  
`SpanNearQuery` não permite, a fim de correspondência. Uma substituição simples de `get-  
FieldQuery` nos permite substituir um `PhraseQuery` com um ordenado `SpanNearQuery`:

```
getFieldQuery Query protegidas (
    String campo, Analyzer analisador, String queryText, slop int)
    throws ParseException {
    Consulta orig = super.getFieldQuery (campo, analisador, queryText);

    if (! (orig PhraseQuery instanceof)) {
        retorno orig;
    }

    PhraseQuery pq = (PhraseQuery) orig;
    Prazo [] = termos pq.getTerms ();Puxe todos os termos
    SpanTermQuery [] = new cláusulas SpanTermQuery [terms.length];
    for (int i = 0; i terms.length <; i + +) {
        cláusulas [i] = new SpanTermQuery (termos [i]);
    }

    Query = new SpanNearQuery SpanNearQuery (
        cláusulas, slop, true);
    retorno da consulta;
}
```



- b** Nós delegar `QueryParser` implementação "s para análise e determinação de tipo de consulta.
- c** Aqui nós substituir `PhraseQuery` e retornar qualquer outra coisa imediatamente.
- d** Puxamos todos os termos do original `PhraseQuery`.
- e** Finalmente, criamos uma `SpanNearQuery` com todos os termos do original `PhraseQuery`.

<sup>4</sup> A `PhraseQuery` poderia ser criado a partir de um único termo se o analisador criado mais de um token para ele.

Nosso caso de teste mostra que nosso costume `getFieldQuery` é eficaz na criação de uma `SpanNearQuery`:

```
testPhraseQuery public void () throws Exception {
    Parser CustomQueryParser =
        nova CustomQueryParser ("campo", analisador);

    Query = parser.parse ("singleTerm");
    assertTrue ("TermQuery", consulta instanceof TermQuery);

    query = parser.parse ("\", uma frase \\"");
    assertTrue ("SpanNearQuery", consulta instanceof SpanNearQuery);
}
```

Outra melhoria possível seria adicionar um interruptor para a consulta personalizada analisador, permitindo a bandeira em ordem a ser controlado pelo usuário da API.

## 6.4 Usando um filtro personalizado

---

Se todas as informações necessárias para realizar a filtragem está no índice, não há necessidade de escrever seu próprio filtro, pois o `QueryFilter` pode lidar com isso. No entanto, existem boas razões para fator de informações externas em um filtro personalizado. Utilizando o nosso livro de dados de exemplo e fingindo que estamos executando uma livraria on-line, nós deseja que os usuários possam pesquisar dentro do nosso ofertas especiais quentes do dia. Um opção é armazenar a bandeira especiais em um campo de índice. No entanto, a mudança `specials`

frequênciça. Em vez de reindexar documentos quando `specials` mudança, optamos por manter os especiais sinalizados na nossa base de dados (hipotético) relacional.

Para fazer isso direito, nós queremos que seja dirigido por testes e demonstrar como a nossa `SpecialsFilter` pode puxar informações de uma fonte externa, mesmo sem ter um extero-fonte na! Usando uma interface, um objeto fictício, e bom ol 'JUnit, aqui vamos nós. Primeiro, aqui está a interface para recuperar especiais:

```
SpecialsAccessor interface pública {
    String [] ISBN ();
}
```

Uma vez que não teremos uma enorme quantidade de especialidades ao mesmo tempo, retornando todos os o ISBNs dos livros em especial será suficiente.

Agora que temos uma interface de recuperação, podemos escrever o nosso filtro personalizado, `SpecialsFilter`. Filtros se estendem desde o `org.jakarta.lucene.search.Filter` classe e deve implementar a `bits (IndexReader leitor)` método, retornando um `BitSet`. Bocado posições correspondem aos números de documentos. Bits habilitado, o documento de que a posição está disponível para ser pesquisada contra a consulta, e bits unset significar a

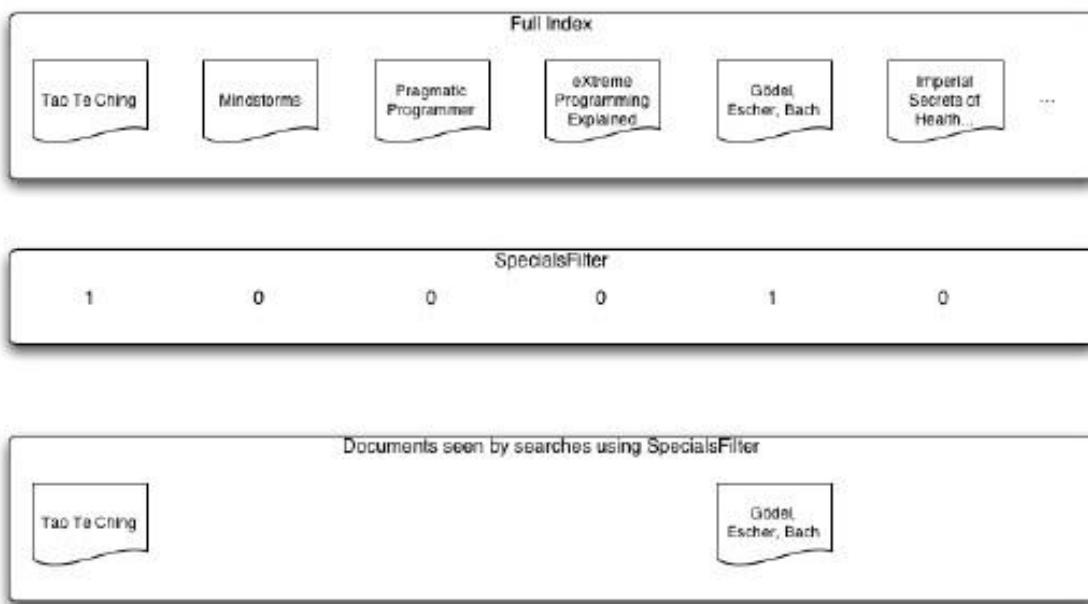


Figura 6.2 Filtragem para livros em especial

documento não será considerado na pesquisa. Figura 6.2 ilustra um exemplo `SpecialsFilter` que define bits para livros em especial (ver listagem 6.6).

**Listagem 6.6** `SpecialsFilter`: um filtro personalizado que recupera informações de um fonte externa

```
public class SpecialsFilter estende Filter {
    assessor SpecialsAccessor privado;

    pública SpecialsFilter (assessor SpecialsAccessor) {
        this.accessor = assessor;
    }

    bits pública BitSet (IndexReader leitor) throws IOException {
        BitSet bits = new BitSet (reader.maxDoc ());

        String [] = ISBNs accessor.isbn ();
        int [] docs = new int [1];
        int [] freqs [1];

        for (int i = 0; i < isbn.length; i++) {
            String isbn = ISBN [i];
            if (isbn != null) {
                TermDocs termDocs =
                    reader.termDocs (Termo de novo ("isbn", isbn));

```

**b** Buscar ISBNs

**c** Ir para prazo

```

        int count = termDocs.read (docs, freqs);
        if (count == 1) {
            bits.set (docs [0]);
        } Conjunto de bits correspondente
    }
}

returnar bits;
}
}

```

- b Aqui, nós buscar o ISBNs dos especiais que deseja habilitar para a pesquisa.
- c isbn está indexado como um Palavra chave campo e é único, por isso usamos IndexReader para saltar diretamente para o termo.
- d Com o documento correspondente foi encontrado, montamos a sua parte correspondente.

Retornando nulo a partir da `bits ()` método é o mesmo que acender todos os bits. Todos documentos serão considerados, como se a consulta havia sido feito sem o uso de um filtro.

Para testar se o nosso filtro está funcionando, nós criamos um simples `MockSpecialsAccessor` para retornar um conjunto específico de ISBNs, dando nosso controle caso de teste sobre o conjunto de especiais:

```

MockSpecialsAccessor public class {SpecialsAccessor
private String [] ISBNs;

MockSpecialsAccessor público (String [] ISBNs) {
    this.ISBNs = ISBNs;
}

public String [] ISBN () {
    ISBN retorno;
}
}

```

Veja como testamos nossos `SpecialsFilter`, Usando a mesma `setUp ()` que o fil-outras testes de ter usado:

```

testCustomFilter public void () throws Exception {
    String [] ISBN = new String [] {"0060812451"};

    SpecialsAccessor assessor = nova MockSpecialsAccessor (ISBN);
    Filtro filtro = new SpecialsFilter (assessor);
    Hits hits = searcher.search (allBooks, filtro);
    assertEquals ("os especiais", ISBNs.length, hits.length ());

}

```

Usamos uma consulta genérica que é amplo o suficiente para recuperar todos os livros, fazendo afirmações mais fácil de ofício, mas porque o nosso filtro aparado o espaço de busca, apenas o especiais são retornados. Com esta infra-estrutura no local, implementando um `Specials-Accessor` para recuperar uma lista de ISBNs a partir de um banco de dados deve ser fácil, pois isso é deixado

como um exercício para o leitor mais experiente.

Note que tomou uma decisão importante a implementação não para o cache `Bits` Conjunto em `SpecialsFilter`. Decoração `SpecialsFilter` com um `CachingWrapperFilter` nos liberta esse aspecto.

#### 6.4.1 Usando uma consulta filtrada

Para adicionar ao filtro sobrecarga de terminologia, uma última opção é nova no Lucene 1.4: `FilteredQuery`<sup>5</sup> `FilteredQuery` inverte a situação que a pesquisa com um Filtro apresenta. Usando um Filtro um `IndexSearcher`'S pesquisa método aplica um fil-single ter antes da consulta. Usando o novo `FilteredQuery`No entanto, você pode aplicar um Filtro a uma cláusula de consulta particular de um `BooleanQuery`.

Vamos pegar o `SpecialsFilter` como exemplo novamente. Desta vez, queremos um mais consulta sofisticada: livros em uma categoria de educação em especial, ou livros sobre Logo.<sup>6</sup> Nós não poderíamos fazer isso com uma consulta direta utilizando as técnicas mostradas, assim, longe, mas `FilteredQuery` torna isso possível. Tinha sido a nossa busca só para livros em a categoria da educação em especial, poderíamos ter usado a técnica mostrada no trecho de código anterior, ao invés.

Nosso caso de teste, na listagem 6.7, demonstra a consulta descrita usando uma `Boolean-Pergunta` com uma nested `TermQuery` e `FilteredQuery`.

#### Listagem 6.7 Usando um `FilteredQuery`

```
testFilteredQuery public void () throws Exception {
    String [] ISBN = new String [] {"0854402624"};
    SpecialsAccessor assessor MockSpecialsAccessor = new (ISBN);
    Filtro filtro = new SpecialsFilter (assessor);
```

b Livro de Rudolf Steiner

```
WildcardQuery educationBooks =
    novo WildcardQuery (novo Termo ("categoria", "educação * *"));
FilteredQuery edBooksOnSpecial =Toda a educação
    nova FilteredQuery (educationBooks, filtro); livros sobre
```

c

especial

<sup>5</sup> Desculpe! Nós sabemos que `Filtro`, `QueryFilter`, `FilteredQuery`, E completamente alheios `Token-Filtro` nomes pode ser confuso.

<sup>6</sup> Erik começou suas aventuras programação com Logo em um Apple] [e. Os tempos não mudaram muito; agora, ele mexe com StarLogo em um PowerBook.

```

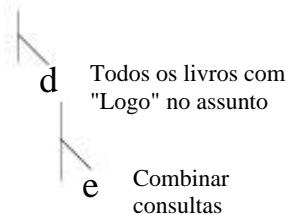
TermQuery logoBooks =
    nova TermQuery (novo Termo ("subject", "logo"));

BooleanQuery logoOrEdBooks BooleanQuery = new ();
logoOrEdBooks.add (logoBooks, false, false);
logoOrEdBooks.add (edBooksOnSpecial, false, false);

Hits hits = searcher.search (logoOrEdBooks);
System.out.println (logoOrEdBooks.toString ());
assertEquals ("Papert e Steiner", 2, hits.length ());

}

```



- b Este é o número ISBN para Rudolf Steiner A Arte Moderna de Educação.
  - c Nós construímos uma consulta de livros sobre educação especial, que inclui apenas Steiner livro neste exemplo.
  - d Nós construímos uma consulta para todos os livros com logotipo no assunto, que inclui apenas Mindstorms em dados de nossa amostra.
  - e As duas consultas são combinados de forma OR.
- O `bits ()` método de nested Filtro é chamada cada vez que um `FilteredQuery` é usado em uma pesquisa, por isso recomendamos que você use um filtro de cache se a consulta é ser usado repetidamente e os resultados de um filtro não mudam.

## 6,5 teste de desempenho

Lucene é rápido e escalável. Mas quão rápido é? É rápido o suficiente? Você pode garantir que as buscas são devolvidos dentro de um período de tempo razoável? Como é que Lucene responder sob carga?

Se o projeto tiver demandas de alto desempenho, você fez a coisa certa ao escolher Lucene, mas não deixe que os números de desempenho ser um mistério. Há SEV-desempenho maneiras gerais Lucene pode ser impactado negativamente pela forma como você usá-lo-

como o uso de consultas fuzzy ou curinga ou uma consulta ampla, como você verá nesta seção.

Nós estivemos destacando o teste de unidade ao longo do livro usando os princípios de JUnit. Nesta seção, utilizamos outra jóia testes de unidade, JUnitPerf. JUnitPerf, um Decorador JUnit, permite testes JUnit devem ser medidos para carga e velocidade.

### 6.5.1 Teste a velocidade de uma pesquisa

Nós discutimos como `FuzzyQuery` e `WildcardQuery` têm o potencial para sair de controle. De forma semelhante, `RangeQuery` pode, também: Como ele enumera todas as termos na faixa, ele forma uma `BooleanQuery` que potencialmente pode ser grande.

O Mike infame "viciados na barra verde" Clark tem graciosamente doados alguns testes de desempenho Lucene para nós.<sup>7</sup> Vamos examinar um exemplo concreto no que determine que um problema de desempenho busca é causado pelo modo como nós índice, e descobrir como podemos facilmente corrigir esse problema. Contamos com JUnitPerf para identificar a questão e garantir que é fixado e permanece assim.

Estamos indexação de documentos que têm um timestamp da última modificação. Por exemplo fins, índice que uma amostra de 1.000 documentos fabricados com timestamps aumentando em incrementos de 1 segundo, a partir de ontem:

```
Calendário timestamp = GregorianCalendar.getInstance ();
timestamp.set (Calendar.DATE,
    timestamp.get (Calendar.DATE) - 1); Ontem           b

for (int i = 0; i < tamanho; i + +) {
    timestamp.set (Calendar.SECOND,
        timestamp.get (Calendar.SECOND) + 1); Aumentar um segundo
    Data agora = timestamp.getTime ();                  c
    Documento documento = new Document ();
    documento.add (Field.Keyword ("última modificação", agora)); Como Data
    writer.addDocument (documento);
}
d
```

Sendo os codificadores test-infectados somos, até mesmo garantir que nossa busca está retornando os resultados esperados através de pesquisa em uma faixa de timestamp que engloba todas as documentos indexados:

```
testSearchByTimestamp public void () throws Exception {
    Busca s = Search new ();
    Hits hits s.searchByTimestamp = (janOneTimestamp,
        todayTimestamp);
    assertEquals (1000 hits.length, ());
}
```

**searchByTimestamp executa uma RangeQuery: 8**

```
searchByTimestamp Hits público (data de início, data final)
    throws Exception {
    Prazo beginTerm Term = new ("última modificação",
        DateField.dateToString (início));
    Prazo endTerm Term = new ("última modificação",
        DateField.dateToString (final));
```

<sup>7</sup> Mike é o co-autor de EJB amargo (Manning) e autor de Automação pragmática (Livro-Pragmatic prateleira); <http://www.clarkware.com>.

<sup>8</sup> Estamos intencionalmente ignorando bits de infra-estrutura de Mike teste para manter nossa discussão centrou-se na desempenho dos ensaios de aspecto, em vez de se atolar seguir seu código muito bem dissociado. Veja o "Sobre este livro", secção no início do livro para obter mais detalhes sobre como obter o código fonte completo.

```
Query = new RangeQuery (beginTerm, endTerm, true);

    retorno newSearcher (
        index.byTimestampIndexDirName ()) pesquisa (query).;
}

}
```

Neste ponto, tudo está bem. Nós indexados 1.000 documentos e descobriu que todos usando uma data abrangente `RangeQuery`. Enviá-lo! Whoa ... não tão rápido ... o que se tinha 2.000 documentos indexados? Aqui está o que acontece quando corremos o -teste `SearchByTimestamp ()` método de teste:

```
org.apache.lucene.search.BooleanQuery $ TooManyClauses
    em org.apache.lucene.search.BooleanQuery.add (BooleanQuery.java: 109)
        em org.apache.lucene.search.BooleanQuery.add (BooleanQuery.java: 101)
        em org.apache.lucene.search.RangeQuery.rewrite (RangeQuery.java: 137)
        em
    org.apache.lucene.search.IndexSearcher.rewrite (IndexSearcher.java: 227)
        em org.apache.lucene.search.Query.weight (Query.java: 84)
        em
    org.apache.lucene.search.IndexSearcher.search (IndexSearcher.java: 129)
        em org.apache.lucene.search.Hits.getMoreDocs (Hits.java: 102)
        . org.apache.lucene.search.Hits em <init> (Hits.java: 81)
        em org.apache.lucene.search.Searcher.search (Searcher.java: 71)
        em org.apache.lucene.search.Searcher.search (Searcher.java: 65)
        em lia.advsearching.perf.Search.searchByTimestamp (Search.java: 40)
        em
    lia.advsearching.perf.SearchTest.testSearchByTimestamp (SearchTest.java: 24)
```

Nosso conjunto de dados é apenas 2.000 documentos, que não é problema para lidar com Lucene.

Mas uma `RangeQuery` internamente reescreve-se a um `BooleanQuery` com um nonrequired cláusula para cada termo no intervalo. Isto é, com 2.000 documentos foram indexados, o `searchByTimestamp ()` método fará com que 2000 OR'd `TermQuery's` aninhados em um `BooleanQuery`. Isso excede o limite padrão de 1024 cláusulas a um `BooleanQuery`, que impede consultas de se deixar levar.

### Modificar o índice

Para fins de pesquisa, porém, o objetivo é ser capaz de pesquisar por intervalo de datas. É improvável que precisaremos para procurar documentos em uma escala de segundos, portanto, usando esse timestamp de granulação fina não é necessária. De fato, é problemático. Indexação 1.000 ou 2.000 documentos em sucessivos incrementos timestamp dá a cada segundo documento um termo completamente original, tudo dentro do espaço de menos de uma hora de no valor de timestamps.

Uma vez que a busca por dia, não segundo, é o objetivo real, vamos índice dos documentos por dia, em vez:

```
String = Search.today hoje ();

for (int i = 0; i < tamanho; i++) {
    Documento documento = new Documento ();
    documento.add(Field.Keyword ("última modificação", hoje));
    writer.addDocument (documento);
}
```

Aqui, `hoje` está definido para o formato YYYYMMDD. Lembre-se, os termos são classificados automaticamente, assim que números precisam levar isso em conta (ver secção 6.3.3 para um número

**padding exemplo:**

```
public static String hoje () {
    SimpleDateFormat dateFormat =
        (SimpleDateFormat) SimpleDateFormat.getDateInstance ();
    dateFormat.applyPattern ("AAAAMMDD");
    retorno dateFormat.format (todayTimestamp ());
}
```

Note que estamos usando um `Corda` valor para `hoje` (Como 20040715) ao invés de usando o `DateField.dateToString ()` método. Independentemente de índice que pelo timestamp ou pelo formato AAAAMMDD, os documentos todos têm o mesmo ano, mês, dia e, por isso na nossa segunda tentativa de indexação de um campo da última modificação, há

apenas um único termo no índice, e não milhares. Esta é uma melhoria dramática que é facilmente visto em testes JUnitPerf. Você pode certamente manter um campo timestamp no documento, também, ele só não deve ser um campo usado em consultas alcance.

### Testar o índice baseado no timestamp

Listagem 6.8 é um JUnitPerf `TimedTest`, O teste que o nosso original 1000 documentos são encontrados em 100 milissegundos ou menos.

#### Listagem 6.8 JUnitPerf decorados teste cronometrado

```
public class SearchTimedTest {

    suite Test public static () {
        int maxTimeInMillis = 100;

        Teste teste = new SearchTest ("testSearchByTimestamp");
        TestSuite suite = new TestSuite ();

        suite.addTest (teste); Teste W
        suite.addTest (novo TimedTest (teste, maxTimeInMillis));

        retorno suite;
    }
}
```

**C** Wrap em teste  
**TimedTest**

- b Nós executado pela primeira vez um teste para aquecer a JVM antes de timing.
- c Então, nós envolvemos o teste simples dentro de um `TimedTest`, Afirmando que ele seja executado em 100 milissegundos ou menos.

Este testar falhará porque ela excede a restrição de 100 milissegundos:

```
[Junit] TestCase: testSearchByTimestamp (lia.advsearching.perf.SearchTest):  
FAILED  
[Junit] Tempo máximo decorrido ultrapassado! Espera 100ms, 138ms, mas foi.  
[Junit] junit.framework.AssertionFailedError: tempo decorrido máxima  
ultrapassado! Espera 100ms, 138ms, mas foi.  
[Junit] no  
com.clarkware.junitperf.TimedTest.runUntilTestCompletion (Unknown Source)  
[Junit] em com.clarkware.junitperf.TimedTest.run (Unknown Source)
```

O teste falhou, mas não por muito. Claro que, quando 2.000 documentos são tentados ele falhar horrivelmente com um `TooManyClauses` exceção.

#### Testar o índice de data-base

Agora vamos escrever um teste de unidade que usa o intervalo `YYYYMMDD`:

```
testSearchByDay public void () throws Exception {  
    Busca s = Search new ();  
    Hits hits = s.searchByDay ("20040101", hoje);  
    assertEquals (1000 hits.length,());  
}
```

O valor de `hoje` em `testSearchByDay ()` é a data atual em `YYYYMMDD` formato. Agora vamos substituir uma linha em `SearchTimedTest` com um `testSearchByDay ()`:

```
Teste teste = new SearchTest ("testSearchByDay");
```

Nosso `SearchTimedTest` passa agora com cores de vôo (ver figura 6.3 para horários de `SearchTest` sob carga).

#### 6.5.2 O teste de carga

Não só pode JUnitPerf decorar um teste e afirmar que ele executa em um tolerados quantidade de tempo, ele também pode executar testes de carga através da simulação de uma série de concorrentes de usuários. O padrão de decorador mesmo é usado como um `TimedTest`. Decorando um `TimedTest` com um `LoadTest` é o uso geral, como mostrado na listagem 6.9.

##### Listagem teste de carga 6.9

```
public class SearchLoadTest  
  
suite Test public static () {  
  
    int maxTimeInMillis = 100;
```

```

        int concurrentUsers = 10;

        Teste teste = new SearchTest ("testSearchByDay");

        Wrap teste básicoTestSuite suite = new TestSuite ();
        TimedTest com
        suite.addTest (teste);
        TimedTest teste = new TimedTest (teste, maxTimeInMillis);
        LoadTest LoadTest LoadTest = new (timedTest, concurrentUsers);
        b
        suite.addTest (LoadTest);
        c

        retorno suite;
    }
}
    
```

Wrap TimedTest  
LoadTest em

- b Nós envolvemos o teste básico (garantindo que 1000 hits são encontrados) com uma TimedTest.
- c Então embrulhar o TimedTest em um LoadTest, Que executa o TimedTest 10 vezes simultaneamente.

SearchLoadTest **executa** testSearchByDay () 10 vezes simultaneamente, com cada thread necessário para executar em menos de 100 milissegundos. Não deve ser nenhuma surpresa

que a mudança do SearchLoadTest para executar SearchTest.testSearchByTimestamp () causa uma falha, uma vez que falha até mesmo o SearchTimedTest. Os horários de cada SearchTest, Executado como 10 testes simultâneos, são mostradas na figura 6.3.

Os resultados indicam que cada teste realizado bem abaixo do 100 milissegundos exigência, mesmo correndo sob carga simultâneas.

### 6.5.3 QueryParser novamente!

QueryParser eleva sua cabeça feia de novo com o nosso formato de data alterada. A built-in intervalo de datas de manuseio analisa DateFormat.SHORT formatos para a DateField texto conversões. Seria bom para permitir aos usuários inserir um formato de data típicos como 1/1/04 e convertê-lo para o nosso formato de revista da data YYYYMMDD. Isto pode ser

Test	Time elapsed
<b>Total:</b>	<b>0.154 s</b>
testSearchByDay	0.016 s
testSearchByDay	0.009 s
testSearchByDay	0.006 s
testSearchByDay	0.005 s
testSearchByDay	0.006 s
testSearchByDay	0.007 s
testSearchByDay	0.069 s
testSearchByDay	0.005 s
testSearchByDay	0.009 s
testSearchByDay	0.022 s

Figura 6.3  
Desempenho resultados do teste por 10 concorrentes SearchTests, cada um necessário para concluir em 100 milissegundos ou menos

feito de uma forma similar ao que fizemos na seção 6.3.3 para inteiros almofada para consultas de intervalo. O efeito desejado é mostrada no seguinte teste:

```
testQueryParsing public void () throws Exception {
    Parser SmartDayQueryParser =
        SmartDayQueryParser nova ("conteúdo",
            nova StandardAnalyzer ());
    Query =
        parser.parse ("da última modificação: [1/1/04 TO 2/29/04]");
    assertEquals ("da última modificação: [20040101 TO 20040229]",
        query.toString ("Conteúdo"));
}
```

Agora que temos o nosso efeito desejado codificado como um caso de teste, vamos fazê-lo passar por bacalhau  
ing SmartDayQueryParser.

#### SmartDayQueryParser compreensão

O SmartDayQueryParser é uma simples adaptação do built-in QueryParser's getRangeQuery método:

```
SmartDayQueryParser public class {QueryParser
formatador DateFormat public static final =
new SimpleDateFormat ("AAAAMMDD");

SmartDayQueryParser pública (campo String, analisador Analyzer) {
    super (campo, analisador);
}

getRangeQuery Query protegido (campo String, Analyzer analisador,
String part1, part2 String,
inclusive boolean)
throws ParseException {

try {
    DateFormat df =
        DateFormat.getDateInstance (DateFormat.SHORT,
            GetLocale ());
    df.setLenient (true);
    D1 = Data df.parse (part1);
    D2 = Data df.parse (part2);
    part1 = formatter.format (d1);
    part2 = formatter.format (d2);
} Catch (Exception ignorada) {

retorno RangeQuery novo (novo Termo (campo, part1),
novo Termo (campo, part2),
```

```
inclusive);  
}  
}
```

A única diferença entre a nossa substituído `getRangeQuery` eo original implementação é o uso de formatação YYYYMMDD.

#### 6.5.4 Morals de testes de desempenho

Além de testar se Lucene podem desempenho aceitável com o ambiente e de dados, unidade de teste de desempenho assistências (como faz o teste JUnit básico)

no projeto de seu código. Neste caso, você viu como o nosso método original de datas de indexação foi menos do que desejável, embora o nosso primeiro teste de unidade conseguiu

com o número certo de resultados. Apenas quando testamos com mais dados ou com limitações de tempo e de carga que um problema se apresentar. Nós poderíamos ter varrido o falta de dados para debaixo do tapete temporariamente através da criação BooleanQuery's

`setMaxClauseCount (int) para Integer.MAX_VALUE`. No entanto, não seria capaz de esconder um por Falha no teste de desempenho.

Nós encorajamos você a adotar o teste de unidade em seus projetos e continuará a evoluir a base de código de teste em testes de unidade de desempenho. Como você pode

dizer a partir de exemplos de código neste livro, estamos muito centrados em teste, e nós também

usar os testes para fins de aprendizagem, explorando APIs. Lucene em si é construída em torno de um

forte conjunto de testes de unidade, que são executados automaticamente a cada build.

Teste de unidade pragmática por Dave Thomas e Andy Hunt é um conciso e introdução elegante para testes de unidade. Manning JUnit em Ação é Fantástic, que vai além do básico e investiga temas como o desempenho testes e testes unitários mock, sendo que ambos temos incorporado código dos livros fonte.

## 6.6 Resumo

Lucene oferece aos desenvolvedores a flexibilidade extrema na busca de recursos.

Personalizado

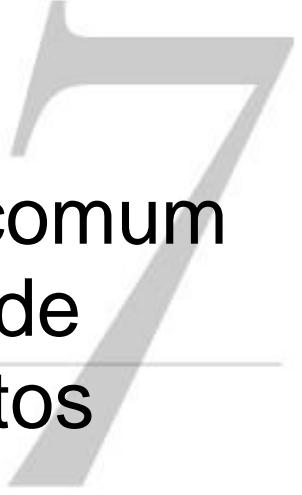
classificação e filtros, `QueryParser` subclasses, e um `HitCollector` implementação todos receberam a atenção neste capítulo. Equipado com os recursos em busca de Neste capítulo e nos capítulos 3 e 5, você tem mais energia suficiente e flexibilidade para integrar em busca Lucene em seus aplicativos.

## Parte 2

# Aplicada Lucene

**L**ucene si é apenas um JAR, Com a verdadeira diversão e poder que vem do que você constrói em torno dela. Esta seção analisa diversas formas de alavancar Lucene. Projetos comumente demanda pesquisa de texto completo do Microsoft Word, PDF,HTML, XMLE outros formatos de documentos. "Analizando os formatos comuns de documento" ilumina as várias maneiras para indexar esses tipos de documentos em Lucene, em última aproximadamente fornecendo uma estrutura reutilizável documento de manipulação de genéricos. Muitos "Ferramentas e extensões" têm sido desenvolvidos para aumentar e prolongar Lucene, com os melhores que estão sendo abordados aqui. Apesar de Java é a linguagem primária usado com Lucene, o formato de índice é uma linguagem neutra. O "portas Lucene" capítulo uso Lucene detalhes de linguagens como C + +, C # e Python. Por último, mas certamente não menos, vários soberba "Estudos de caso" foram gentilmente contribuíram, dando-lhe um olhar profundo em projetos que alcançaram grande sucesso com o poder do Lucene.





# Análise comum formatos de documentos

## Este capítulo aborda

- Parsing XML usando a API SAX 2,0 e Jacarta Commons Digester
- Análise de documentos PDF com PDFBox
- HTML análise usando JTidy e NekoHTML
- Análise de documentos do Microsoft Word com Jacarta POI e da API TextMining.org
- Análise de documentos RTF usando o JDK built-in analisador
- Criação de um quadro de indexação de documentos e aplicação

Até agora neste livro, nós cobrimos vários aspectos do uso do Lucene. Em todos os casos, No entanto, temos tratado exclusivamente com texto simples dos dados. No mundo real, documentos em formato de texto simples estão diminuindo, e em seu lugar, cada vez mais encontrar informações apresentadas nos documentos rich media. Por exemplo:

- Ambientes empresariais mais freqüentemente trabalhar com PDF, Microsoft Word, ou documentos Excel.
- A World Wide Web, tipicamente, contém dados em HTML.
- Aplicações de software são cada vez mais usando XML para trocar dados.

Você pode usar Lucene para pesquisar rich-text documentos como estes? Sim, você pode!

Embora Lucene não inclui ferramentas para documentos automaticamente índice que não são de texto puro, há uma série de ferramentas livres e comerciais que você pode usar para extrair os dados textual dos ricos media.<sup>1</sup> Uma vez extraído, você pode os dados com Lucene como discutido no capítulo 2.

Neste capítulo, vamos começar por apresentar uma simples `DocumentHandler` interface como uma abstração ao ninho dentro de um quadro rico para análise e indexação de documentos de qualquer tipo.

A seguir, vamos caminhar através de exemplos para mostrar a você como analisar e indexar vários

tipos de documentos tais como texto puro, PDF, Microsoft Word, HTML, XML e RTF com Lucene. Cada exemplo utiliza uma ferramenta de terceiros para extrair o texto. Além disso, cada exemplo implementa uma versão especializada do `DocumentHandler` interface.

Finalmente, vamos desenvolver uma pequena estrutura capaz de indexação de arquivos múltiplos

formatos. Vamos usar essa estrutura para escrever um full-blown indexador do sistema de arquivo sam- aplicação ple.

## 7.1 Manuseamento rich-text documentos

Além de mostrar a você como analisar e formatos índice de documento individual, nosso objetivo neste capítulo é criar uma pequena estrutura que você pode usar para indexar documentos comumente encontrados no ambiente de escritório, bem como na Internet.

Esse quadro é útil quando seu objetivo é indexar e permitir que os usuários pesquisem para os arquivos que residem em vários diretórios e são de diferentes formatos, ou se você necessidade de buscar e indexar páginas web de diferentes tipos de conteúdo. Em ambos os casos, o uso

uma estrutura que lida com vários tipos de arquivos automatiza o processo de extração o texto de cada formato de arquivo de modo que você não precisa se preocupar com a forma como

<sup>1</sup> Lucene desenvolvedores propositadamente manter o núcleo Lucene pequena e focada. Uma vez que uma série de ferramentas existem que o texto a partir de extrato de mídia rica, não há necessidade de incluir a funcionalidade duplicada no Lucene.

exatamente o que é realizado. Começamos por definir um genérico `DocumentHandler` interface que define um contrato para parsers documento individual.

### 7.1.1 Criação de uma interface comum DocumentHandler

A interface simples mostrada na listagem de 7.1 consiste de um único método, `getDocumento (InputStream)`, Que retorna uma instância do Lucene `Documento`. Cada dos analisadores documento que criaremos neste capítulo devem implementar esta único método.

Usamos `InputStream` como o tipo de entrada, porque todas as ferramentas que usamos neste capítulo permitir a extração de texto a partir `InputStream`. Utilização `InputStream` também é útil quando você está indexação de arquivos em um sistema de arquivo, porque você pode transformar cada Arquivo instância da classe `em um FileInputStream usando o java.io.FileInputStream (File) construtor.`

Listagem 7.1 DocumentHandler interface que todos os analisadores formato de documento executar

```
public interface DocumentHandler {

    /**
     * Cria um documento Lucene a partir de um InputStream.
     * Esse método pode retornar null <code> </ code>.
     *
     * @ Param é o InputStream para converter em um documento
     * @ Return uma instância de índice para pronta do Documento
     */
    Documento getDocumento (InputStream is)
        joga DocumentHandlerException;

}
```

Todas as implementações deste manipulador retornar uma instância do Lucene `Documento` classe com um ou mais `Campos`. Porque tipos diferentes de armazenar documentos diferentemente meta-dados, o retornado `Documento` instâncias contêm diferentes `Campos`. Para exemplo, documentos HTML normalmente têm títulos, ao passo que os documentos XML não. Assim, o HTML `DocumentHandler` pode retornar um `Documento` com um título `Campo`, Mas o XML `DocumentHandler` não pode.

Se qualquer tipo de erro ocorre, todas as classes de implementação do `DocumentHandler` jogar uma interface `DocumentHandlerException`. Esta é uma exceção verificada, a-sim soas subclasse de Java `Exceção classe`, por isso vamos omitir sua lista.

Em geral, todos os analisadores neste capítulo siga os passos descritos na tabela 7.1.

Tabela 7.1 Comum DocumentHandler etapas de implementação

Passo	Descrição
1. Processo InputStream.	Tomar InputStream entrada. Ler e analisar a InputStream. Extrair o texto das InputStream.
2. Criar Lucene Documento.	Criar uma instância de Lucene Documento. Criar Lucene Campos textual com valores extraídos da InputStream. Adicionar Campos para o Lucene Documento. Devolver o Lucene Documento, pronto para ser indexado pelo chamador.

Por fim, cada implementação de DocumentHandler que apresentamos neste capítulo inclui um principal método, permitindo que você invocá-lo na linha de comando. Todos principais esperam um parâmetro de linha de comando único que representa um caminho para um arquivo. Para manter as listagens de código mais curto e simples possível, nós não verificamos para a entrada válida, por isso certifique-se de sempre dar um contributo válido.

Os analisadores você vai aprender sobre o foco exclusivamente na análise de entrada e criação de Lucene Documento casos, não na indexação real. Afinal de contas, uma vez que esses parsers converter sua contribuição para pronto-a-índice Lucene Documentos, o indexing passo é idêntico para todos os tipos de documentos; não queremos para duplicá-lo em todos os sistemas de arquivo. Além disso, tendo o quadro envolvendo lidar com a indexação desacopla a arquitetura muito bem e permite que a estrutura para adicionar campos comuns para todos os documentos, se desejar (como data da última modificação, temperatura caminho, URL, e assim por diante).

Agora que você tem um entendimento de alto nível de como rich text-documento analisadores de trabalho e como as peças se encaixam no final, vamos começar com o implementação de analisador. XML é um formato de documento comum nestes dias, mesmo aplicações de escritório como OpenOffice usá-lo como seu documento em disco principal formato-então vamos começar com um parser XML.

## 7.2 indexação XML

Nesta seção, vamos converter um trecho de um documento XML em um Lucene Documento. Primeiro vamos usar a API SAX, e então nós vamos usar o Jakarta Commons Digest. Então vamos indexar o trecho com Lucene.

Listagem 7.2 é um trecho de XML que representa uma única entrada de um imaginário livro de endereços. Nosso objetivo final é fazer com que este livro de endereços pesquisável, para que possamos encontrar entradas correspondentes no-lo usando uma sintaxe simples pesquisa, como nome: Zane ou cidade: "New York" (Ver secção 3.1.2 para mais detalhes sobre QueryParser sintaxe).

**Listagem 7,2 trecho de XML que representa uma entrada de livro de endereços**

```
<? Xml version = 'encoding =' '1 .0 utf-8 '>
<address-book>
    <Contact type="individual">
        <name> Zane Pasolini </ name>
        <endereço> 999 W. Prince St. </ address>
        <Cidade> New York </ cidade>
        <provincia> NY </ província>
        <CEP> 10013 < código postal />
        <pais> EUA </ país>
        <telephone> +1 212 345 6789 </ telefone>
    </ Contato>
</ Address book>
```

Apesar de que seria possível criar uma aplicação poderosa e flexível que pode lidar com XML para qualquer Document Type Definition (DTD), o XML dois Documento-Manipulador implementações nesta seção assumem o livro de endereços formato XML mostrado na listagem 7.2, a fim de manter as coisas simples.

Nosso XML DocumentHandler índice de implementações subelemento cada um dos < contato > elemento. Note que o < contato > elemento tem um atributo tipo. Em tanto SAX e implementações Digestor de nossa DocumentHandler, Vamos tratar este atributo como apenas outro campo.

### 7.2.1 Análise e indexação usando SAX

API simples para XML (SAX) define uma interface orientada a eventos em que o analisador invoca um dos vários métodos fornecidos pelo chamador quando um evento de análise ocorre. Os eventos incluem começos e finais de documentos e seus elementos, erros de análise, e assim por diante.

Para dar um exemplo de extração de dados textual a partir de documentos XML, que use Xerces2 Parser Java. Xerces2 Parser Java é desenvolvido sob o Apache XML projeto e pode ser encontrada em <http://xml.apache.org/xerces2-j/index.html>. É implementos versão 2.0.1 da API SAX. Este não é o mais rápido analisador XML atualmente disponível, mas é um dos mais amplamente utilizados analisadores Java XML, e tem sido em torno de muitos anos.

Listagem 7.3 mostra a nossa solução para analisar o livro de endereços XML e convertê-lo a um Lucene Documento.

**Listagem 7,3 DocumentHandler usando a API SAX para analisar uma entrada do catálogo de endereços**

```
public class SAXXMLHandler
    estende DefaultHandler implementa {DocumentHandler}
```

```
/ ** Um buffer para cada elemento XML * /
privada StringBuffer elementBuffer = new StringBuffer ();
privada HashMap attributeMap;
```

```
privada Documento doc;
```

```
Documento público getDocument (InputStream is)
{throws DocumentHandlerException
```

**b** Implementar DocumentHandler interface; parser começar

```
SAXParserFactory SPF = SAXParserFactory.newInstance ();
try {
    Parser SAXParser SPF.newSAXParser = ();
    parser.parse (é, this);
}
catch (IOException e) {
    throw new DocumentHandlerException (
        "Não é possível analisar documentos XML", e);
}
catch (ParserConfigurationException e) {
    throw new DocumentHandlerException (
        "Não é possível analisar documentos XML", e);
}
catch (SAXException e) {
    throw new DocumentHandlerException (
        "Não é possível analisar documentos XML", e);
}
```

```
    retorno doc;
}
```

```
startDocument public void () {
    doc = new Document ();
}
```

**c** Chamado quando análise começa

```
startElement public void (String uri, localName String,
String QName, atts atributos) throws SAXException {
```

```
    elementBuffer.setLength (0);
    attributeMap.clear ();
    if (atts.getLength () > 0) {
        attributeMap = new HashMap ();
        for (int i = 0; i < atts.getLength (); i + +) {
            attributeMap.put (atts.getQName (i), atts.getValue (i));
        }
    }
```

```
}
```

```
    / Chamado quando é encontrada cdata
    pública caracteres void (char [] texto começar, int comprimento, int) {
        elementBuffer.append (texto, iniciar, comprimento);
```

**d** Início de novos Elemento XML  
**e** Acrescentar conteúdo do elemento para elementBuffer

```

    / / Chamado no final do elemento
endElement public void (String uri, localName String, String QName)
    {throws SAXException
    }

    if (qName.equals ("address book")) {
        retorno;
    }
    else if (qName.equals ("contato")) {
        Iterator iter = attributeMap.keySet () iterator ();
        while (iter.hasNext ()) {
            String attName = (String) iter.next ();
            String attValue = (String) attributeMap.get (attName);
            doc.add (Field.Keyword (attName, attValue));
        }
    }
    else {
        doc.add (Field.Keyword (QName, elementBuffer.toString ()));
    }
}

public static void main (String args []) throws Exception {
    Handler = new SAXXMLHandler SAXXMLHandler ();
    Documento doc = handler.getDocument (
        new FileInputStream (new File (args [0])));
    System.out.println (doc);
}
}

```

Os cinco principais métodos nesta lista são `getDocument`, `startDocument`, `startElement`, `caracteres` e `endElement`. Observe também o `elementBuffer` `StringBuffer` e o `attributeMap` `HashMap`. O primeiro é usado para armazenar a representação textual de o CDATA delimitada pela elemento do documento atual. Alguns elementos podem con-Tain atributos, tais como a `<contato>` elemento que contém atributos tipo, Em nossa livro de entrada de endereço. O `attributeMap` é usado para armazenar nomes e o valor de atributos do elemento atual.

- b O `getDocument` método não fazer muito trabalho: Ele cria um novo analisador SAX e passa uma referência para o `InputStream` do documento XML. De lá, o implementação de analisador chama os outros quatro principais métodos desta classe, que juntos criar um Lucene `Documento` que é, eventualmente, devolvido pelo `getDocument` método.
- c Em `startDocument`, Que é chamado ao analisar documento XML começa, temos apenas criar uma nova instância do Lucene `Documento`. Este é o `Documento` que vamos eventualmente preencher com Campos.
- d O `startElement` método é chamado sempre que o início de um novo XML ele-mento é encontrado. Nós primeiro apagar o `elementBuffer` `StringBuffer` definindo sua comprimento para zero, e limpar o `attributeMap` para remover os dados associados com o

elemento anterior. Se o elemento atual tem atributos, podemos percorrer os e salvar os seus nomes e valores na `attributeMap`. No caso do XML documento na listagem 7.2, isso só acontece quando `startElement` método é chamado para o `<contato>` elemento, pois só elemento que tem um atributo.

- e O `caracteres` método pode ser chamado várias vezes durante o processamento de um único elemento XML. Em que anexamos ao nosso `elementBuffer` o conteúdo do elemento passado para o método.

- f O último método é de interesse `endElement`, Onde você pode finalmente ver mais Lucene em ação. Este método é chamado quando o analisador processa o fechamento tag do elemento atual. Portanto, este é o método onde temos todas as informações sobre o elemento XML que acabou processado. Nós não estamos interessados na indexação o elemento de nível superior, `<address-book>`, De modo que retornar imediatamente

a partir do método nesse caso. Da mesma forma, não estamos interessados na indexação do `<contato>` elemento. No entanto, estamos interessados em que a indexação `<contato>'S` atributos, por isso usamos `attributeMap` para obter os nomes de atributos e valores, e adicionar -los para o Lucene Documento. Todos os outros elementos da nossa entrada do endereço são

tratados de forma igual, e nós cegamente indexá-los como palavra-chave `Field.Keyword`.

Atributo

valores como elemento de dados assim são indexados.

Se você olhar para trás à tabela 7.1, você verá que o parser XML em 7.3 listagem segue todos os passos que descrevemos. Como resultado, temos uma Lucene ready-to-index Documento

preenchida com Campos, cujos nomes são derivados de nomes de elementos XML e cujos valores correspondem ao conteúdo textual desses elementos. Embora esta código por si só vai deixar você indexar documentos XML, vamos olhar para outra ferramenta útil para analisar XML: Digester.

### 7.2.2 Análise e indexação usando Digester

Digester, localizado na <http://jakarta.apache.org/commons/digester>, é um subprojeto do projeto Jakarta Commons. Ele oferece uma interface simples e de alto nível para o mapping documentos XML para objetos Java; alguns desenvolvedores acham mais fácil de usar do que

DOM ou SAX parsers XML. Quando Digester encontra desenvolvedor-definidos padrões de um documento XML, é preciso desenvolvedor especificado ações.

O `DigesterXMLHandler` classe na listagem de 7.4 analisa documentos XML, como nossa entrada do catálogo de endereços (mostrado na listagem 7.2), e retorna um Lucene Documento

com elementos XML representados como Campos.

Listagem 7.4 DocumentHandler usando Jakarta Commons Digestor para analisar XML

```
public class DigesterXMLHandler implements {DocumentHandler  
    privada Digester cavar;
```

```

doc Documento private static;

pública DigesterXMLHandler () {

    / / Digestor instanciar e desabilitar a validação de XML
    dig = new Digester ();
    dig.setValidating (false);

    / / Instanciar a classe DigesterXMLHandler
    dig.addObjectCreate ("address book", DigesterXMLHandler.class);
    / / Contato instanciar a classe
    dig.addObjectCreate ("address-book/contact", Contact.class);
}

Propriedade de tipo / set / de instância Fale quando 'tipo'
/ / Atributo é encontrado
dig.addSetProperties ("address-book/contact", "tipo", "tipo");
dig.addCallMethod ("address-book/contact/name", Regra 4: Conjunto de contato do
                    propriedade de nome"SetName", 0); e
dig.addCallMethod ("address-book/contact/address",
                    "SetAddress", 0);
dig.addCallMethod ("address-book/contact/city",
                    "SetCity", 0);
dig.addCallMethod ("address-book/contact/province",
                    "SetProvince", 0);
dig.addCallMethod ("address-book/contact/postalcode",
                    "SetPostalcode", 0);
dig.addCallMethod ("address-book/contact/country",
                    "SetCountry", 0);
dig.addCallMethod ("address-book/contact/telephone",
                    "SetTelephone", 0);

/ / Chamada de método "populateDocument 'quando o próximo
/ / 'Address-book/contact' padrão é visto
dig.addSetNext ("address-book/contact", "populateDocument"); f
}

Regra 5: Call populateDocument
}

Documento sincronizado público getDocument (InputStream is) g
throws DocumentHandlerExceptionExecutar

try {
    dig.parse (is); Inicia a análise em XML
    InputStream)
    catch (IOException e) {
        throw new DocumentHandlerException (
            "Não é possível analisar documentos XML", e);
    }
    catch (SAXException e) {
        throw new DocumentHandlerException (

```

Regra 1: Criar instância de  
DigesterXMLHandler      b

Regra 2: Cria uma instância de Contato      c

Regra 3: Conjunto de contato atributo do tipo      d

Regra 4: Conjunto de contato do  
propriedade de nome"SetName", 0); e

Regra 5: Call populateDocument      f

DocumentHandler interface      g

Início da análise em XML      h

```

        "Não é possível analisar documentos XML",
    );
}

retorno doc;
}

public void populateDocument (contact) {
    / / Cria um documento em branco Lucene
    doc = new Document ();

    doc.add (Field.Keyword ("tipo", contact.getType ()));
    doc.add (Field.Keyword ("nome", contact.getName ()));
    doc.add (Field.Keyword ("endereço", contact.getAddress ()));
    doc.add (Field.Keyword ("city", contact.getCity ()));
    doc.add (Field.Keyword ("provincia", contact.getProvince ()));
    doc.add (Field.Keyword ("CEP", contact.getPostalcode ()));
    doc.add (Field.Keyword ("country", contact.getCountry ()));
    doc.add (Field.Keyword ("telefone", contact.getTelephone ()));

}

/*
 * Classe JavaBean que mantém as propriedades de cada Contato
 * Entrada. É importante que esta classe ser pública e
 * Estática, para Digestor para ser capaz de instanciar
 * It.
 */
Fale classe public static {
    tipo String privado;
    private String nome;
    Endereço String privado;
    cidade String privado;
    província de private String;
    CEP String privado;
    pais String privado;
    telefone private String;

    settype public void (String Newtype) {
        type = Newtype;
    }
    getType public String () {
        tipo de retorno;
    }

    setName public void (String newName) {
        name = newName;
    }
    public String getName () {
        retornar o nome;
    }
}

```

**Eu** Preencher Lucene  
Documento com campos

```
setAddress public void (String newAddress) {
    address = newAddress;
}
getAddress public String () {
    endereço de retorno;
}

setCity public void (newCity String) {
    cidade = newCity;
}
getCity public String () {
    retorno da cidade;
}

setProvince public void (String newProvince) {
    província = newProvince;
}
getProvince public String () {
    retorno província;
}

setPostalcode public void (String newPostalcode) {
    CEP = newPostalcode;
}
getPostalcode public String () {
    retorno CEP;
}

setCountry public void (String newCountry) {
    country = newCountry;
}
getCountry public String () {
    retorno país;
}

setTelephone public void (String newTelephone) {
    telefone = newTelephone;
}
getTelephone public String () {
    retorno de telefone;
}

}

public static void main (String [] args) throws Exception {
    Handler = new DigesterXMLHandler DigesterXMLHandler ();
    Documento doc =
        handler.getDocument (FileInputStream novo (new File (args [0]))));
    System.out.println (doc);
}
```

Esta é uma peça longa de código, e que merece algumas explicações. No `Digestor-XMLHandler` construtor, criamos uma instância de `Digestor` e configurá-lo por especificação várias regras. Cada regra especifica uma ação e um padrão que irá acionar o ação quando encontrado.

- b A primeira regra diz `Digestor` para criar uma instância da `DigesterXMLHandler` classe quando o padrão "Address book" é encontrado. Ele faz isso usando `digestor addObjectCreate` método. Porque `<address-book>` é o elemento de abertura em nosso Documento XML, esta regra é disparada em primeiro lugar.  
A próxima regra instrui `Digestor` para criar uma instância da classe `Contato` quando encontra o `<contato>` elemento filho sob a `<address-book>` pai, especificado com o "Address-book/contact" padrão.  
Para lidar com o `<contato>` atributo do elemento, vamos definir o `tipo` propriedade da `Contato` exemplo, quando encontra o `Digestor` `tipo` atributo do `<contato>` elemento.
- c Para isso, usamos `digestor addSetProperties` método. O `Contato` classe é escrita como uma classe interna e contém apenas métodos setter e getter.  
Nosso `DigesterXMLHandler` classe contém várias regras de aparência semelhante, que `Digestor` é chamada `addCallMethod` método. Eles são usados para definir várias `Contato` propriedades. Por exemplo, uma chamada como `dig.addCallMethod ("address-book/contact / nome ", " setName ", 0)` chama o `setName` método de nossa `Contato` exemplo. Ele faz isso quando começa o processamento do `Digestor` `<name>` elemento, encontrado em o pai `<address-book>` e `<contato>` elementos. O valor da `setName` param-método é o valor fechado por `<name>` e `</ Name>` tags. Se você considerar nossa amostra livro de endereços a partir da listagem 7.2, este chamaria `setName ("Zane Pasolini")`. Usamos `digestor addSetNext` método para especificar que o `populateDocument` (`Contato`) método deve ser chamado quando o fechamento `</ Contato>` elemento é processado.  
O `DocumentHandler`'S `getDocument` método tem uma `InputStream` para o XML documento para analisar.
- d Aqui começamos analisar o XML `InputStream`. Finalmente, preenche uma `Lucene Document` com Campos dados coletados por contendo o `Contato` classe durante a análise.
- e
- f
- g
- h
- Eu

É importante que você considere a ordem em que as regras são passados para `Digestor`. Embora nós poderíamos mudar a ordem das várias `addSetProperties ()` regras na nossa classe e código ainda tem bom funcionamento, trocando a ordem de `addObjectCreate ()` e `addSetNext ()` resultaria em um erro.

Como você pode ver, `Digestivo` fornece uma interface de alto nível para analisar XML documents. Porque nós temos as nossas regras especificadas análise XML programaticamente, a nossa

`DigesterXMLHandler` pode analisar apenas o nosso livro de endereços formato XML. Por sorte, `Digestor` permite que você especifique as mesmas normas de forma declarativa usando o esquema XML

descrito na DTD digestor-regras, que está incluído no digestor dis-  
ção. Usando uma abordagem declarativa, você pode criar um XML baseado em Digestor  
parser que pode ser configurado em tempo de execução, permitindo uma maior flexibilidade. Se  
você está  
curioso, um exemplo de digestor de regras aparece na seção 10.7.

Nos bastidores, Digestivo usa recursos de reflexão do Java para criar instâncias de  
classes, então você tem que prestar atenção ao acesso modificadores de evitar o asfixia  
Digestor.

Por exemplo, o interior `Contato` classe é instanciada dinamicamente, por isso deve ser publica-  
lic. Da mesma forma, a nossa `populateDocument` (`Contato`) método precisa ser público porque  
que, também, será chamado de forma dinâmica. Digestor também necessário que os nossos  
Documento

exemplo, ser declarado como estático; a fim de fazer `DigesterXMLHandler` thread-safe,  
temos que sincronizar o acesso à `getDocument` (`InputStream`) método.

Agora que você começou uma idéia de como o nosso `DocumentHandler` implementações  
trabalho, e você sabe como usar tanto a API SAX e Digestor. Vamos passar para  
o formato seguinte popular: PDF.

### 7.3 indexação de um documento PDF

---

Portable Document Format (PDF) é um formato de documento inventado pela Adobe Sys-  
tems mais de uma década atrás. Este formato vai além de simples dados textuais, permitindo  
autores de documentos para incorporar imagens, hyperlinks, cores e muito mais. Hoje, PDF  
é generalizada e, em alguns domínios, é o formato dominante. Por exemplo, ofi-  
formulários especiais tais como formulários de pedido de visto de viagem, formas de seguro de  
saúde, fiscais dos EUA  
formulários de declaração, manuais de produtos, e assim por diante na maioria das vezes vêm  
como PDF docu-  
mentos. Mesmo este livro está disponível em formato PDF; Manning Publications vende capítulos  
de  
a maioria de seus livros eletronicamente, permitindo aos clientes comprar capítulos individuais  
e imediatamente baixá-los.

Se você já abriu documentos PDF, você provavelmente usou um aplicativo  
chamado Adobe Reader. Embora esta aplicação tem uma funcionalidade incorporada em busca,  
que  
não é muito poderoso, permitindo que o usuário apenas duas opções de busca: correspondência  
total ou  
palavras parciais e executar uma busca case-sensitive ou insensível. Sua busca PDF  
necessidades podem ir além disso. Além disso, o que você faz se você precisa procurar um  
coleção completa de documentos PDF? Você usa o Lucene, é claro!

Nesta seção, você aprenderá como usar PDFBox, um terceiro biblioteca Java, para  
analisar documentos PDF, ao furar com a nossa `DocumentHandler` interface. Em  
Além de nossa própria integração do Lucene e PDFBox, vamos mostrar-lhe como  
use built-in PDFBox classes de integração Lucene.

### 7.3.1 Extraindo texto e indexação usando PDFBox

PDFBox é livre, biblioteca open-source escrito por Ben Litchfield, você pode encontrá-lo em <http://www.pdfbox.org/>. Existem diversas ferramentas gratuitas capaz de extrair o texto a partir de arquivos PDF, optamos PDFBox para sua popularidade, dedicou o autor suporta na lista de discussão Lucene, eo fato de que esta biblioteca inclui classes que trabalham com Lucene muito bem.

Listagem 7.5 mostra como extrair o conteúdo textual de um documento PDF, bem como documento de meta-dados, e criar um Lucene Documento adequados para a indexação.

Listagem 7.5 DocumentHandler usando a biblioteca PDFBox extrair texto de Arquivos PDF

```

PDFBoxPDFHandler public class {DocumentHandler
    password String public static = "password";
    PDFBoxPDFHandler pública () {
        }
Documento público getDocument (InputStream is) throws DocumentHandlerException
    COSDocument cosDoc = null;
    try {
        cosDoc = parseDocument (is);InputStream carga na memória
    }
    catch (IOException e) {
        closeCOSDocument (cosDoc);
        throw new DocumentHandlerException (
            "Não é possível analisar documento PDF", e);
    }
    / / Descriptografar o documento PDF, se for criptografado
    try {
        if (cosDoc.isEncrypted ()) {Documento descriptografar
            Decryptor DecryptDocument DecryptDocument = new (cosDoc);
            decryptor.decryptDocument (password);
        }
    }
    catch (CryptographyException e) {
        closeCOSDocument (cosDoc);
        throw new DocumentHandlerException (
            "Não é possível descriptografar documento PDF", e);
    }
    catch (InvalidPasswordException e) {
        closeCOSDocument (cosDoc);
        throw new DocumentHandlerException (

```

```

        "Não é possível descriptografar documento
        PDF", e);
    }
    catch (IOException e) {
        closeCOSDocument (cosDoc);
        throw new DocumentHandlerException (
            "Não é possível descriptografar documento
            PDF", e);
    }

    / / Extrair o conteúdo textual do documento PDF
    String docText = null;
    try {
        PDFTextStripper stripper PDFTextStripper = new ();
        docText = stripper.getText (novo PDDocument (cosDoc));
    }
    catch (IOException e) {
        closeCOSDocument (cosDoc);
        throw new DocumentHandlerException (
            "Não é possível analisar documento PDF", e);
    }
}

Documento doc = new Document ();
if (docText != null) {
    doc.add (Field.UnStored ("corpo", docText));
}

/ / Extrair documento PDF de meta-dados
PDDocument pdDoc = null;
try {
    PDDocumentInformation DOCINFO =
        pdDoc.getDocumentInformation ();

    Autor String = docInfo.getAuthor ();
    String title = docInfo.getTitle ();

    Palavras-chave String = docInfo.getKeywords ();
    Resumo String = docInfo.getSubject ();
    if ((autor! = null) & &! author.equals ("")) {
        doc.add (Field.Text ("autor", autor));
    }
    if ((título! = null) & &! title.equals ("")) {
        doc.add (Field.Text ("title", title));
    }
    if ((palavras-chave! = null) & &! keywords.equals ("")) {
        doc.add (Field.Text ("keywords", palavras-chave));
    }
    if ((resumo! = null) & &! summary.equals ("")) {
        doc.add (Field.Text ("resumo", resumo));
    }
}
catch (Exception e) {
    closeCOSDocument (cosDoc);
    closePDDocument (pdDoc);
    System.err.println ("Não é possível obter documento PDF meta-dados:");
}

```

e    Extrato textual  
conteúdo

f    Save Field UnStored  
no documento Lucene

Extrato  
documento  
meta-dados

```

        + E.getMessage ());
    }

    retorno doc;
}

private static COSDocument parseDocument (InputStream is)
throws IOException {
    Parser PDFParser PDFParser = new (is);
    parser.parse ();
    retorno parser.getDocument ();
}

private void closeCOSDocument (cosDoc COSDocument) {
    if (cosDoc! = null) {
        try {
            cosDoc.close ();
        }
        catch (IOException e) {
            / / Comê-lo, o que mais podemos fazer?
        }
    }
}

private void closePDDocument (pdDoc PDDocument) {
    if (pdDoc! = null) {
        try {
            pdDoc.close ();
        }
        catch (IOException e) {
            / / Comê-lo, o que mais podemos fazer?
        }
    }
}

public static void main (String [] args) throws Exception {
    Manipulador PDFBoxPDFHandler PDFBoxPDFHandler = new ();
    Documento doc = handler.getDocument (
        new FileInputStream (new File (args [0]))));
    System.out.println (doc);
}
}

```

- b** O DocumentHandler's getDocument método leva uma referência para o PDF documento do InputStream.
- c** Aqui nós carregar o InputStream na memória, é representado como uma instância de uma COSDocument objeto.

- d Os documentos PDF podem ser protegidas por senha, e PDFBox permite descriptografar -los antes da análise deles. Nossa `PDFBoxPDFHandler` expõe a senha a ser utilizado para a decodificação como uma variável `public static`, que deve ser definida explicitamente pela chamador, antes da análise de documentos criptografados.
- e Agora vamos extraír o conteúdo textual do documento, formatação e ignorando outras estruturas PDF.
- f Nós salvar o Campo `UnStored` em um Lucene Documento e usar o texto extraído como seu valor.
- g Como você pode ver nesta lista de códigos, PDFBox faz uso do documento PDF estrutura e extratos do documento de meta-dados, como autor, palavras-chave, sommary, título e, além de retirar o conteúdo textual do documento corpo. Isso nos permite adicionar mais ricos Documentos para o índice e proporcionar um melhor resultados de pesquisa no final.

Nós armazenar os meta-dados nos seguintes Campos: autor, palavras-chave, sumário E título. Temos que ter cuidado para não armazenar valores nulos, porque nulos Campos são inválido. Nós também não deseja armazenar em branco Campos, por isso realizamos apropriado verifica antes de adicionar meta-dados a uma instância do Lucene Documento.

Desde documento meta-dados não é crucial para que, se lança uma PDFBox IOException ao extraír meta-dados que escolher apenas para imprimir um aviso em vez de jogando um DocumentHandlerException.

### 7.3.2 apoio Built-in Lucene

Listagem 7.5 demonstra a maneira de baixo nível de extraír dados de um PDF document. A distribuição PDFBox também vem com duas classes que os usuários podem querer considerar usar se você não precisa de um bom controle sobre Lucene Documento creation. Se você só precisa de uma maneira rápida de índice de um diretório de arquivos PDF ou um PDF único arquivo, ou se você só quer testar PDFBox, você pode usar o suporte incorporado Lucene PDFBox. Esta abordagem pode ser rápido, como você está prestes a ver, mas também limita o que é extraído do arquivo PDF, o Lucene Field documentos são criados, e como eles são analisados e indexados.

PDFBox de `org.pdfbox.searchengine.lucene` pacote contém duas classes: `IndexFiles` e `LucenePDFDocument`. Nós discutirão os em seguida.

#### Usando a classe IndexFiles

`IndexFiles` é uma classe simples que expõe um método único para a indexação de um único Diretório sistema de arquivos. Veja como você pode usá-lo:

```
pública PDFBoxIndexFiles classe {
    public static void main (String [] args) throws Exception {
```

```

IndexFiles indexFiles = new IndexFiles ();
indexFiles.index (new File (args [0]), é verdade, args [1]);
}
}

```

Este código chama o `índice` método em `IndexFiles` classe passando argumentos de linha de comando. A saída deste programa é o seguinte (é claro, você tem para garantir que seu classpath inclui os JARs PDFBox e Lucene, bem como o JAR que vem com este livro):

```

$ Java lia.handlingtypes.pdf.PDFBoxIndexFiles
    / Home / otis / PDFs / tmp / pdfindex

Indexação de documentos PDF: / home/otis/PDFs/Concurrency-j-jtp07233.pdf
Indexação de documentos PDF: / home / otis / PDFs / CoreJSTLAppendixA.pdf
Indexação de documentos PDF: / home/otis/PDFs/CoreJSTLChapter2.pdf
Indexação de documentos PDF: / home/otis/PDFs/CoreJSTLChapter5.pdf
Indexação de documentos PDF: / home / otis / PDFs / Google-Arch.pdf
Indexação de documentos PDF: / home/otis/PDFs/JavaCookbook-Chapter22-RMI.pdf
Indexação de documentos PDF: / home / otis / PDFs / JavaSockets.pdf
Indexação de documentos PDF: / home / otis / PDFs / LinuxBackup.pdf
Indexação de documentos PDF: / home / otis / PDFs / SEDA.pdf
Indexação de documentos PDF: / home / otis / PDFs / ViTutorialWithCheatsheet.pdf
Indexação de documentos PDF: / home / otis / PDFs / design-patterns.pdf
Indexação de documentos PDF: / home / otis / PDFs / jndi.pdf
Indexação de documentos PDF: / home / otis / PDFs / pagerank.pdf
Indexação de documentos PDF: / home/otis/PDFs/servlet-2_3-fcs-spec.pdf
Indexação de documentos PDF: / home / otis / PDFs / tilesAdvancedFeatures.pdf
Otimizando index ...
42971 milissegundos total

```

O `IndexFiles` classe fez tudo por nós: Ele encontrou todos os PDFs em um determinado di-história, é analisado eles, e indexados-los com Lucene. Isso pode ser um pouco tanto para aqueles que gostam de manter algum controle em suas próprias mãos. Assim, PDFBox

vem com um `LucenePDFDocument` classe que é ainda mais simples: Ele analisa um dado PDF arquivo e retorna um Lucene povoadas Documento exemplo. Vamos ver como ele funciona.

### Usando a classe `LucenePDFDocument`

O `LucenePDFDocument` classe é um pouco semelhante ao nosso `DocumentHandler` 'S `getDocument (InputStream)` método. Ele oferece dois métodos estáticos que retornam um `Lucene Documento` quando passou uma instância de `Arquivo` ou uma instância de uma `URL` objeto.

O código a seguir demonstra o uso do método que leva um `Arquivo` objeto como um parâmetro:

```

PDFBoxLucenePDFDocument public class {
    public static void main (String [] args) throws Exception {
        Documento doc = LucenePDFDocument.getDocument (new File (args [0]));
    }
}

```

```
        System.out.println (doc);
    }
}
```

Essa classe é um wrapper simples em torno do PDFBox LucenePDFDocument classe. Depois acrescentando todos os JARs necessários para o classpath, passamos o nome do arquivo de especadas na linha de comando para esta classe e, em seguida, imprimir o Lucene, resultando Documento. Como mostrado aqui, esta classe cria um Lucene Documento com Campos chamado sumário, produtor, conteúdo, modificado, url E caminho:

```
$ Java lia.handlingtypes.pdf.PDFBoxLucenePDFDocument
=> / Home / otis / PDFs / Google-Arch.pdf
Documento <UnIndexed <resumo: 22
Serviço Web poucos exigem tanto
computação por solicitação como motores de busca.
Em média, uma única consulta no Google lê
centenas de megabytes de dados e consome
dezenas de bilhões de ciclos de CPU. Apoio a uma
fluxo de solicitação de pico de milhares de consultas
por segundo requer uma infra-estrutura compa-
comparável em tamanho à do maior supercom-
instalações puter. Combinar mais de
15.000 commodity classe PCs com culpa tol-
ERANT software cria uma solução que é mais
relação custo-benefício do que um c>
Distiller texto <Producer:Acrobat 4,05 para Macintosh>
<CreationDate:0demeknhc> Texto
<contents:java.io.InputStreamReader@1193779> Texto
org.apache.lucene.document.Field @ 8916a2
<modified:0dhb25ujs> Palavra-chave
<url:/home/otis/PDFs/Google-Arch.pdf> Não indexados
Não indexados <path:/home/otis/PDFs/Google-Arch.pdf>>
```

PDFBox e Lucene fazer um bom casal. Mais importante, eles tornam mais fácil para nos a fazer coleções de documentos PDF pesquisáveis.

## 7,4 indexação de um documento HTML

---

HTML está em toda parte. A maioria dos documentos web estão no formato HTML. A Web é cur-  
atualmente o maior repositório de informações sobre o planeta. Adicionar dois e dois  
juntos, e é claro que precisamos para ser capaz de indexar e pesquisar volumes de  
documentos HTML existentes. Que é o pão ea manteiga de motores de busca da Web,  
e muitas empresas construíram negócios com base nessa necessidade. HTML análise é  
não trivial, porém, porque muitos sites ainda não se conformam com as últimas W3C stan-  
emenda as normas de XHTML (HTML como um dialeto XML). Analisadores especializados foram  
desenvolvidos, que pode interpretar leniently bastardizations várias HTML.

#### 7.4.1 Obtendo os dados de origem HTML

Listagem 7.6 contém o documento HTML que estaremos usando a análise de HTML parsers destaque nesta seção. Uma grande porcentagem de documentos HTML descapazes na Web não são bem formados, e nem todos os parsers lidar com essa situação igualmente bem. Nesta seção, usamos o parsers JTidy e NekoHTML, tanto de quais são sólidos parsers HTML capaz de lidar com HTML quebrado.

Listagem 7.6 O documento HTML que vamos analisar, indexar e, finalmente, pesquisa

```
<html>
  <head>
    <title>
      Suprimentos de energia do laptop estão disponíveis apenas na Primeira Classe
    </ Title>
  </ Head>
  <body>
    Código <h1>, Escrever, Fly </ h1>
    Este capítulo está sendo escrito 11 mil metros acima do Terra Nova, Novo.
  </ Body>
</ Html>
```

Agora que temos um pouco de HTML para trabalhar, vamos ver como podemos processá-lo com JTidy.

#### 7.4.2 Usando JTidy

Com uma década atrás dele, Tidy é um veterano entre os parsers HTML. O original Tidy foi implementado em C por Dave Raggett, mas o desenvolvimento do projeto parou em 2000. Um grupo de desenvolvedores entusiasmados recentemente assumiu o projeto e deu-lhe uma segunda vida. Tidy está agora ativamente desenvolvidos em <http://tidy.sourceforge.net/>.

JTidy é uma porta de Tidy Java, escrito por Andy Quick, sua casa está em <http://jtidy.sourceforge.net/>. Depois de quatro anos sem um lançamento, o projeto JTidy comecei recentemente um administrador novo projeto e desenvolvedor, Fabrizio Giustina, ele começou a trabalhar no JTidy no início de 2004 e começou a prepará-lo para novos lançamentos.

O código na listagem de 7.7 representa uma implementação baseada em JTidy do nosso `DocumentHandler` interface. JTidy é invocada pelo seu `parseDOM` método, a que passamos um documento HTML `InputStream`. A partir daí nós usamos padrão DOM Métodos API para obter valores textual por dois elementos HTML que queremos índice: do documento `título` e `corpo`.

## Listagem 7.7 DocumentHandler usando JTidy para extrair texto de documentos HTML

```

public class JTidyHTMLHandler implements DocumentHandler

    pública org.apache.lucene.document.Document
        getDocument (InputStream é) throws DocumentHandlerException {
            Arrumado arrumado = new Tidy ();
            tidy.setQuiet (true);
            tidy.setShowWarnings (false);
            root = org.w3c.dom.Document tidy.parseDOM (é, null);
            Elemento rawDoc = root.getDocumentElement ();

            org.apache.lucene.document.Document doc =
                nova org.apache.lucene.document.Document ();

            String title = getTitle (rawDoc); Obter título
            String body = getBody (rawDoc);
            if ((título! = null) & & (! title.equals ("")) ) {
                doc.add (Field.Text ("title", title));
            }
            if ((corpo! = null) & & (! body.equals ("")) ) {
                doc.add (Field.Text ("corpo", corpo));
            }

            retorno doc;
        }

        / ***
         * Obtém o texto do título do documento HTML.
         *
         * @ RawDoc o elemento DOM para extrair Node título de
         * @ Return o texto do título
         */
        protegidos String getTitle (Element rawDoc) {
            if (rawDoc == null) {
                return null;
            }

            String title = "";

            Crianças NodeList = rawDoc.getElementsByTagName ("title");
            if (children.getLength ()> 0) {
                TitleElement elemento = ((Element) children.item (0));
                Texto = texto (Texto) titleElement.getFirstChild ();
                if (texto! = null) {
                    title = text.getData ();
                }
            }
            retorno título;
        }
    }
}

```

b      método getDocument

c      Parse HTML  
InputStream

d      Obter texto em todos os elementos  
entre <body> e  
</ Body>

e      Obter texto de  
<title> primeiro

f

```

    /**
     * Obtém o texto do corpo do documento HTML.
     *
     * @ RawDoc elemento do DOM para extrair Node corpo de
     * @ Return o corpo do texto
     */
    protegidos String getBody (Element rawDoc) {
        if (rawDoc == null) {
            return null;
        }Obter referências a

        <body>
Corpo String = "";
Crianças NodeList = rawDoc.getElementsByTagName ("body");
if (children.getLength ()> 0) {
    body = getText (children.item (0));Extrair todo o texto entre
        <body> e </ body>)           h
    retorno do corpo;

}

/**
 * Exrai texto a partir do nó DOM.
 *
 * @ Param node um nó DOM
 * @ Return o valor de texto do nó
 */
String getText protegidos (Node nó) {Extrair todo o texto em todos os elementos
    sob o nó especificadoCrianças NodeList = node.getChildNodes ();
    StringBuffer sb = new StringBuffer ();
    Eu
    for (int i = 0; i <children.getLength (); i++) {
        Nó filho = children.item (i);
        switch (child.getNodeType ()) {
            Node.ELEMENT_NODE caso:
                sb.Append (getText (criança));
                sb.Append ("");
                break;
            Node.TEXT_NODE caso:
                sb.Append (. ((texto) criança) getData ());
                break;
        }
        sb.toString return ();
    }
}

public static void main (String args []) throws Exception {
    Handler = new JTidyHTMLHandler JTidyHTMLHandler ();
    org.apache.lucene.document.Document doc = handler.getDocument (
        new FileInputStream (new File (args [0 ])));
    System.out.println (doc);
}
}

```

- b DocumentHandler's `getDocument` método, a que passamos documento HTML `InputStream`, Chama analisador JTidy do DOM e então cria um Lucene Documento.
- c A chamada para a JTidy `parseDOM` método analisa o código HTML dado `InputStream` e formas de uma árvore DOM, adequado para travessia.
- d A chamada para `getTitle` obtém o valor textual do título do documento HTML. Este texto é então usado para preencher o Lucene Documento exemplo.
- e A chamada para `getBody` obtém o texto completo do documento HTML. Este texto é, então, usado para preencher o Lucene Documento exemplo.
- f O `getTitle` método percorre a árvore DOM e retorna o valor textual da primeiro `<title>` elemento que encontra.
- g Nós usamos o padrão DOM API chamada para obter uma lista de referências a todos os `<body>` elementos. Normalmente, há apenas um `<body>` elemento presente em um container Documento HTML.
- h O `getBody` método chama o genérico `getText` método para retirar todo o texto do documento HTML. Todo o texto encontrado entre `<body>` e `</ Body>` elementos é retornado.
- i O `getText` método é um método genérico para extrair todo o texto encontrado em todos os elementos sob o DOM especificado Nô.

Eu

Como foi o caso em outras partes deste capítulo, Campos pode ser nulo ou vazio, de modo que forma as verificações necessárias antes de adicionar título e corpo para o índice. Porque a API DOM contém uma classe chamada Documento (`org.w3c.dom.Document`), Evitamos conflitos com Lucene Documento usando nomes de classe totalmente qualificado para ambos Documento classes.

Em seguida, vamos olhar para o primo mais novo do JTidy, NekoHTML.

#### 7.4.3 Usando NekoHTML

NekoHTML é um parente recém-chegado ao trabalho de parsers HTML, mas seu autor Andy Clark não é. Ele é um nome conhecido no mundo de analisadores, e um monte de sua trabalho pode ser encontrado no parser XML Xerces-J. Como tal, não é nenhuma surpresa que NekoHTML é escrito usando o Interface Xerces Native (XNI), que é o fundação da implementação Xerces2.

NekoHTML é um scanner simples HTML e balanceador de tag que permite que aplicações programadores ção para analisar documentos HTML e acessá-los usando o padrão Interfaces XML. O analisador pode verificar os arquivos HTML e arrumar uma série de erros comuns que os autores humanos e do computador fazem por escrito HTML documentos. NekoHTML acrescenta faltando elementos pai, fecha automaticamente elementos com tags end opcional, e pode lidar com marcas de elemento inline incompatíveis.

NekoHTML faz parte do conjunto de Andy Clark de CyberNeko Ferramentas para XNI, você pode encontrá-lo em <http://www.apache.org/~andyc/neko/doc/index.html>. Listagem 7.8 mostra nosso DocumentHandler implementação baseada em NekoHTML. Ele usa o DOM API, assim como o exemplo JTidy. No entanto, aqui vamos nós dar um passo adiante e fornecer uma implementação pouco mais geral nos dois `getText` métodos.

**Listagem 7.8 DocumentHandler usando o NekoHTML extrair texto de HTML documentos**

```

public class NekoHTMLHandler implements DocumentHandler {
    parser DOMFragmentParser iativa DOMFragmentParser = new ();
    Document público getDocument (InputStream is) throws DocumentHandlerException {
        DocumentFragment nó =
            . HTMLDocumentImpl nova () createDocumentFragment (); Criar
        try (DocumentFragment
            parser.parse (novo InputSource (é), o nó); Analisar InputStream
        }
        catch (IOException e) {
            throw new DocumentHandlerException (
                "Não é possível analisar documento HTML:", e);
        }
        catch (SAXException e) {
            throw new DocumentHandlerException (
                "Não é possível analisar documento HTML:", e);
        }
    }

    org.apache.lucene.document.Document doc =
        nova org.apache.lucene.document.Document ();

    StringBuffer sb = new StringBuffer ();
    getText (sb, nó, "title");
    String title = sb.toString ();
    sb.setLength (0); StringBuffer clara
    getText (sb, nó);
    String texto sb.toString = ();
    if ((titulo! = null) & & (! title.equals ("")))
        doc.add (Field.Text ("title", title));
    if ((texto! = null) & & (! text.equals ("")))
        doc.add (Field.Text ("corpo", texto));
    }

    retorno doc;
}

```

b Parser DOM Neko para HTML  
c Método getDocument

d Criar DocumentFragment

e Analisar InputStream

f Extrato / texto loja de <title>

g Extrair todo o texto DOM de Node

h

```

private void getText (StringBuffer sb, Node nó) {
    if (node.getNodeType () == Node.TEXT_NODE) {
        sb.Append (node.getNodeValue ());
    }
    Crianças NodeList = node.getChildNodes ();
    if (crianças! = null) {
        int len = children.getLength ();
        for (int i = 0; i <len; i + +) {
            getText (sb, children.item (i));
        }
    }
}

private boolean getText (StringBuffer sb, nó Nó,
    Elemento String) {Extrair texto de nós que representam elemento especificado
if (node.getNodeType () == Node.ELEMENT_NODE) {
    if (element.equalsIgnoreCase (node.get nodeName ())) {
        getText (sb, nó);
        return true;
    }
    Crianças NodeList = node.getChildNodes ();
    if (crianças! = null) {
        int len = children.getLength ();
        for (int i = 0; i <len; i + +) {
            if (getText (sb, children.item (i), elemento)) {
                return true;
            }
        }
    }
    return false;
}

public static void main (String args []) throws Exception {
    Handler = new NekoHTMLHandler NekoHTMLHandler ();
    org.apache.lucene.document.Document doc = handler.getDocument (
        new FileInputStream (new File (args [0])));
    System.out.println (doc);

}
}

```

- b NekoHTML oferece diversos parsers HTML. Nesta implementação usamos NekoHTML de DOMFragmentParser, Que é capaz de processar até mesmo incompleta Documentos HTML.
- A implementação de DocumentHandler's getDocument método leva o HTML
- c documento como InputStream, Usa API NekoHTML para analisá-lo em uma árvore DOM, e em seguida, puxa os valores necessários textual da árvore.

- d Nós criamos uma instância em branco do Xerces " DocumentFragment classe que nós vamos depois pop-  
Ulate com dados DOM.
- e A chamada para parser NekoHTML de processos a dada InputStream e suas lojas de DOM representação no espaço em branco DocumentFragment instância que criamos anteriormente.
- f Extraímos o texto da <title> elemento no DOM dada NÓ chamando um versão do genérico getText método. O valor textual é armazenado na especi-  
cadas StringBuffer.  
Reciclagem dos StringBuffer não é necessário, mas fazemos assim mesmo, só para ser
- g agradável.
- h Usando a outra variante do genérico getText método, extrair todos os Texto HTML documento.
- O genérico e recursiva getText método é usado para extrair valores textuais de
- Eu todos os DOM Nós que contém texto. O getText (StringBuffer, Node) método puxa todos os dados textual que encontra no documento HTML. Fá-lo, chamando-se recor-  
rivamente, uma vez que atravessa a árvore DOM e texto coleta de todo o texto DOM Nós em o caminho. Porque usamos esta versão do getText método para obter o corpo de nosso documento HTML de exemplo, acabamos por recolher todos os dados textual do doc-  
que este documento, não apenas que a encontrada entre os <body> e </ Body> elementos. Esta é uma outra variante do genérico e recursiva getText método. Este,  
no entanto, limita-se a DOM Nós com o nome fornecido. Usamos este método para
- j extrair o texto entre <title> e </ Title> elementos.

NOTA Embora nós lhe mostramos como usar seu parser DOM, você deve estar NekoHTML ciente de que também oferece um analisador SAX HTML.

Você já sabe como analisar HTML, o formato mais popular na web.  
Embora o HTML é o formato de arquivo dominante web, documentos do Microsoft Word ainda regra em ambientes corporativos. Vamos olhar como analisá-los.

## 7,5 indexação de um documento do Microsoft Word

Goste ou não, praticamente todos os negócios na Terra usa o Microsoft Word.<sup>2</sup> Se você fosse para imprimir todos os documentos do MS Word na existência e empilhá-los em cima de cada outro, você poderia provavelmente alcançar distantes planetas do nosso sistema solar. Como você

perfurar como uma pilha grande de árvores mortas para encontrar algo que você está procurando?

Em vez de qualquer coisa impressão, você leia a secção seguinte e aprender a documentos do Word analisar MS e torná-los pesquisáveis com Lucene.

<sup>2</sup> Dolorosamente, mesmo este livro foi escrito no Microsoft Word.

Ao contrário de todos os outros formatos de documentos abordados neste capítulo, o formato do MS

Documentos Word é proprietária. Em outras palavras, a Microsoft Corporation mantém o formato de um segredo exata, o que torna difícil para os outros para escrever aplicações para ler e escrever documentos no formato MS Word. Felizmente, vários projetos de código aberto tornou sua meta de superar este obstáculo. Nesta seção, você verá como usar ferramentas criadas por dois projetos como: Jakarta POI e extratores de texto TextMining.org.

### 7.5.1 Usando POI

POI é um projeto Jakarta, você pode encontrá-lo em <http://jakarta.apache.org/poi>. É um projeto altamente ativo cujo objetivo é fornecer uma API Java para manipulação de vários formatos de arquivo baseado em Microsoft OLE do formato de documento 2 Compound. Assim, POI permite extrair dados textuais de documentos do Microsoft Word, bem como Excel e outros documentos, usando o OLE 2 Compound Document formato.

No exemplo apresentado na listagem 7.9, usamos uma classe única POI, `WordDocument`,

para extrair texto de um documento Word amostra Microsoft; nós então usar o texto para preencher um Lucene Documento exemplo. Além de o conteúdo do documento, Documentos do Microsoft Word também possuem alguns dados meta-, tais como o documento somam o nome do autor, embora o nosso exemplo não extrair essa meta-dados, certamente você pode utilizar o POI para isso, se você precisa de documento de índice meta-dados, também.

#### Listagem 7.9 POI DocumentHandler análise de documentos do Microsoft Word

```
POIWordDocHandler public class {DocumentHandler

    Documento público getDocument (InputStream is)
    throws DocumentHandlerException

    String BodyText = null;

    try {
        WordDocument wd = new WordDocument (is);
        DocTextWriter StringWriter StringWriter = new ();
        wd.writeAllText (PrintWriter novo (docTextWriter));
        docTextWriter.close ();
        BodyText docTextWriter.toString ();
    }
    catch (Exception e) {
        throw new DocumentHandlerException (
            "Não é possível extrair o texto de um documento do Word", e);
    }

    if ((BodyText! = null) & & (bodyText.trim () comprimento. ()> 0)) {
        Documento doc = new Document ();
        
```

**b** método `getDocument`

**c** Extrato textual dados do MS Documento do Word

```
        doc.add (Field.UnStored ("corpo", BodyText));
        retorno doc;
    }
    return null;
}

public static void main (String [] args) throws Exception {
    Manipulador POIWordDocHandler POIWordDocHandler = new ();
    Documento doc = handler.getDocument (
        new FileInputStream (new File (args [0])));
    System.out.println (doc);
}
```

- b Este é o `DocumentHandler's getDocument` método, ao qual passamos o MS Documento do Word `InputStream`.
- c POI faz a extração de texto simples. Sua `WordDocument classe` prontamente leva uma referência ao `InputStream` de um documento do Microsoft Word e nos permite extrair o texto escrevendo-o a um `Escriptor classe`. Uma vez que precisamos do texto em uma `Corda` variável, use a combinação de `StringWriter e PrintWriter` para obter tex-do documento tual valor. Qualquer estrutura é descartada. Como os outros exemplos neste capítulo, salvar esses dados em um `Campo corpo`.

Simples, não é? Acredite ou não, extractores de texto TextMining.org, descrito a seguir, tornar esta tarefa ainda mais simples.

### 7.5.2 Usando TextMining.org 's API

A API TextMining.org fornece uma interface alternativa para o POI Jakarta API, fazendo com que a extração de texto do Microsoft Word documentos uma brisa. É interessante notar que Ackley Ryan, o autor do texto TextMining.org extractores, também é um dos desenvolvedores do projeto Jakarta POI. Além do mais simples API, você deve estar ciente das seguintes vantagens que o TextMining.org API tem mais POI:

- Esta biblioteca é otimizado para extração de texto. POI não é.
- A biblioteca TextMining.org suporta extrair texto de Word 6 / 95, enquanto POI não.
- A biblioteca TextMining.org não extraí o texto excluído que ainda está presente no documento para fins de revisão de marcação. Por outro lado, POI não lidar com isso.

Listagem 7,10 mostra o quanto fácil é usar o toolkit TextMining.org: Demora apenas uma linha!

**Listagem 7,10 TextMining.org DocumentHandler para documentos do Microsoft Word**

```
TextMiningWordDocHandler public class {DocumentHandler

    Documento público getDocument (InputStream is)           b  método getDocument
    {throws DocumentHandlerException

        String BodyText = null;
        try {
            . BodyText = new WordExtractor () extractText (is);
        }
        catch (Exception e) {
            throw new DocumentHandlerException (
                "Não é possível extrair o texto de um documento do Word", e);
        }

        if ((BodyText! = null) & & (bodyText.trim () comprimento. ()> 0)) {
            Documento doc = new Document ();
            doc.add (Field.UnStored ("corpo", BodyText));
            retorno doc;
        }
        return null;

    }

    public static void main (String [] args) throws Exception {
        Manipulador TextMiningWordDocHandler =
            nova TextMiningWordDocHandler ();
        Documento doc = handler.getDocument (
            new FileInputStream (new File (args [0])));
        System.out.println (doc);
    }

}
```

b método getDocument

c Extraindo texto primas  
a partir de InputStream

- b Este é o DocumentHandler's getDocument método ao qual passamos o MS Word documento InputStream.
- c TextMining.org's API simples exige que lidar com apenas uma única classe, WordExtractor, é um único método de que classe, extractText (InputStream), que puxa todo o texto do documento do Word da Microsoft em uma string. Uma vez que temos uma referência para o texto do documento, que adicioná-la a uma instância de uma Lucene Documento o mesmo forma que temos vindo a fazer em outros exemplos neste capítulo.

Em seguida, você vai aprender como analisar documentos RTF. Embora tais documentos não são quase tão populares como documentos do Microsoft Word, o formato RTF é atraente porque oferece plataforma e portabilidade de aplicações.

## 7.6 indexação de um documento RTF

Apesar de que precisávamos bibliotecas de terceiros para extrair texto de todos os rich media documents abordados neste capítulo, para documentos em Rich Text Format (RTF), podemos usar as classes que fazem parte da distribuição padrão do Java. Eles se escondem nas `javax.swing.text` e `javax.swing.text.rtf` pacotes, mas entregar o prometido funcionalidade quando utilizado, como mostrado na listagem 7.11.

Listagem 7.11 DocumentHandler usando built-in do Java RTF texto extractor

```
JavaBuiltInRTFHandler public class {DocumentHandler

    Documento público getDocument (InputStream is)           b     método getDocument
    {throws DocumentHandlerException

        String BodyText = null;
        DefaultStyledDocument styledDoc DefaultStyledDocument = new ();
        try {
            nova () ler (é, styledDoc, 0);
            BodyText = styledDoc.getText (0 styledDoc.getLength, ());
        }Extrair texto
        catch (IOException e) {
            throw new DocumentHandlerException (
                "Não é possível extrair o texto de um documento RTF", e);
        }
        catch (BadLocationException e) {
            throw new DocumentHandlerException (
                "Não é possível extrair o texto de um documento RTF", e);
        }

        if (BodyText! = null) {
            Documento doc = new Document ();
            doc.add (Field.UnStored ("corpo", BodyText));
            retorno doc;
        }
        return null;

    }

    public static void main (String [] args) throws Exception {
        Manipulador JavaBuiltInRTFHandler JavaBuiltInRTFHandler = new ();
        Documento doc = handler.getDocument (
            new FileInputStream (new File (args [0])));
        System.out.println (doc);
    }

}
```

Instância de `javax.swing.text.Document`

c

d

e

- b Este é o `DocumentHandler's getDocument` método, a que passamos a RTF documento `InputStream`.
- c Nós instanciar a implementação específica do `javax.swing.text.Document` interface e depois usá-lo para ler o conteúdo do documento RTF.
- d Para extrair texto de um documento RTF, usamos Java built-in `RTFEditorKit` classe.
- e Com a sua `ler` método, podemos ler o nosso documento RTF em uma instância de `Padrão-StyledDocument`.  
Para obter todo o texto do documento RTF que lê-lo na íntegra a partir do `DefaultStyledDocumento`. Essa classe implementa um `javax.swing.text.Document` interface, que nos permite obter qualquer intervalo de caracteres do documento. Estamos, naturalmente, interessado em todos os dados textual, de modo que especificar o intervalo desde o primeiro até o último char-  
cional. Especificando diferentes offset e comprimento que poderia ter extraído apenas um parte do texto todo.

Depois de todo o texto foi retirado do documento RTF, vemos a familiar bloco de código que adiciona o texto extraído para um Lucene Documento como `Field.UnStored`.

Nosso último `DocumentHandler` vai lidar com arquivos de texto simples. Vamos dar uma olhada.

## 7.7 indexação de um documento de texto simples

Finalmente, vamos implementar um `DocumentHandler` para documentos de texto simples, mostrado na listagem 7.12. Esta é a classe mais simples neste capítulo, e requer muito pouco explicação, porque ele usa apenas o familiar, Java core classes, não tem terço partido dependências.

### Listagem 7.12 Plain-texto DocumentHandler utilizando apenas o núcleo classes Java

```
PlainTextHandler public class {DocumentHandler

    Documento público getDocument (InputStream is)
    {throws DocumentHandlerException

        String BodyText = "";

        try {
            BufferedReader br =
                new BufferedReader (new InputStreamReader (is));
            String linha = null;
            while ((linha = br.readLine ()) != null) {
                BodyText += linha;
            }
            br.close ();
        }
        catch (IOException e) {
```

**b** método `getDocument`

Leia `InputStream`  
uma linha de cada vez

```
        throw new DocumentHandlerException (
            "Não é possível ler o documento de texto", e);
    }

    if (! bodyText.equals ("")) {
        Documento doc = new Document ();
        doc.add (Field.UnStored ("corpo", BodyText));
        retorno doc;
    }

    return null;
}

public static void main (String [] args) throws Exception {
    Manipulador PlainTextHandler PlainTextHandler = new ();
    Documento doc = handler.getDocument (
        new FileInputStream (new File (args [0])));
    System.out.println (doc);
}
```

- b Este é o `DocumentHandler`'s `getDocument` método, ao qual passamos o puro documento de texto `InputStream`.
- c Este `DocumentHandler` aplicação lê o documento de texto simples de uma linha em um tempo e acrescenta cada linha a um `Corda`, O que acaba com a plena contenda do documento original. Este texto é, então, indexada como uma `Field.UnStored` chamado `corpo`.

Como dissemos na introdução deste capítulo, nosso objetivo é criar um pequeno quadro trabalho para análise e indexação de documentos de vários formatos. Todos os `Documento-Manipulador` implementações apresentadas até agora são o primeiro passo nessa direção. Nós agora passar para o nosso próximo passo, onde as coisas começam a ficar interessantes: Nós vamos começar a colagem coisas juntas, eo quadro vai começar a tomar forma.

## 7.8 Criando uma estrutura de tratamento de documentos

Até agora, neste capítulo, temos apresentado soluções independentes: individual `Documento-Manipulador` implementações para analisar vários formatos de documentos comuns. Porque todas as classes temos apresentado implementar nossa genéricos `DocumentHandler` interface, definida no início deste capítulo, é fácil criar um mínimo de quadro para o tratamento e indexação de documentos de vários tipos, sem preocupação sobre os formatos de arquivos individuais.

Para a nossa infra-estrutura existente, que consiste na `DocumentHandler` interface e acompanhante `DocumentHandlerException`, Agora adicionar um novo `FileHandler` interface e `FileHandlerException`. Além disso, vamos implementar a `FileHandler` interface com uma classe chamada `ExtensionFileHandler`. Tabela 7.2 resume as componentes do framework.

Tabela 7.2 Classes Java que compõem um quadro de arquivo-indexação

Java classe	Propósito
<code>DocumentHandler</code>	Define o <code>getDocument (InputStream)</code> método implementado por todos os analisadores documento
<code>DocumentHandlerException</code>	Exceção verificada jogados por todos os parsers em caso de erro
<code>FileHandler</code>	Define o <code>getDocument (File)</code> método implementado por <code>ExtensionFileHandler</code>
<code>FileHandlerException</code>	Exceção verificada jogado por concreto <code>FileHandler</code> implementações
<code>ExtensionFileHandler</code>	Implementação de <code>FileHandler</code> que atua como uma fachada para a individual <code>DocumentHandler</code> implementações, invocando o aproparer comeu com base na extensão do arquivo passado para ele através do <code>getDocument (File)</code> método

Finalmente, criamos uma `FileIndexer` de linha de comando do aplicativo que usa todos os componentes listados na figura 7.1, bem como todos os analisadores apresentados neste capítulo.

Este ready-to-use aplicativo pode recursivamente percorrer diretórios do sistema de arquivos, ao longo do caminho arquivos de indexação em todos os formatos que cobrimos. Figura 7.1 mostra a quadro depois de tudo ter sido juntos.

Com este quadro de alto nível em mente, vamos dar uma olhada mais detalhada no componentes individuais que compõem o sistema.

### 7.8.1 interface FileHandler

Até agora você deve estar familiarizado com o `DocumentHandler` e um número de seus implementações. `FileHandler`, Apresentado na listagem 7.13, é uma interface simples, muito semelhante ao de um `DocumentHandler`. No entanto, ao contrário `DocumentHandler`, Que expõe o genérico `InputStream` como o tipo de entrada aceitável, o `FileHandler` interface define `Arquivo` como o seu tipo de entrada, tornando assim mais fácil trabalhar com a classes de nível superior que lidam com `Arquivo` objetos.

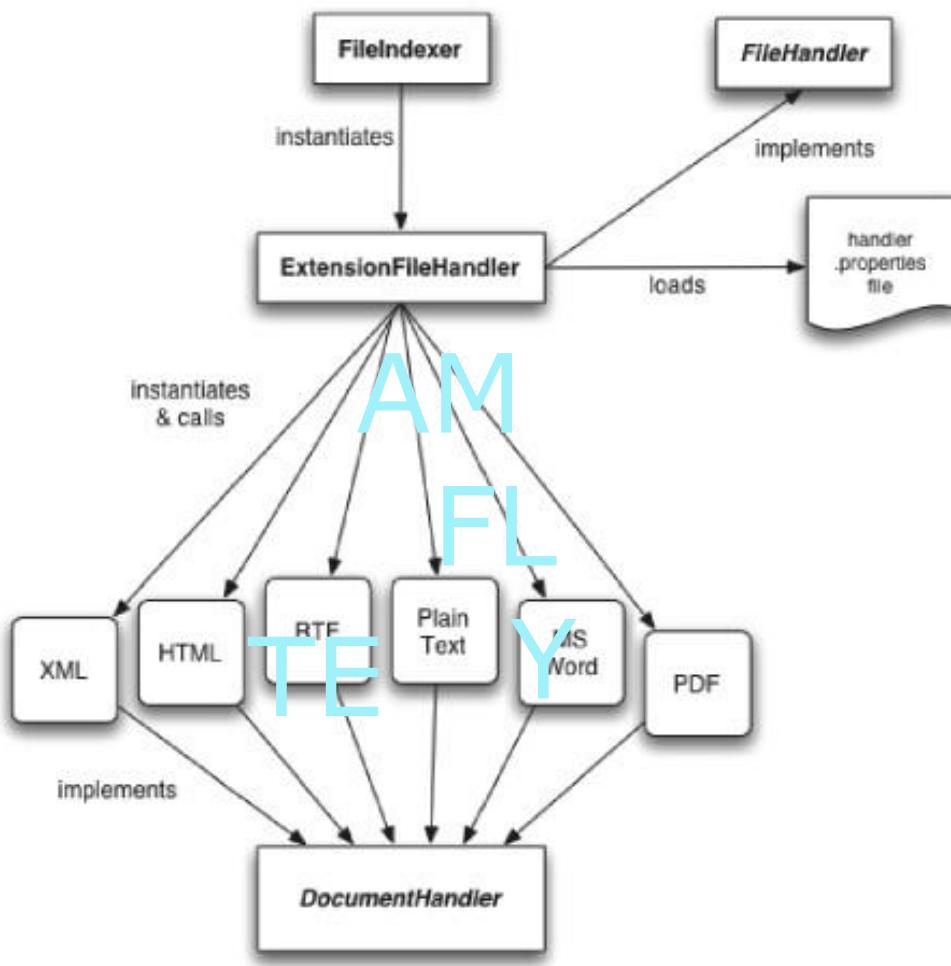


Figura 7.1 A estrutura do quadro de análise de documentos, combinada com um arquivo-indexação aplicativo que usa-lo

**Listagem 7.13 FileHandler interface para a criação de Lucene Documentos a partir de arquivos**

```

public interface FileHandler {

    /**
     * Cria um documento Lucene de um arquivo.
     * Esse método pode retornar null </code> </code>.
     *
     * @ Param file o arquivo para converter em um documento
     * @ Return uma instância de índice para pronta do Documento
     */
    GetDocument documento (arquivo)
        joga FileHandlerException;
    }

    Converter objeto File para
    Documento Lucene
    <-->

```

Dada uma instância de uma `Arquivo` classe, uma implementação do `FileHandler` interface retorna uma instância de um povoado `Lucene Documento` para o chamador. Cada `FileHandler` implementação envolve todas as exceções que ele encontra e relança-los envolto em um `FileHandlerException`. Uma classe tão chata como a maioria das classes outra exceção.

Em vez de listar que classe de exceção aqui, vamos olhar para `ExtensionFileHandler`.

### 7.8.2 ExtensionFileHandler

`ExtensionFileHandler`, Mostrada na listagem 7.14, é a nossa única implementação do `FileHandler` interface. A implementação do `getDocument (File)` método usa a extensão do arquivo de dados para deduzir o tipo do arquivo e chamar o apro-implementação de analisador adequado. Porque todos os nossos parsers implementar o comum `DocumentHandler` interface, o `ExtensionFileHandler` pode passá-las cegamente a `Arquivo` objeto envolto em um `FileInputStream`. Que parsers sabem como lidar.

Listagem 7.14 `ExtensionFileHandler`: um `FileHandler` com base em extensões de arquivo

```

/ **
 * A implementação FileHandler que a responsabilidade de delegados
 * Implementação DocumentHandler adequadas, com base em um arquivo
 * Extensão.
 */
ExtensionFileHandler public class {FileHandler
Propriedades privadas handlerProps;

ExtensionFileHandler público (Properties props) throws IOException {
    handlerProps = adereços; MapaBExtensão de arquivo
}

getDocument documento público (arquivo)
{throws FileHandlerException

Documento doc = null;
String nome = file.getName ();Extrato
int = dotIndex name.indexOf (".");nome do arquivo
if (dotIndex (> 0) & & dotIndex (<name.length ())) {extensão
String ext = name.substring (dotIndex + 1, name.length ());
String = handlerClassName handlerProps.getProperty (ext);

try {nome da classe
    HandlerClass classe = Class.forName (handlerClassName);
    Manipulador DocumentHandler =
        (DocumentHandler) handlerClass.newInstance ();
    retorno handler.getDocument (FileInputStream novo (arquivo));
}
catch (ClassNotFoundException e) {
    throw new FileHandlerException (

```

Olhe para cima parser **C**

Passar arquivo para implementação de analisador **E**

```

        "Não é possível criar instância:"
        + HandlerClassName, e);

    }
    catch (InstantiationException e) {
        throw new FileHandlerException (
            "Não é possível criar instância:"
            + HandlerClassName, e);
    }
    catch (IllegalAccessException e) {
        throw new FileHandlerException (
            "Não é possível criar instância:"
            + HandlerClassName, e);
    }
    catch (FileNotFoundException e) {
        throw new FileHandlerException (
            "File not found:"
            + File.getAbsolutePath (), e);
    }
    catch (DocumentHandlerException e) {
        throw new FileHandlerException (
            "O documento não pode ser manipulador:"
            + File.getAbsolutePath (), e);
    }
}

}
return null;
}

public static void main (String [] args) throws Exception {
    if (args.length <2) {
        usage ();
        System.exit (0);
    }

Properties props = new Properties ();
props.load (new FileInputStream (args [0]));

ExtensionFileHandler fileHandler =
    nova ExtensionFileHandler (props);
Documento doc = (new File (args [1])) fileHandler.getDocument;

}

uso de private static void () {
    System.err.println ("Uso: java"
        + ExtensionFileHandler.class.getName ()
        + "/ Caminho / para / properties / caminho / para /
        documento");
}

}

```



Carga  
arquivo de propriedades

- b) O Propriedades exemplo, extensões de arquivos de mapas para o DocumentHandler classes capa-ble de análise de arquivos com essas extensões.

- c Para extrair a extensão do arquivo, olhamos para o último ponto no nome do arquivo e pegar tudo, desde que compensado ao final do nome do arquivo.
- d Nós usamos a extensão do arquivo extraído e os Propriedades instância a instanti- comeu a adequada DocumentHandler. Depois que instanciar dinamicamente nosso DocumentHandler, Nós passá-lo a Arquivo envolto
- e por FileInputStream para análise. O arquivo de propriedades especificadas na linha de comando é carregado no Propriedades exemplo.
- f

Há diversas peças importantes nesta aplicação digna de nota. O primeiro coisa a observar é que o construtor só é o que leva uma instância de uma Propriedades classe. Isto é importante porque este FileHandler precisa de uma configuração que mapeia diferentes extensões de arquivo para diferentes DocumentHandler classes. Aqui é um exemplo de arquivo de propriedades. Nós mapeou várias extensões de arquivos comuns para vários DocumentHandler implementações apresentadas anteriormente neste capítulo:

```
txt    = lia.handlingtypes.text.PlainTextHandler
html   = lia.handlingtypes.html.JTidyHTMLHandler
rtf    = lia.handlingtypes.rtf.JavaBuiltInRTFHandler
doutor = lia.handlingtypes.msdoc.TextMiningWordDocHandler
pdf    = lia.handlingtypes.pdf.PDFBoxPDFHandler
xml    = lia.handlingtypes.xml.DigesterXMLHandler
```

Olhando para além do construtor e na getDocument (File) método, você pode ver que o código extrai a extensão do arquivo e usa-lo para criar a apro-adequado DocumentHandler. Após consulta do Propriedades exemplo, situado no con- tor. A correspondência DocumentHandler é dinamicamente instanciado, que é possível porque todos os DocumentHandler implementações conter um público padrão construtor. Finalmente, o arquivo de entrada é convertido para um FileInputStream. Uma subclasse de InputStream, E passado para o getDocument (InputStream) método definido no DocumentHandler interface. Um número de exceções que nós estamos pegando estão relacionados a instanciação de DocumentHandler implementações usando o reflexo Java.

Você pode optar por chamar ExtensionFileHandler de outra classe Java, mas nós incluiu a principal método, que permite que você execute esta classe a partir do comando linha também. Dois argumentos de linha de comando deve ser especificado: o caminho para o propriedades do arquivo que mapeia extensões de arquivo para DocumentHandlers, E um caminho para o arquivo que precisa ser processado.

O principal método é apenas um método de conveniência. O poder real de -Extensão FileHandler é evidente quando ele é chamado de programação e que é exatamente o que fazemos a partir do FileIndexer aplicação, descrito na próxima seção.

### 7.8.3 aplicação FileIndexer

Listagem 7.15 mostra uma classe chamada `FileIndexer`. O produto final deste capítulo. Ele une todos os componentes descritos neste capítulo em uma linha de comando

aplicação capaz de percorrer recursivamente os diretórios do sistema de arquivos e indexação todos os arquivos encontrados ao longo do caminho, enquanto temos um analisador capaz de lidar

seu formato de arquivo. `FileIndexer` pode lembrá-lo do `Indexador` aplicação a partir seção 1.4. Ambos recursivamente percorrer os diretórios do sistema de arquivos. No entanto, enquanto que `Indexador` limita-se a indexação de texto simples arquivos, `FileIndexer` pode analisar

e índice de todos os formatos de documentos abordados neste capítulo.

Listagem 7.15 `FileIndexer`: um indexador do sistema de arquivos recursiva

```

/ **
 * A indexador de arquivos de forma recursiva capaz de indexar uma árvore de
diretórios.
 */
FileIndexer public class
{
    protegidos FileHandler fileHandler;

    FileIndexer público (Properties props) throws IOException {
        FileHandler = new ExtensionFileHandler (props);ExtensionFileHandler
    }

    índice public void (escritor IndexWriter, arquivo File) throws FileHandlerException
        c     método do índice

        if (file.canRead ()) {
            if (file.isDirectory ()) {
                String [] arquivos = file.list ();
                if (arquivo! = null) {
                    for (int i = 0; i < files.length <; i + +) {
                        índice (escritor, new File (arquivo, arquivos [i]));
                    }
                }
            }
            else {
                System.out.println ("Indexação" file +);
                try {
                    Documento doc = fileHandler.getDocument (arquivo);
                    if (doc! = null) {
                        Adicionar
                        retornouwriter.addDocument (doc);
                    }
                }
                catch (IOException e) {
                    Documento Lucene} e     Mão arquivos off para
                    para o índiceelse {
                        System.err.println ("Não é possível lidar com"
                            + File.getAbsolutePath () + "; pular");
                }
            }
        }
    }
}

```

```

        System.err.println ("Não é possível indexar"
            + File.getAbsolutePath () + "; pular ("
            + E.getMessage () + ")");
    }
}
}

public static void main (String [] args) throws Exception {
    if (args.length <3) {
        usage ();
        System.exit (0);
    }

Properties props = new Properties ();
props.load (new FileInputStream (args [0]));
g Propriedades carga especificada
na linha de comando
Diretório dir = FSDirectory.getDirectory (args [2], true);
índice OpenAnalizador
analisador = new SimpleAnalyzer ();
Escritor IndexWriter IndexWriter = new (dir, analisador, true);
h

Indexador FileIndexer FileIndexer = new (props);
Eu Criar FileIndexer
exemplo

longo start = new Date () getTime ();
indexer.index (escritor, new File (args [1]));
writer.optimize ();
writer.Close ();
j Primeira chamada para o método de
índice
Optimize
final longo = new Date () getTime (); 1
índice, perto
escritor índice
System.out.println ();
IndexReader reader = IndexReader.open (dir);
System.out.println ("Documentos indexados:" + reader.numDocs ());
System.out.println ("Tempo total:" + (final - inicial) + "ms");
reader.Close ();User-friendly
1!
sumário
}
}

uso de private static void () {
    System.err.println ("Uso: java"
        + FileIndexer.class.getName ()
        + " / Caminho para / properties / caminho / para / arquivo / ou
        / directory"
        + " / Caminho / para / index");
}
}

```

- b** FileIndexer tem um construtor privado e um construtor público que leva uma instância de Propriedades classe como um parâmetro para as mesmas razões que ExtensionFileHandler exigiu um Propriedades exemplo. Além disso, olhando para Arquivo IndexadorConstrutor público 's revela que o especificado Propriedades são passados para O ExtensionFileHandler construtor.

- c A carne de `FileIndexer` está no `index (File IndexWriter)` método. Isto é onde vamos implementar a indexação de arquivos.
- d Se a instância especificada do `Arquivo` classe representa um diretório do sistema de arquivo, o `index (File IndexWriter)` método chama-se recursivamente.
- e Eventualmente, porém, `index (File IndexWriter)` chama a si mesma com um `Arquivo` instância que
- representa um arquivo real. Nesse ponto, o `ExtensionFileHandler` entra em jogo, porque o controle de execução é passado para ele com uma chamada para a sua `getDocument (File)` método.
- f A chamada para `getDocument (File)` retorna um Lucene povoadas `Documento`, Se um dos `o DocumentHandler` implementações foi capaz de analisar o arquivo especificado. Se não `DocumentHandler` foi capaz de processar o arquivo, um Lucene null `Documento` é retornado. Assim, vamos verificar o objeto retornado para null, e adicioná-lo ao Lucene índice apenas se o `Documento` não é nulo.
- g O arquivo de propriedades especificadas na linha de comando é carregado em uma instância de `Propriedades`.
- h O índice para o qual todos `Arquivos` convertido em Lucene `Documentos` são adicionados é aberto para escrita com Lucene `IndexWriter` classe.
- i A instância do `FileIndexer` realizar passagem de diretório e indexação de arquivos é criadas com a `Propriedades` que venham a ser passado para `ExtensionFileHandler`.
- j Eu A primeira chamada para `FileIndexer'S índice` método começa diretório e processamento de arquivos.
- k Nós passá-lo a `IndexWriter` que anteriormente abertos, eo ponto de partida o nome do arquivo ou diretório especificado na linha de comando.
- l Uma vez que tenha atravessado a árvore de diretórios todo, o recursiva `índice` método retorna o controle de execução para o chamador. É então da responsabilidade do chamador para lidar com a `IndexWriter` corretamente, fechá-lo, opcionalmente, otimizando-o primeiro.
- m Nosso resumo user-friendly informa o utilizador sobre o número de arquivos indexados e do tempo necessário.

1!

#### 7.8.4 Usando FileIndexer

O `FileIndexer` classe inclui um principal método que pode ser usado para invocar o classe a partir da linha de comando e arquivos recursivamente índice em uma árvore de diretórios dado.

Para executar `FileIndexer` na linha de comando, passá-lo um caminho para o arquivo de propriedades

como o primeiro argumento, semelhante ao mostrado no exemplo a seguir, como um segundo argumento, passá-la um caminho para uma árvore de diretórios ou um único arquivo que você quer

para o índice: `handlingtypes.framework.FileIndexer`

```
⇒ ~ / Handler.properties ~ / ~ dados índice /
Indexação / home / otis / data / FileWithoutExtension
Não pode lidar com / home / otis / data / FileWithoutExtension; pulando
```

```

Indexação / home / otis / data / HTML.html
Indexação / home / otis / data / MSWord.doc
Indexação / home / otis / data / PlainText.txt
Indexação / home / otis / data / PowerPoint.ppt
Não pode lidar com / home / otis / data / PowerPoint.ppt; pulando
Indexação / home / otis / data / RTF.rtf
Indexação / home / otis / data / addressbook-entry.xml

```

Documentos indexados: 6  
Tempo total: 3046 ms

Como ele funciona através de uma árvore de diretórios, `FileIndexer` imprime informações sobre o seu progresso. Você pode ver aqui que ele indexa apenas os arquivos com as extensões que temos mapeados para específicos `DocumentHandlers`; Todos os outros arquivos são ignorados.

### 7.8.5 inconvenientes `FileIndexer`, e como estender o quadro

Este quadro tem uma óbvia, embora falha, menor: Assume-se que o arquivo extensões não mentem, e isso requer que todos os arquivos de tê-los. Por exemplo, ele assume que um arquivo de texto simples sempre tem uma extensão de arquivo txt, e nenhum outro;. doc que o.

extensão é reservado para documentos do Microsoft Word e assim por diante.

O quadro que desenvolvemos neste capítulo inclui analisadores que podem lidar com os seguintes tipos de entrada:

- XML
- PDF
- HTML
- Microsoft Word
- RTF
- Texto simples

Então, o que você faria se você precisa para indexar e fazer arquivos de searchable de um tipo que

nossa estrutura não manipula? Você estender o quadro, é claro! Mais precisamente, siga estes passos:

- 1 Escrever um parser para o tipo de arquivo desejado e implementar o `DocumentHandler` interface.
- 2 Adicione a sua classe para o analisador `handler.properties` arquivo, mapeá-lo para o extensão de arquivo apropriado.
- 3 Continue usando `FileIndexer` como mostrado.

Isto leva-nos para a próxima seção, onde você pode encontrar uma lista de documentos para análise de ferramentas que você pode usar, além dos apresentados neste capítulo.

## Outros 7,9 texto extração de ferramentas

---

Neste capítulo, temos apresentado a extração de texto a partir de, e indexação de, os mais comuns formatos de documentos. Escolhemos as ferramentas que são os mais populares entre desenvolvedores, ferramentas que ainda estão sendo desenvolvidos (ou pelo menos mantida), e ferramentas que são fáceis de usar. Todas as bibliotecas que temos apresentado estão disponíveis gratuitamente. Lá também, naturalmente, uma série de outras ferramentas livres e comerciais que você pode usar; vários que conhecemos estão listados na tabela 7.3.

Tabela 7.3 Ferramentas para análise de diferentes formatos de documento, que pode ser usado com Lucene para fazer documentos nesses formatos pesquisáveis

Formato de documento	Ferramenta	Onde fazer o download
PDF	Xpdf	<a href="http://www.foolabs.com/xpdf/">http://www.foolabs.com/xpdf/</a>
	JPedal	<a href="http://www.jpedal.org/">http://www.jpedal.org/</a>
	Étimo PJ	<a href="http://www.etymon.com/">http://www.etymon.com/</a>
	PDF Text Stream	<a href="http://snowtide.com/home/PDFTextStream">http://snowtide.com/home/PDFTextStream</a>
	Multivalente	<a href="http://multivalent.sourceforge.net/">http://multivalent.sourceforge.net/</a>
XML	JDOM	<a href="http://www.jdom.org/">http://www.jdom.org/</a>
	Flautim	<a href="http://piccolo.sourceforge.net/">http://piccolo.sourceforge.net/</a>
HTML	HTMLParser	<a href="http://htmlparser.sourceforge.net/">http://htmlparser.sourceforge.net/</a>
	Multivalente	<a href="http://multivalent.sourceforge.net/">http://multivalent.sourceforge.net/</a>
Microsoft Word	Antiword	<a href="http://www.winfield.demon.nl/">http://www.winfield.demon.nl/</a>
	OpenOffice SDK	<a href="http://www.openoffice.org/">http://www.openoffice.org/</a>
Microsoft Excel	POI	<a href="http://jakarta.apache.org/poi">http://jakarta.apache.org/poi</a>

### 7.9.1 Documento de gerenciamento de sistemas e serviços

Além de bibliotecas individuais que você pode usar para implementar documento parsing e indexação da forma como fizemos neste capítulo, alguns pacotes de software livre e serviços já fazem isso e, curiosamente, contam com Lucene para lidar com indexação de documentos:

- DocSearcher (<http://www.brownsite.net/docsearch.htm>) é descrita por seus autor da seguinte forma: "DocSearcher usa o Open Source Lucene e POI Apache APIs, bem como o Open Source API Caixa de PDF para fornecer busca

capacidades para HTML, MS Word, MS Excel, RTF, PDF, Open Office (e Star Office) documentos, e documentos de texto."

- Docco (<http://tockit.sourceforge.net/docco/index.html>) é um pequeno e pessoal sistema de gerenciamento de documentos construída em cima do Lucene. Ele fornece-indexação e busca com Lucene, este último é reforçada usando Formal Técnicas de análise de conceito de visualização. De acordo com a documentação em sua home page, Docco pode lidar com um número de formatos de documentos: texto simples, XML, HTML, PDF, Microsoft Word e Excel, OpenOffice, e StarOffice 6.0, bem como páginas de manual UNIX. Note que a lista não incluir documentos RTF.
- SearchBlox (<http://www.searchblox.com/>) é um componente de pesquisa J2EE que é implementado como uma aplicação web. É controlado e personalizado através de um web interface do navegador, e pode indexar e pesquisar HTML, PDF, Word, Excel, e documentos PowerPoint. Você pode ler um estudo de caso SearchBlox no seção 10.3.
- Simpy (<http://www.simpy.com/>) é um serviço online gratuito criado por um dos autores deste livro. Ele permite que você salve links para seus documentos online, seja que páginas HTML web; PDF, Microsoft Word, RTF ou documentos, ou qualquer outro formato. Além da meta-dados que você pode entrar para cada documento, Simpy vai rastrear e indexar o texto completo de seus documentos, permitindo que você procurá-los de qualquer computador. Seus documentos podem ser mantidos privados ou podem ser compartilhados, permitindo-lhe formar círculos de colaboração online. É claro, todos os indexação e busca é alimentado por Lucene, e algumas porções de o back-end use Nutch (ver o estudo de caso na seção 10.1).

Nova Lucene de gerenciamento de documentos sistemas e serviços, sem dúvida, surgir após este livro vai para impressão. Um bom lugar para procurar Lucene-powered soluções é o Wiki Lucene, bem como SourceForge.

## Resumo 7,10

---

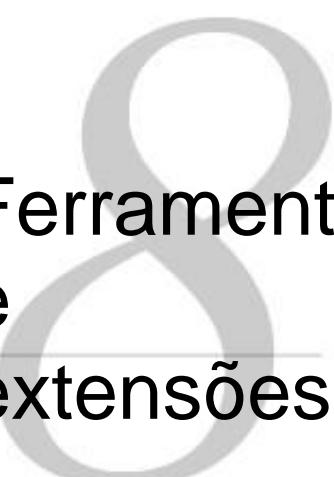
Neste capítulo código-rico, você aprendeu a lidar com vários documentos comuns formatos, a partir do formato onipresente, mas de propriedade do Microsoft Word para o HTML onipresente e aberto. Como você pode ver, qualquer tipo de dados que podem ser convertidas em texto podem ser indexados e fez searchable com Lucene. Se você pode extrair dados textuais de som ou arquivos gráficos, você pode indexar os, também. Por uma questão de fato, 10,6 seção descreve uma abordagem interessante para indexação de imagens JPEG.

Nós usamos um número de parsers livremente disponível para analisar documentos diferentes for-  
esteiras: Xerces e Digestor para XML, JTidy e NekoHTML para HTML, PDFBox  
para PDF, e POI e extratores TextMining.org para documentos do Microsoft Word.  
Para analisar RTF e documentos de texto simples, contamos com classes core Java.

No início do capítulo, definiu-se um `DocumentHandler` interface que nos ajudou a definir o mecanismo de invocação padrão para todos os analisadores de nosso documento. Este, por sua vez, tornou simples para nós, para agrupar todos os analisadores em um pequeno quadro-chave na mão trabalho capaz de analisar recursivamente e indexação de um sistema de arquivos.

O que você aprendeu neste capítulo não se limita a indexação de arquivos armazenados em seu sistema de arquivos local. Você pode usar a mesma estrutura para indexar páginas web, arquivos armazenados em servidores remotos FTP, arquivos armazenados em servidores remotos em seu LAN ou E-mail, WAN de entrada e saída de mensagens instantâneas ou mensagens, ou qualquer coisa outra coisa que você pode se transformar em texto. Sua imaginação é o limite.

AN  
EL  
TE Y



# Ferramentas e extensões

Este capítulo aborda

- A utilização de componentes do Lucene Sandbox
- Trabalhar com ferramentas de terceiros Lucene

Você construiu um índice, mas você pode navegar ou consultá-lo sem escrever código? Absolutely! Neste capítulo, vamos discutir três ferramentas para fazer isso. Você precisa de análise além do que os analisadores de built-in oferece? Analizadores especializados para muitas línguas estão disponíveis no Sandbox do Lucene. Como sobre o fornecimento de Google-destacando como prazo nos resultados da pesquisa? Nós temos que, também!

Este capítulo examina o software de terceiros (não-Jakarta), bem como várias Projetos Sandbox. Jacarta hospeda um repositório CVS separada onde add-ons para Lucene são mantidos. Cuidado deliberado foi tomada com o design do Lucene para manter o código fonte do núcleo coeso ainda extensível. Nós estamos tomando o mesmo cuidado neste livro , mantendo uma separação entre o que é intencional no âmago de Lucene e as ferramentas e as extensões que têm sido desenvolvidos para aumentá-la.

## Jogando em 8.1 Sandbox do Lucene

Em um esforço para acomodar as contribuições crescentes para o projeto Lucene que estão acima e além da base de código do núcleo, um repositório CVS Sandbox foi criado a abrigá-los. O Sandbox está em constante evolução, tornando-se difícil escrever sobre concretamente. Nós vamos cobrir os pedaços estáveis e aludem ao outro interesting bits. Nós encorajamos você, quando você precisar de mais peças Lucene, para contato o repositório Sandbox e se familiarizar com o que está lá, você pode descobrir que uma peça em falta que você precisa. E na mesma linha, se você desenvolveu Lucene peças e quero compartilhar os esforços de manutenção, as contribuições são mais que bem-vindos.

Tabela 8.1 lista o conteúdo mais atual do Sandbox com ponteiros para onde cada é abordado neste livro.

Tabela 8.1 Major componente Sandbox referência cruzada

Área Sandbox	Descrição	Cobertura
analisadores	Analisadores para vários idiomas	Seção 8.3
formiga	Um Ant <index> tarefa	Seção 8.4
db	Berkeley DB aplicação Directory	Seção 8.9
highlighter	Pesquisa destacando trecho resultado	Seção 8.7
javascript	Query Builder e validador para navegadores web	Seção 8.5
lucli	Interface de linha de comando para interagir com um índice	Seção 8.2.1
diverso	Algumas probabilidades e extremidades, incluindo o ChainedFilter	Seção 8.8

continua na página seguinte

Tabela 8.1 Major componente Sandbox referência cruzada (Continuação)

Área Sandbox	Descrição	Cobertura
bola de neve	Família sofisticada de derivações e analisador de embrulho	Seção 8.3.1
WordNet	Utilitário para construir um índice Lucene da WordNet banco de dados	Seção 8.6

Há alguns componentes mais Sandbox do que as que cobrem neste capítulo. Consulte o Sandbox diretamente para cavar em volta e ver qualquer coisas novas desde este foi impresso.

## 8.2 Interagindo com um índice

Você criou um índice ótimo. E agora? Não seria bom procurar o índice e executar consultas ad hoc? Você vai, claro, escrever código Java para integrar Lucene em suas aplicações, e você poderia facilmente escrever código utilidade como JUnit caso de teste, um utilitário de linha de comando, ou um aplicativo web para interagir com o índice. Felizmente, no entanto, alguns utilitários de bom já foi criado para deixar você interage com os índices do sistema de arquivos Lucene. Vamos explorar três utilitários tais, cada um único e ter um tipo diferente de interface em um índice:

- lucli (Lucene Interface de linha de comando)-A CLI que permite ad-hoc consulta e inspeção índice
- Luke (Lucene Índice Toolbox)-A aplicação desktop com usabilidade boa
- LIMO (Lucene Índice de Monitor)-A interface web que permite controle remoto do índice navegação

### 8.2.1 lucli: uma interface de linha de comando

Ao invés de escrever código para interagir com um índice, ele pode ser mais fácil fazer um pouco de linha de comando sapateado para ad-hoc pesquisas ou para obter uma explicação rápida de um pontuação. O Sandbox contém a interface de linha de comando Lucene (lucli) contribuição de Dror Matalon. Lucli fornece uma capacidade de readline opcional (on apoiar os sistemas operacionais), que permite percorrer uma história de comandos e executar novamente um comando introduzido anteriormente para melhorar a sua usabilidade.

Utilizando o índice WordNet vamos construir em secção 8.6 como um exemplo, listando 8.1 demonstra uma sessão interativa.

Listagem 8.1 lucli em ação

```
% Java lucli.Lucli

          Open existentesLucene CLI. Usando diretório: índice
          índice pelo caminho lucli> index .. / WordNet /
index
Lucene CLI. Usando diretório: .. / WordNet / index
Índice tem 39.718 documentos
Todos os campos: [syn, word]
          Realizar os campos indexados: [palavra]
          pesquisalucli> Procura de salto
Procurando por: syn: palavra jump: salto
          Consulta em todos os
Um total documentos correspondentes
          condições
-----
----- 0 score :1.0 -----
syn: startle
syn: start
syn: Primavera
syn: skip
syn: aumento
syn: pára-quedismo
syn: salto
syn: jumpstart
syn: saltar
syn: descarrilar
syn: obrigado
syn: alternativo
palavra: jump
          explicações
lucli#####
          de comandos lucli ajuda>
count: Retorna o número de hits para uma pesquisa.
Exemplo: contar foo
Explico: Explicação que descreve como o documento
marcou contra consulta. Exemplo: explicar foo
help: Exibe a ajuda sobre os comandos.
índice: Escolha um índice lucene diferentes.
Exemplo meu_idx indice
info: Mostra informações sobre o índice Lucene atual.
Exemplo: info
otimizar: Otimizar o índice atual
quit: Quit / sair do programa
pesquisa: Procure o índice atual. Exemplo: pesquisa foo
termos: Mostrar os primeiros 100 termos neste índice.
        Fornecer um nome de campo para
        mostrar apenas termos em um campo específico. Exemplo: os termos
tokens: faz uma pesquisa e mostra o top 10 fichas para cada
        documento.
        Verbose! Exemplo: foo tokens
lucli> cão explicar
          De busca e
Procurando por: syn: palavra cão: dog
          explicar os resultados
Um total documentos correspondentes
```

```
Procurando por: palavra: cão
-----
----- 0 score :1.0 -----
syn: Trilha
syn: faixa
syn: tail
syn: tag
syn: lingueta
syn: hound
syn: calcanhar
syn: frump
syn: firedog
syn: dogtooth
syn: dogiron
syn: retenção
syn: clique
syn: chase
syn: cad
syn: bounder
syn: blackguard
syn: andiron
palavra: cão
Explicação: 10.896413 = fieldWeight (palavra: cão em 262), produto de:
 1.0 = tf (termFreq (palavra: cão) = 1)
 = 10.896413 idf (docFreq = 1)
 1.0 = fieldNorm (campo = palavra, doc = 262)
```

```
#####
#
```

Lucli é relativamente novo para a cena, e como tal ainda tem espaço para evoluir em recursos e apresentação. Tem algumas limitações para nota, mas geralmente eles não desvirtuar a sua utilidade: A versão atual do lucli usa o `MultiFieldQuery-Analisador` de expressões de busca e é codificado para uso `StandardAnalyzer` com o analisador.

### 8.2.2 Lucas: a caixa de ferramentas Índice Lucene

Andrzej Bialecki criado Lucas (encontrado em <http://www.getopt.org/luke/>), um elegante Lucene browser índice. Esta gema oferece uma visão íntima dentro de um sistema de arquivos índice baseado a partir de uma aplicação desktop Java atraente (ver figura 8.1). Nós altamente recomendável ter Luke útil quando você está desenvolvendo com Lucene pois permite ad-hoc consulta e fornece insights sobre os termos e estrutura em um índice.

Lucas se tornou uma parte regular do nosso kit de ferramentas de desenvolvimento Lucene. Sua interface com o usuário conectado permite navegação rápida e experimentação. Lucas



Figura 8.1  
Página de Lucas Sobre

pode forçar um índice para ser desbloqueado quando abrir, otimizar um índice, e também delete undelete e documentos, por isso é realmente apenas para os desenvolvedores ou, talvez, sys-administradores tem. Mas o que é uma ferramenta maravilhosa que é!

Você pode lançar Luke via Java WebStart a partir do site web Lucas ou instalá-lo localmente. É um único arquivo JAR que podem ser lançados diretamente (clicando duas vezes a partir de

um navegador de arquivos do sistema, se seu sistema suporta isso) ou correr `java-jar luke.jar` a partir da linha de comando. A versão mais recente no momento da redação deste texto é

0,5; ele incorpora um comunicado de pré-final do Lucene 1.4. A JAR separado está disponível sem

Lucene embutido, você pode usá-lo se você quiser usar uma versão diferente do Lucene.<sup>1</sup> Claro, a primeira coisa que Lucas precisa é de um caminho para o arquivo de índice, como mostrado na

de seleção de ficheiros de diálogo na figura 8.2.

Interface de Lucas está muito bem interligado de modo que você pode saltar de um ponto de vista

para outro no mesmo contexto. A interface é dividida em cinco abas: visão geral, Pesquise documentos, arquivos e Plugins. No menu Ferramentas fornece opções para optimizar o índice atual, undelete quaisquer documentos marcados para exclusão, e alternar Ressources entre o formato XML e o formato padrão.

Guia de Lucas Visão Geral mostra as peças principais de um índice do Lucene, incluindo o número de campos, documentos e termos (figura 8.3). Os termos de topo em um ou mais campos selecionados são apresentados no "Top termos ranking" painel. Duplo clique um termo abre a guia Documentos para o termo selecionado, onde você pode navegar todos os documentos que contêm esse termo. Botão direito do mouse um termo traz um menu

com duas opções: "Mostrar todos os docs termo" abre a guia Pesquisar para esse termo para que todos os

<sup>1</sup> As questões habituais da versão Lucene e compatibilidade índice de aplicar.

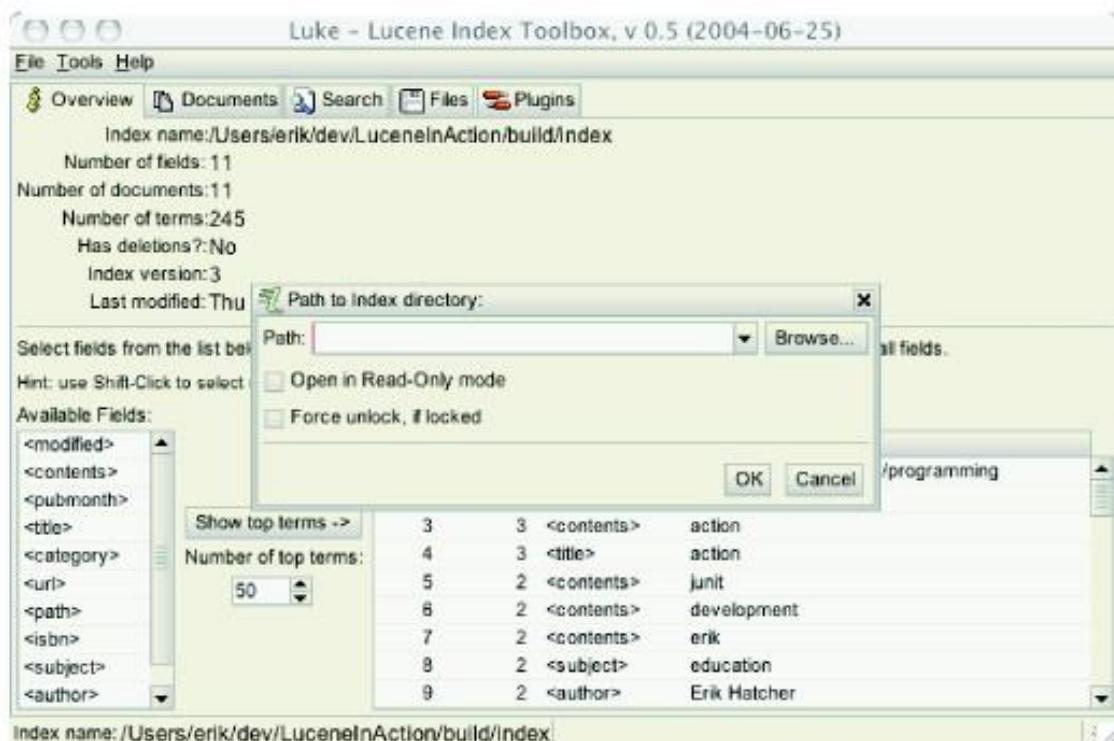


Figura 8.2 Luke: abertura de um índice

documentos aparecem em uma lista, e "Browse docs termo" abre a guia Documentos para o termo selecionado.

#### Navegação de documentos

Na guia Documentos é a tela mais sofisticada de Lucas, onde você pode navegar documentos por número e por prazo (ver figura 8.4). Percorrer por documento número é simples, você pode usar as setas para navegar através do documentos seqüencialmente. A tabela na parte inferior da tela mostra todos armazenados campos para o documento atualmente selecionado.

Percorrer por termo é mais complicado, você pode ir sobre ele de várias maneiras. Clicando Primeiro

Prazo navega a seleção prazo para o primeiro termo no índice. Você pode rolar através de termos clicando no botão próximo mandato. O número de documentos containing um determinado termo é mostrado como o "Doc freq do termo" valor. Para selecionar um es- prazo específicos, tipo de todos, mas o último caractere na caixa de texto, clique em próximo mandato, e navegar para frente até encontrar o termo desejado.

Logo abaixo do browser termo é o documento browser prazo, o que permite que você navegar através dos documentos que contenham o termo que você selecionou. O primeiro documento

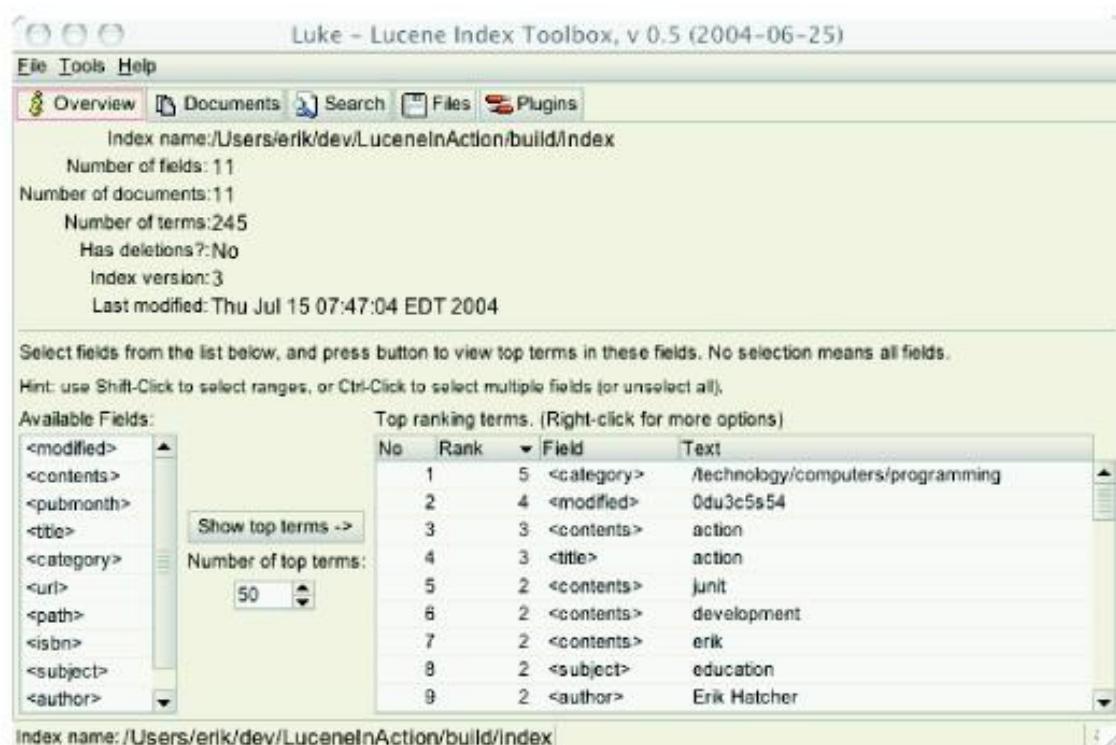


Figura 8.3 Luke: visão geral do índice, permitindo que você navegue campos e os termos

botão seleciona o primeiro documento que contém o termo selecionado, e, como quando você está navegando termos, Doc Em seguida navega para a frente.

O documento selecionado, ou todos os documentos que contenham o termo selecionado, também pode

ser excluído a partir desta tela (use cautela se este é um índice de produção, é claro!).

Outra característica do guia Documentos é o "Copiar texto para área de transferência" de recursos.

Todos os campos mostrado, ou o campo selecionado, podem ser copiados para o clipboard. Por exemplo, copiar todo o documento para a área de transferência coloca o seguinte texto lá:

```

<pubmonth:200310> Palavra-chave
<title:JUnit Texto em Action>
<category:/technology/computers/programming> Palavra-chave
<url:http://www.manning.com/massol> Não indexados
Palavra-chave <caminho: C: \ dev \ LuceneInAction \ Manuscrito \ data \
tecnologia \
computadores \ programação \ jia.properties>
<isbn:1930110995> Palavra-chave
Massol> <author:Vincent palavra-chave
Husted> palavra-chave <author:Ted

```

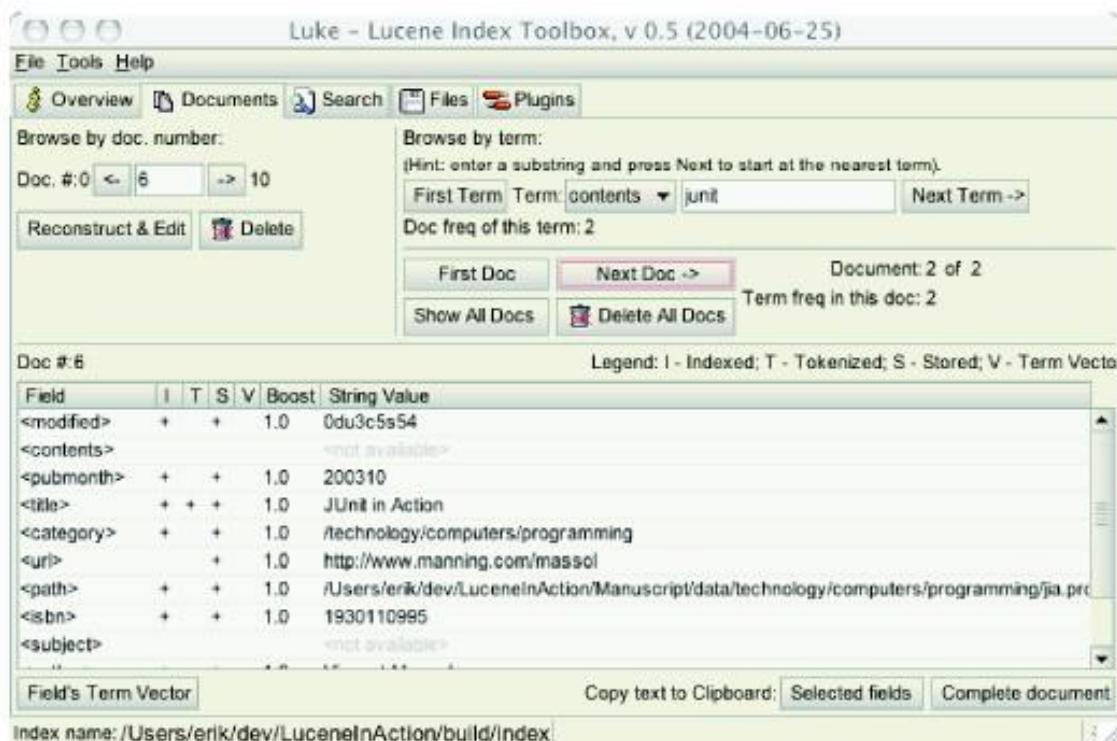


Figura 8.4 Guia de Lucas Documentos: sinta o poder!

#### NOTA

É importante notar que Lucas só pode trabalhar dentro das limitações de um índice Lucene, e campos UnStored não tem o texto disponível na sua forma original. Os termos desses campos, é claro, são navegáveis com Lucas, mas os campos não estão disponíveis no visualizador de documentos ou para copiar para a área de transferência (por exemplo, a nossa `conteúdo` campo, neste caso).

Clicar no botão Mostrar todos os Docs desloca a visão para a guia de pesquisa com uma busca sobre o termo selecionado, de tal forma que todos os documentos contendo este termo serão exibidos.

Se vetores prazo de um campo foram armazenados, o botão de Campo Vector Termo exibe uma janela mostrando termos e freqüências.

Uma característica final do guia Documentos é o "Reconstruir & Editar".

Ao clicar neste botão abre um editor de documentos que lhe permite editar (apagar e re-adicionar) o documento no índice ou adicionar um novo documento. Figura 8.5 mostra um doc que este documento está sendo editado.

Lucas reconstrói campos que foram tokenized mas não armazenados, agregando em posição fim todos os termos que foram indexadas. Reconstrução de um campo é um potencialmente perda de operação, e Lucas alerta sobre isso quando você ver um campo reconstruída

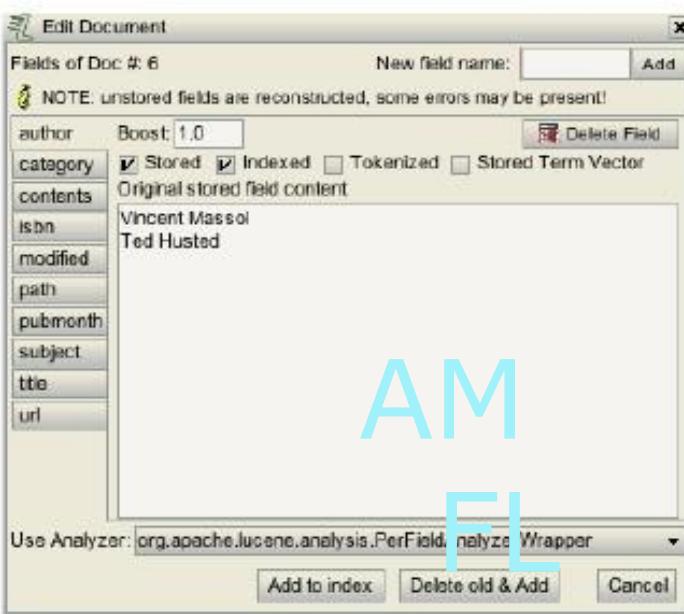


Figura 8.5  
Editor de documentos

(Por exemplo, se palavras de parada foram removidos ou tokens foram stemmed durante o processo de análise, em seguida, o valor original não está disponível).

Ainda está procurando por aqui chefe,

Nós já mostramos duas maneiras de chegar a automaticamente na guia Pesquisa: escolher "Mostrar todos os docs termo" a partir do menu do botão direito do "Top termos ranking" secção da guia Visão Geral, e clicando em Mostrar Todos os documentos a partir do navegador no prazo

na guia Documentos.

Você também pode usar a guia Pesquisar manualmente, entrando `QueryParser` expressão sintaxe, juntamente com sua escolha de Analisador e de campo padrão. Clique em Procurar quando

a expressão e os outros campos são como desejado. A tabela a baixo mostra todas as documentos dos hits de busca, como mostrado na figura 8.6.

Clicando duas vezes um documento muda de volta para a guia Documentos com o documento adequado pré-selecionados. É útil interativamente experimentar pesquisa expressões e ver como `QueryParser` reage a eles (mas não se esqueça de confirmar as suposições para testar casos, também!). Lucas mostra todos os analisadores que encontra no classpath, analisadores, mas apenas com construtores sem-arg pode ser usado com Luke. Lucas também fornece uma visão de pontuação com o recurso de explicação.

Para ver explicação pontuação, selecione um resultado e clique no botão Explicação; um exemplo é mostrado na figura 8.7.

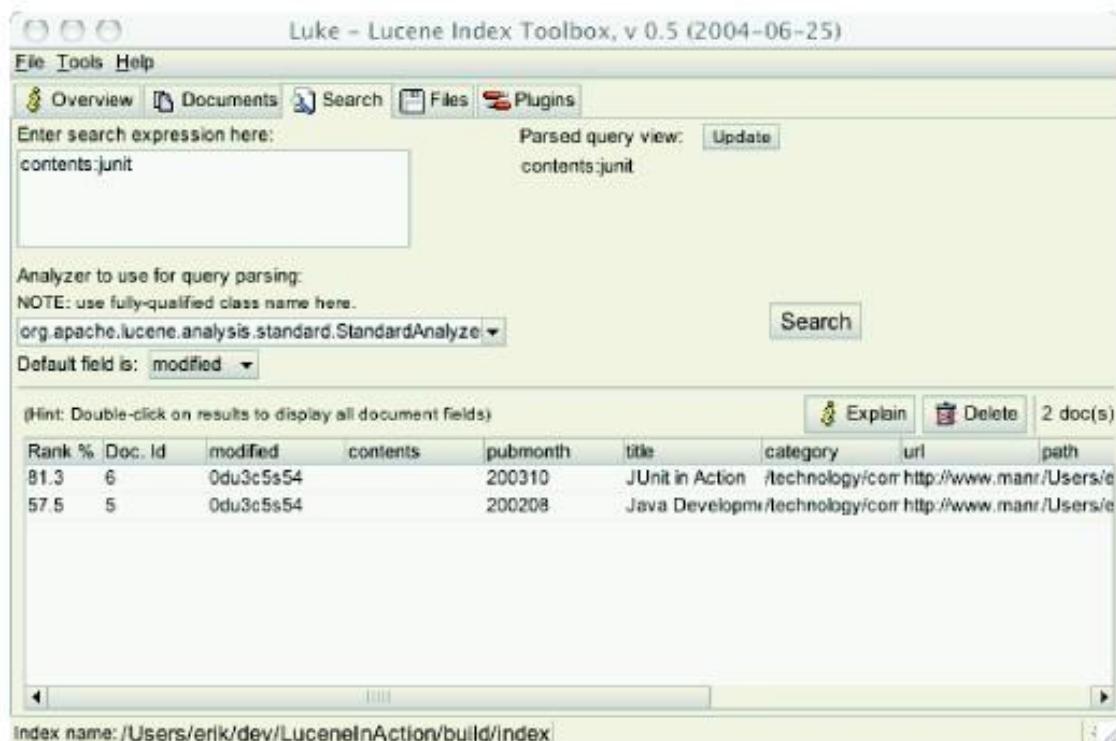


Figura 8.6 Searching: uma maneira fácil de experimentar com QueryParser

### Visualizar arquivos

A visão final em Lucas exibe os arquivos (e seus tamanhos) que compõem a internais de um diretório índice Lucene. O tamanho do índice total também é mostrada, como você pode ver na figura 8.8.

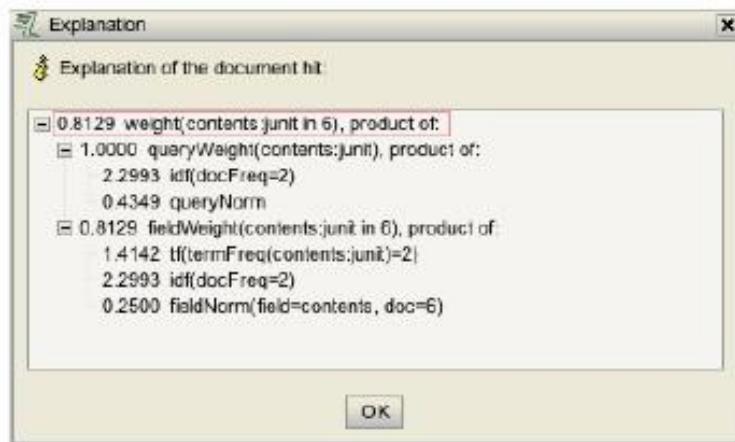
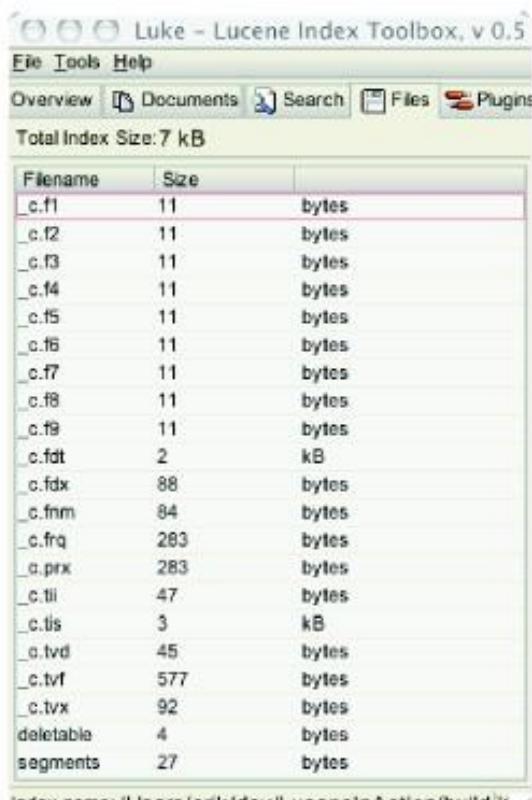


Figura 8.7  
Lucene explicação de pontuação



The screenshot shows the Luke - Lucene Index Toolbox, v 0.5 application window. At the top, there's a menu bar with File, Tools, Help. Below the menu is a toolbar with icons for Overview, Documents, Search, Files, and Plugins. A status bar at the bottom displays the index name: /Users/erik/dev/LuceneInAction/build/index.

Filename	Size	
_c.f1	11	bytes
_c.f2	11	bytes
_c.f3	11	bytes
_c.f4	11	bytes
_c.f5	11	bytes
_c.f6	11	bytes
_c.f7	11	bytes
_c.f8	11	bytes
_c.f9	11	bytes
_c.fdt	2	kB
_c.fdx	88	bytes
_c.fnm	84	bytes
_c.frq	283	bytes
_c.prx	283	bytes
_c.tii	47	bytes
_c.tis	3	kB
_c.tvd	45	bytes
_c.tvf	577	bytes
_c.tvx	92	bytes
deletable	4	bytes
segments	27	bytes

Figura 8.8  
 Visão de Lucas Arquivos mostra quão grande é um índice.

### Plugins vista

Como se as características já descritas sobre Lucas não bastasse, Andrzej passou o quilômetro extra e acrescentou um plug-in quadro para que outros possam adicionar ferramentas

Lucas. Um plug-in vem construído em: a Ferramenta de Analisador. Esta ferramenta tem o mesmo

finalidade que a AnalyzerDemo desenvolvida no ponto 4.2.3, mostrando os resultados da o processo de análise em um bloco de texto. Como um bônus adicionado, destacando um selecionados

token é um mero botão de clique de distância, como mostrado na figura 8.9.

Consulte a documentação do Lucas e código fonte para obter informações sobre como desenvolver seus próprios plug-in.

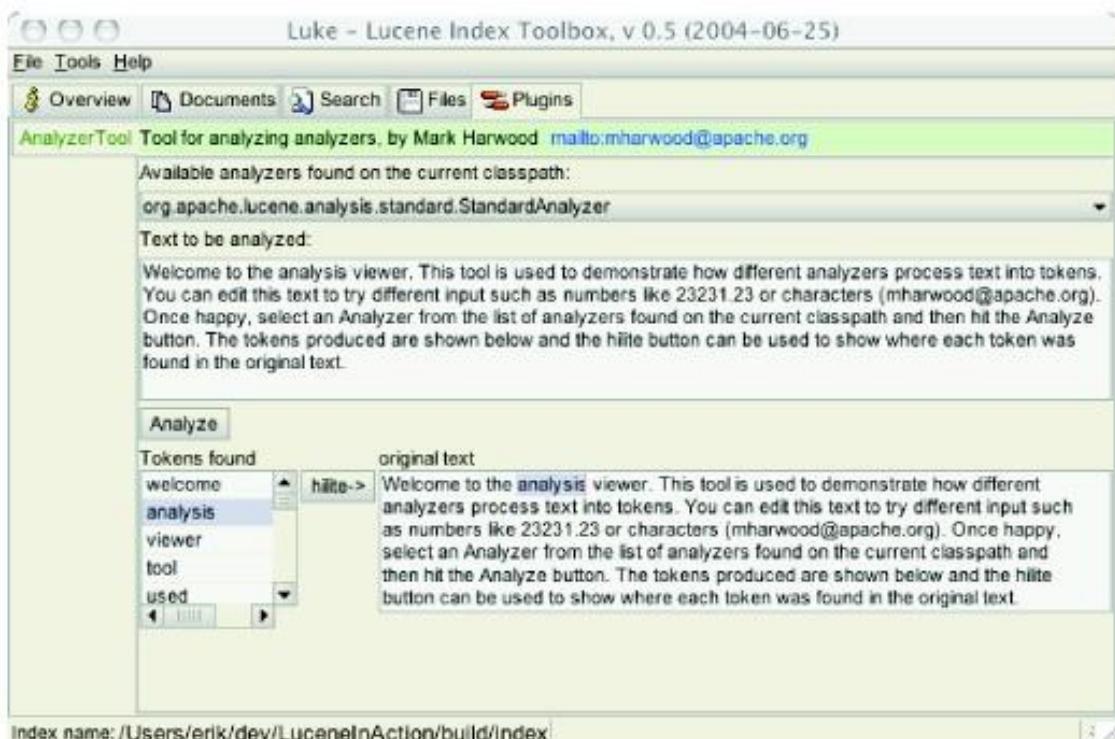


Figura 8.9 Analyzer Tool plug-in

### 8.2.3 LIMO: Lucene Índice de Monitor

Julien Nioche é o criador do Índice Lucene Monitor (LIMO).<sup>2</sup> Ele está disponível online em <http://limo.sourceforge.net/>. LIMO fornece uma interface de navegador da Web para Índices Lucene, dando-lhe um rápido olhar para a informação do índice de status, como se um índice está bloqueado, a última data de modificação, o número de documentos, e um resumo de campo. Além disso, um browser rudimentar documento permite que você navegue através de documentos em seqüência.

Figura 8.10 mostra a página inicial, onde você pode selecionar um ou mais pré-configurado figurado índices.

Para instalar LIMO, siga estes passos:

- 1 Baixar a distribuição LIMO, que é um arquivo WAR.
- 2 Expandir o arquivo WAR no webapps do Tomcat / webapps limo.

<sup>2</sup> LIMO v0.3 é a versão mais recente no momento da redação deste texto.

- 3 Editar o arquivo de limo / WEB -INF/web.xml, adicionando um par de referências a Índice de diretórios Lucene.

LIMO usa parâmetros de contexto no arquivo web.xml para controlar quais índices se tornam visíveis. Uma de nossas entradas aparece no web.xml assim:

```
<context-param>
    <param-name> LIA </ param-name>
    <param-value>
        / Users / erik / dev / LuceneInAction / build / index
    </ Param-value>
    <description> Lucene No índice de amostra Action </ description>
</ Context-param>
```

A versão do LIMO que usamos incorpora Lucene 1.3; se você precisa usar uma nova versão do Lucene que LIMO incorpora, substitua o JAR Lucene na WEB -INF/lib por remover o arquivo existente e adicionar um mais novo.

Depois de seguir as etapas de instalação e configuração, inicie o web contentor. Navegue até a URL apropriada (<http://localhost:8080/limo/> no nosso caso), e ter um assento no LIMO. Selecione um índice configurado para pesquisar.

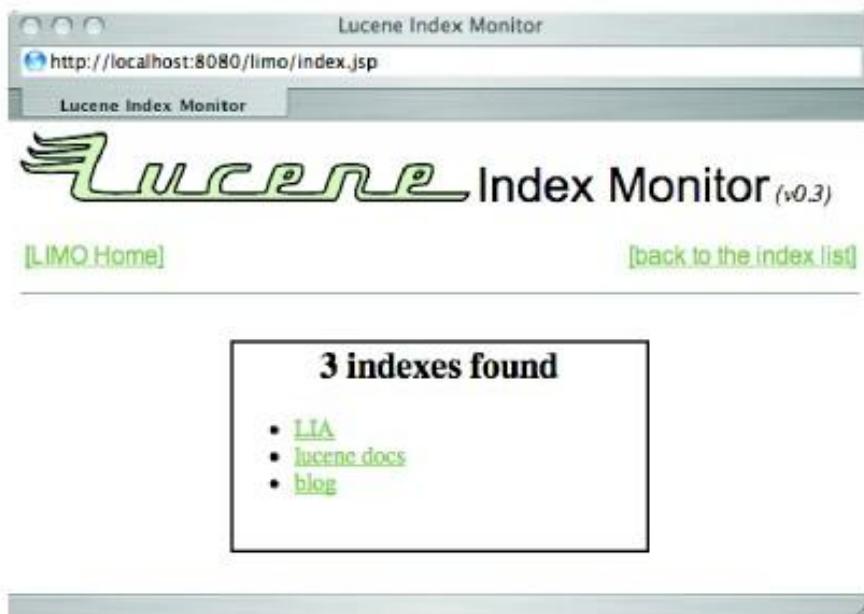


Figura 8.10 LIMO: selecionando um índice

### Navegando um índice

LIMO tela é apenas outro é o resumo de índice e visualização do browser de documento. Figure 8.11 mostra um exemplo.

Clique no links Anterior e Próximo para navegar através dos documentos. Todos os campos armazenados são mostrados à direita, indicando se eles são armazenados e / ou indexados.

### Usando LIMO

LIMO interface do usuário não é fantasia, mas ele faz o trabalho. Você pode querer ter LIMO instalado em uma instância Tomcat fixada em um servidor de produção. Ser capaz de obter uma visão rápida de quantos documentos estão em um índice, se é bloqueado, e quando foi a última atualização pode ser útil para fins de monitoramento. Além disso, usando o Páginas JSP LIMO como base para a construção de seu ponto de vista próprio acompanhamento personalizado poderia ser uma poupança de tempo. Porque as funções LIMO como uma aplicação web e não permite quaisquer operações destrutivas em um índice, ele fornece uma maneira prática de espreitar uma Índice remoto.

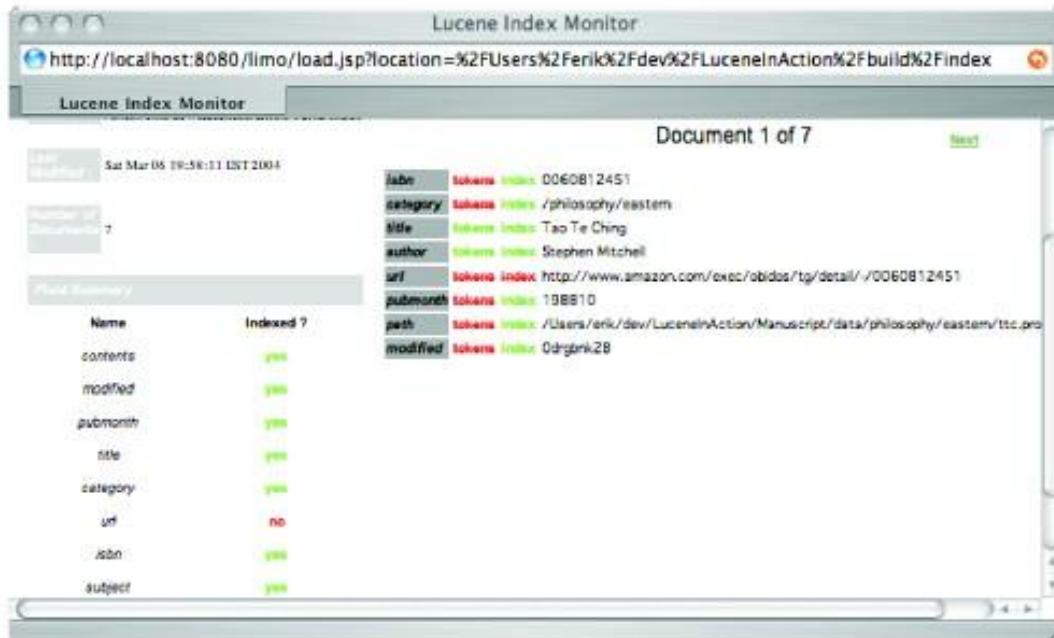


Figura 8.11 Cruzeiro no LIMO

### 8.3 Analisadores, tokenizers e TokenFilters, oh meu

---

Os analisadores mais, melhor, sempre dizemos. E o Sandbox não desaponta nesta área: Abriga vários analisadores específicos de idiomas, uma relacionada com alguns filters e tokenizers, e os analisadores algoritmo slick Snowball. Os analisadores estão listados na tabela 8.2.

Tabela 8.2 Sandbox analisadores

Analyzera	TokenStream fluxo
org.apache.lucene.analysis.br.BrazilianAnalyzer	StandardTokenizer → StandardFilter → StopFilter (Tabela stop personalizada) → BrazilianStemFilter → LowerCaseFilter
org.apache.lucene.analysis.cjk.CJKAnalyzer	CJKTokenizer → StopFilter (Personalizado para Inglês palavras ironicamente)
org.apache.lucene.analysis.cn.ChineseAnalyzer	ChineseTokenizer → ChineseFilter
org.apache.lucene.analysis.cz.CzechAnalyzer	StandardTokenizer → StandardFilter → LowerCaseFilter → StopFilter (Lista stop personalizada)
org.apache.lucene.analyzer.nl.DutchAnalyzer	StandardTokenizer → StandardFilter → StopFilter (Tabela stop personalizada) → DutchStemFilter
org.apache.lucene.analyzer.fr.FrenchAnalyzer	StandardTokenizer → StandardFilter → StopFilter (Tabela stop personalizada) → FrenchStemFilter → LowerCaseFilter
org.apache.lucene.analysis.snowball.SnowballAnalyzer	StandardTokenizer → StandardFilter → LowerCaseFilter [→ StopFilter ] → SnowballFilter

<sup>um</sup> Note o nome do pacote diferente para o SnowballAnalyzer-it está alojado em um diretório sandbox diferentes que os outros.

Os analisadores específicos do idioma variam na forma como eles tokenize. O brasileiro e Analisadores francesa uso específico do idioma e costumes decorrentes parar de listas de palavras.

O analisador usa Checa tokenization padrão, mas também incorpora um costume parar lista de palavras. Os chineses e CJK (chinês-japonês-coreano) analisadores tokenize caracteres de dois bytes como um token único a manter um caráter lógico intacta. Nós demonstrar a análise de caracteres chineses na seção 4.8.3, ilustrando como estes dois analisadores de trabalho.

Cada um destes analisadores, incluindo a `SnowballAnalyzer` discutido na próxima seção, permite que você personalize a lista de stop word-assim como o `stopAnalyzer` faz (ver seção 4.3.1). A maioria destes analisadores de fazer um pouco no processo de filtragem. Se os decorrentes ou tokenization é tudo que você precisa, peça emprestado as peças relevantes, e construir seu próprio costume analisador das partes aqui. Seção 4.6 cobre creating analisadores personalizados.

### 8.3.1 SnowballAnalyzer

O `SnowballAnalyzer` merece uma menção especial porque serve como um condutor de um família inteira de derivações para as diferentes línguas. Decorrentes foi introduzido pela primeira vez

na secção 4.7. Dr. Martin Porter, que também desenvolveu a Porter decorrentes algoritmo, criou o Snowball algorithm.<sup>3</sup> O algoritmo de Porter foi concebido para Inglês apenas e, além disso, muitas "supostas" implementações não aderem a a definição faithfully.<sup>4</sup> Para tratar dessas questões, o Dr. Porter rigorosamente definidos o sistema de algoritmos Snowball decorrentes. Através destes algoritmos definições, as implementações precisa pode ser gerada. Na verdade, o projeto de bolas de neve em Sandbox Lucene tem um processo de compilação que pode puxar as definições do Dr. Porter site e gerar a aplicação Java.

Um dos casos de teste demonstra o resultado da stemmer Inglês stripping fora da fuga ming a partir de decorrentes eo sa partir de algoritmos:

```
testEnglish public void () throws Exception {
    Analisador analisador = new SnowballAnalyzer ("Inglês");

    assertAnalyzesTo (analisador,
        "Decorrentes algoritmos", new String [] {"tronco", "algoritmo"});
}
```

`SnowballAnalyzer` tem dois construtores, ambos aceitam o nome stemmer apenas, e uma especifica um `String []` stop palavra-lista para usar. Muitas derivações única existir para várias línguas. O derivações não-Inglês incluem dinamarquês, holandês, finlandês, Francês, alemão, German2, italiano, Kp (Kraaij-Pohlmann algoritmo para holandês), Norueguês, Português, russo, espanhol e sueco. Há alguns Inglês-derivações específicas chamado Inglês, Lovins e Porter. Estes são os nomes exatos valores válidos para o argumento `SnowballAnalyzer` construtores. Aqui está um exemplo usando o algoritmo Espanhol decorrentes:

<sup>3</sup> O nome Bola de neve é uma homenagem ao SNOBOL linguagem de manipulação de string.

<sup>4</sup> De <http://snowball.tartarus.org/texts/introduction.html>

```
testSpanish public void () throws Exception {
    Analisador analisador = new SnowballAnalyzer ("espanhol");

    assertAnalyzesTo (analisador,
        "Algoritmos", new String [] {"algoritmo"});
}
```

Se seu projeto exige decorrentes, recomendamos que você dá o Snowball analisador de sua atenção pela primeira vez desde um perito no campo decorrentes desenvolveu. E, como já mencionado, mas vale a pena repetir, você pode querer usar o inteligente pedaço deste analisador (o `SnowballFilter`) Envolto em seu próprio costume ana-implementação lyzer. Várias seções no capítulo 4 discutem escrever personalizado analyzers em grande detalhe.

### 8.3.2 Obtendo os analisadores Sandbox

Dependendo de suas necessidades, você pode querer distribuições JAR binária desses analyzers ou código-fonte-prima da qual a tomar emprestadas idéias. Seção 8.10 fornece detalhes sobre como acessar o repositório CVS Sandbox e como construir binários distribuições. Dentro do repositório, o analisador de Snowball reside em contribuições / bola de neve, a outros analisadores discutidos aqui são de contribuições / analisadores. Não há dependências externas para estes analisadores que não Lucene em si, então eles são fáceis de incorporar. Um programa de teste chamado `TestApp` está incluído para o Projeto bola de neve. É executado da seguinte maneira:

```
> Java-cp dist / snowball.jar net.sf.snowball.TestApp
Uso: TestApp <stemmer name> <input file> [<saida-o file>]

> Java-cp dist / snowball.jar
⇒ net.sf.snowball.TestApp Lovins spoonful.txt
... saída de stemmer aplicado arquivo especificado
```

A Bola de Neve `TestApp` ignora `SnowballAnalyzer`. Somente o lematizador Snowball em si é usado com divisão rudimentar de texto em branco.

## 8.4 Java Development com Ant e Lucene

---

Um ponto de integração natural com o Lucene incorpora indexação de documentos em um processo de construção. Como parte de Java Development com Ant (Hatcher e Loughran, Manning Publications, 2002), Erik criado uma tarefa Ant para o índice de um diretório baseado em arquivo documentos. Este código já foi melhorada e é mantido no Sandbox.

Por documentos índice durante um processo de criação? Imagine um projeto que é fornecendo um sistema de ajuda integrado com capacidade de busca. Os documentos são provavelmente estático para uma versão específica do sistema, e ter um read-only

índice criado em tempo de compilação se encaixa perfeitamente. Por exemplo, que se o Ant, Lucene, e outros projetos tiveram uma pesquisa de domínio específico em seus respectivos sites? Ele faz sentido para a documentação de busca para ser a versão mais recente lançamento, que não precisa ser atualizada dinamicamente.

#### 8.4.1 Usando a tarefa <index>

Listagem 8.2 mostra um simplista Ant 1.6.x compatível com arquivo de construção que os índices de um diretório de arquivos de texto e HTML.

##### Listagem 8.2 Usando o Ant <index> tarefa

```
<? Xml version = "1.0"?>
<project name="ant-example" default="index">

    <description>
        Lucene exemplo índice de Ant
    </ Description>

    <property name="index.base.dir" location="build"/>
    <property name="files.dir" location="."/>

    <target name="index"> <mkdir dir="${index.base.dir}"/>

        <índice = "${index.base.dir} / index"
            xmlns = "antlib: org.apache.lucene.ant">
            <fileset dir="${files.dir}"/>
        </ Index>
    </ Target>

</ Project>
```

A integração é Ant Ant 1.6 Antlib compatível, como pode ser visto com o xmlns especificação. O legado `<taskdef>` método pode ainda ser usado, também. Listagem 8.2 mostra o uso mais básico do `<index>` especificação da tarefa, exigindo minimamente do diretório de índice e um conjunto de arquivos de arquivos a serem consideradas para indexação. O padrão de arquivo manipulação de índices único mecanismo arquivos que terminam com. txt ou. html.<sup>5</sup> Tabela 8.3 lista os campos criados pela tarefa índice eo manipulador de documento padrão. Apenas caminho e modificado são campos fixos, os outros vêm do manipulador de documento.

<sup>5</sup> JTidy é atualmente usado para extrair o conteúdo HTML para indexação. Consulte a secção 7.4 para mais informações sobre índice ing HTML.

Tabela 8.3 <index> padrão campos de tarefas

Nome do campo	Tipo de campo	Comentários
caminho	Palavra chave	Caminho absoluto para um arquivo
modificado	Palavra chave (Como Data da última modificação de um arquivo)	
título	Texto	<title> em arquivos HTML, e nome de arquivo para arquivos txt..
Conteúdo	Texto	Conteúdo completo de arquivos txt;. Analisado <body> de arquivos HTML
rawcontents	UnIndexed	Conteúdo bruto do arquivo

É muito provável que o manipulador de documento padrão é insuficiente para suas necessidades. Felizmente, um documento personalizado manipulador de ponto de extensão existe.

#### 8.4.2 Criando um manipulador de documento personalizado

Uma instalação de documento manipulador-swappable está embutido no <index> tarefa, permitindo

implementações personalizadas para lidar com diferentes tipos de documentos e controlar o Campos Lucene criado.<sup>6</sup> Não só o manipulador de documento seja especificado, configuração parâmetros podem ser passados para o manipulador de documento personalizado.

Utilizou-se o

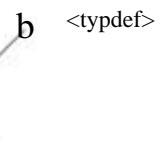
Formiga <index> tarefa, como mostrado na listagem 8.3, para criar o índice utilizado na maioria do código para este livro.

Use listagem 8.3 do <index> tarefa de construir o índice de amostra para este livro

```

<target name="build-index" depends="compile">
    resource="org/apache/lucene/ant/antlib.xml"> <typedef
        <classpath>
            <caminho refid="compile.classpath"/>
            location="${build.dir}/classes"/> <pathelement
        </ Classpath>
    </ Typedef>

    <índice = "${build.dir} / index"
        DocumentHandler = "lia.common.TestDataDocumentHandler">
        <fileset dir ="${data.dir}"/>
        <config basedir ="${data.dir}" />basedir propriedade de configuração
    </ Index>
</ Target>
```



Use manipulador de documento personalizado

d

c

<sup>6</sup> O <index> tarefa instalação manipulador documento foi elaborado muito antes da Otis quadro construído em capítulo 7. Neste ponto, os dois documentos de manipulação de estruturas são independentes uns dos outros, al- que eles são semelhantes e podem ser facilmente fundidos.

- b Usamos `<typedef>` porque precisamos de uma dependência adicional acrescentado à classe caminho para o nosso manipulador de documento. Se nós não precisamos de um manipulador de documento personalizado, o `<typedef>` seria desnecessário.
- c Usamos um manipulador de documento personalizado para processar arquivos de forma diferente.
- d Aqui nós a mão do nosso manipulador de documento uma propriedade de configuração, `basedir`. Este diretório que os caminhos relativos da pasta se referem dentro da hierarquia de pastas e . Propriedades arquivos. Cada um. Arquivo de propriedades contém informações sobre um livro único, como neste exemplo:

```
title = Tao Te Ching \ u9053 \ u5FB7 \ u7D93
isbn = 0060812451
author = Stephen Mitchell
subject = taoism
pubmonth = 198810
url = http://www.amazon.com/exec/obidos/tg/detail/-/0060812451
```

A hierarquia da pasta serve como meta-dados também, especificando as categorias de livros. Figura 8.12 mostra o diretório de dados da amostra. Por exemplo, as propriedades. Exemplos só é mostrado o arquivo `ttc.properties` que reside no `data/philosophy/eastern` diretório. Os pontos de base de dados e diretório para se despiu na manipulador do documento, como mostrado na listagem 8.4.

Para escrever um manipulador de documento personalizado, escolha uma das duas interfaces para implementar. Se você não precisa de nenhum adicional de meta-dados do arquivo de construção Ant, implemente `DocumentHandler`, Que tem o seguinte método retornando um único Lucene Documento exemplo:

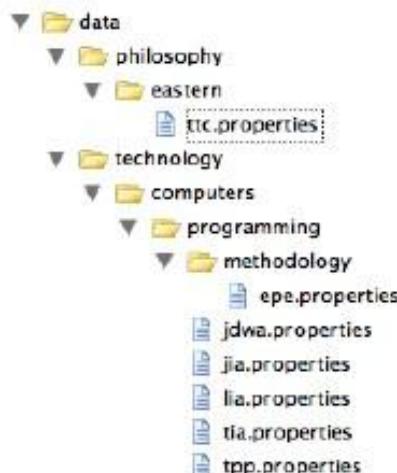


Figura 8.12  
Exemplo de estrutura de diretório de dados, com o caminho do arquivo especificando uma categoria

```
public interface DocumentHandler {
    GetDocument documento (arquivo)
        joga DocumentHandlerException;
}
```

**Implementação** ConfigurableDocumentHandler permite que o <index> tarefa de transmitir informações adicionais como java.util.Properties objeto:

```
ConfigurableDocumentHandler interface pública
    estende DocumentHandler {
    void configure (Properties props);
}
```

Opcões de configuração são passados através de um único <config> subelemento com arbitrariamente chamado atributos. O <config> nomes de atributos tornam-se as chaves do propriedades. Nossa completa TestDataDocumentHandler classe é mostrada na listagem 8.4.

Listagem 8.4 TestDataDocumentHandler: como construímos o nosso índice de amostra

```
TestDataDocumentHandler public class
    implementa {ConfigurableDocumentHandler
basedir String privado;

getDocument documento público (arquivo)
    throws DocumentHandlerException
Properties props = new Properties ();
try {
    props.load (new FileInputStream (arquivo));
} Catch (IOException e) {
    throw new DocumentHandlerException (e);
}

Documento doc = new Document ();

/ / Categoria vem de caminho relativo abaixo do diretório base
String categoria file.getParent () substring (basedir.length ()) .;
category = category.replace (File.separatorChar ,'/');

Obter categoria b
Corda isbn = props.getProperty ("isbn");
Corda title = props.getProperty ("title");
Corda author = props.getProperty ("autor");
Corda url = props.getProperty ("url");
Corda subject = props.getProperty ("subject");
Corda pubmonth = props.getProperty ("pubmonth");

Puxa campos
d
doc.add (Field.Keyword ("isbn", isbn)); Adicionar campos a
doc.add (Field.Keyword ("categoria", categoria)); Instância de documento
doc.add (Field.Text ("title", title));
```

```

    / / Split múltiplos autores em instâncias campo único
    String [] = autores author.split ",";
    for (int i = 0; i < authors.length; i + +) {
        doc.add (Field.Keyword ("autor", os autores [i]));
    }

    Adicionar campos a}

    doc.add (Field.UnIndexed ("url", url));
    doc.add (Field.UnStored ("sujeito", o assunto, true));

    doc.add (Field.Keyword ("pubmonth", pubmonth));

    doc.add (Field.UnStored ("conteúdo",
        agregada (new String [] {título, assunto, autor })));
}

retorno doc;
}

Adicionar o conteúdo do
campo
}
}

agregada private String (String [] strings) {
    StringBuffer buffer = new StringBuffer ();

    for (int i = 0; i < strings.length; i + +) {
        buffer.append (strings [i]);
        buffer.append ("");
    }

    buffer.toString return ();
}

configurar public void (Properties props) {
    this.basedir = props.getProperty ("basedir");
}

}

```

- b Baseamos a categoria sobre o caminho relativo do diretório base de dados, assegurando que as barras são usados como separadores.
- c Aqui puxamos cada campo a partir dos valores no arquivo de propriedades..
- d Nós adicionamos cada campo para a Documento exemplo; note os diferentes tipos de campos utilizados.
- e O assunto campo é marcada para o armazenamento do vetor prazo.
- f O conteúdo campo é um campo agregado: Podemos buscar um único campo contendo o autor eo assunto.

Quando você usa um manipulador de documento personalizado, além dos campos do manipulador cria, a <index> tarefa adiciona automaticamente caminho e modificado campos. Estes dois campos são utilizados para a indexação incremental, permitindo que apenas os arquivos recém-modificado para ser processado.

O arquivo de construção também pode controlar o analisador e mesclar fator. O `fac-merge` padrões para 20, mas você pode configurá-lo para outro valor, especificando `mergeFactor = "..."` como um atributo para o `<index>` tarefa. O analisador é especificada em uma das duas maneiras. Os analisadores built-in estão disponíveis usando `analisador = "..."`, onde o valor é simples, padrão, parar, espaços em branco, alemão ou russo. Se você precisar usar qualquer analisador de outros, especificar `analyzerClass = "..."` em vez disso, com a classe totalmente qualificado

nome. Atualmente, analisadores apenas que ter um construtor sem argumentos podem ser utilizados

com `<index>`; Isto exclui usando o `SnowballAnalyzer` diretamente, por exemplo.

Existem várias possibilidades interessantes, graças à flexibilidade do `<index>` tarefa, como indexação de documentação em vários idiomas. Você pode tem documentos separados por estrutura de diretórios (docs / en, docs / fr, docs / nl, e assim por diante), pelo nome do arquivo (index.html.en, index.html.fr, e assim por diante), ou por algum outro esquema. Você poderia usar o `<index>` vezes de tarefas múltiplas em um processo de construção para construir um índice separado para cada idioma, ou você poderia escrevê-los todos para o mesmo índice e usar um analisador diferentes para cada idioma.

#### 8.4.3 Instalação

O `<index>` tarefa requer três bibliotecas e pelo menos Ant 1.5.4 (embora Ant 1.6 ou superior é recomendado para tirar proveito do recurso Antlib). O JAR Lucene, JAR JTidy, eo JAR do `<index>` tarefa em si são obrigatórios. Obter esses JARs, colocá-los em um único diretório juntos, e usar o `-Lib` Ant 1.6 de linha de comando para apontar para esse diretório (ou use `<taskdef>` com o classpath adequada). Ver seção 8.10 para elaboração sobre como obter JARs do componente Sandbox, e consulte a documentação do Ant e Manning Java Development com Ant para detalhes sobre como trabalhar com Ant.

## 8.5 browser utilitários JavaScript

---

Integração Lucene em um aplicativo freqüentemente requer a colocação de um interface de pesquisa em uma aplicação web. `QueryParser` é útil, e é fácil de expor um texto simples caixa, permitindo ao usuário inserir uma consulta, mas pode ser mais amigável para os usuários a ver opções de consulta separados em campos, como uma seleção intervalo de datas em conjunto com uma caixa de texto para texto livre busca. Os utilitários JavaScript no Sandbox auxiliar do lado do navegador usabilidade na construção e validação sofisticados expressões adequadas para `QueryParser`.

### 8.5.1 construção consulta JavaScript e validação

Como já explorado em vários capítulos anteriores, expondo `QueryParser` diretamente para os usuários finais podem levar a confusão. Se você estiver fornecendo uma interface web para procurar um

índice Lucene, você pode querer considerar o uso do bem feito consulta JavaScript construtor e validador no Sandbox, originalmente escrito por Lucene companheiros desenvolvedor Kelvin Tan. O javascript Sandbox projeto inclui uma amostra de HTML arquivo que imita avançadas do Google opções de busca, como mostrado na figura 8.13.

O construtor de consulta suporta todos os campos HTML, incluindo texto e escondido campos, botões de rádio e única e múltipla seleciona. Cada campo HTML deve ter um campo correspondente HTML nomeado com o sufixo Modificador, controlar a forma como os termos são adicionados à consulta. O campo modificador pode ser um campo oculto para evitar um usuário de controlá-lo, como no caso dos campos de texto na figura 8.12. O construtor de consulta é colocado em um campo HTML (normalmente uma escondida), que é entregue a `QueryParser` no lado do servidor.

O validador consulta usa expressões regulares para fazer o seu melhor aproximação das o que é aceitável para `QueryParser`. Ambos os arquivos JavaScript permitem a personalização com características como modo de depuração para alertá-lo que está acontecendo, sufixos campo modificador, especificando se deseja enviar o formulário em construção, e muito mais. O Java-Arquivos de script estão bem documentadas e fácil de cair no seu próprio ambiente.

### Lucene Javascript Query Constructor

`luceneQueryConstructor.js` is a Javascript framework for constructing queries using the "advanced" search features of lucene, namely field-based searching, group searching (via parentheses) and various combinations of the aforementioned.

It also provides a convenient way to mimic a Google-like search, where all terms are ANDed, as opposed to Lucene's default OR modifier.

This HTML form provides examples on the usage of `luceneQueryConstructor.js`. An interface similar to [Google's Advanced Search](#) form is shown below:

<b>Find results</b>	<b>With all of the words</b>	<input type="text" value="jakarta lucene"/> <b>50 results</b>
	<b>With the exact phrase</b>	<input type="text"/>
	<b>With at least one of the words</b>	<input type="text"/>
	<b>Without the words</b>	<input type="text"/>
<b>File Format</b>	<b>Only</b> <input checked="" type="radio"/> return results of the file format	<input type="text" value="Adobe Acrobat PDF (.pdf)"/>
<b>Date</b>	Return results updated in the	<input type="text" value="anytime"/>
<b>Current Query:</b>	<code>+jakarta +lucene +fileName:(+pdf)</code> <input type="button" value="Update Query"/> <input type="button" value="Validate"/>	

Figura 8.13 Exemplo JavaScript

No momento da redação deste texto, o Sandbox javascript estava sendo reforçada. Ao invés de HTML show potencialmente out-of-date, que encaminhá-lo para os exemplos o Sandbox quando você precisa dessa capacidade.

### 8.5.2 Escapar caracteres especiais

`QueryParser` usa muitos caracteres especiais para as operadoras e agrupamento. A-`char`  
`acters` devem ser ignorados se eles são usados em um nome de campo ou como parte de um  
termo (ver  
seção 3.5 para obter mais detalhes sobre `QueryParser` escapar caracteres). Usando o `lucene-QueryEscaper.js` apoio da Sandbox, você pode escapar de uma seqüência de consulta.

Você deve usar o Escaper consulta apenas em campos ou cordas que não devem  
conter quaisquer caracteres especiais já Lucene. Por exemplo, seria incorreto  
para escapar de uma consulta construída com o construtor de consulta, uma vez que qualquer  
parênteses e  
operadores, acrescentou seria posteriormente fugiu.

### 8.5.3 Usando suporta JavaScript

Adicionando suporte JavaScript para o seu arquivo HTML requer apenas agarrando (ver secção  
8.10)

os arquivos JavaScript e referindo-se a eles no `<head>` seção da seguinte maneira:

```
<Script type = "text / javascript"  
       src = "luceneQueryConstructor.js"> </ script>  
<Script type = "text / javascript"  
       src = "luceneQueryValidator.js"> </ script>  
<script type="text/javascript" src="luceneQueryEscaper.js"> </ script>
```

Chamada `doMakeQuery` para construir uma consulta e `doCheckLuceneQuery` para validar um  
consulta. Ambos os métodos requerem um argumento de campo de formulário que especifica o  
campo para  
preencher ou validar. Para escapar de uma consulta, ligue `doEscapeQuery` com o campo de  
formulário ou  
uma seqüência de texto (que detecta o tipo); a string de consulta escapou serão devolvidos.

## Sinônimos de 8,6 WordNet

O que um web emaranhado de palavras que tecem. Um sistema desenvolvido na Universidade  
de Princeton Uni-

Cognitive Science Laboratory sity, conduzido pelo professor de psicologia George Miller,  
ilustra a rede de synonyms.<sup>7</sup> WordNet representa formas de palavras que são inter-  
mutável, tanto lexicalmente e semanticamente. Funcionalidade do Google define (tipo define:  
palavra como uma pesquisa no Google e veja por si mesmo), muitas vezes refere-se aos  
usuários on-line

<sup>7</sup> Curiosamente, este é o mesmo que relatou George Miller sobre o fenômeno de sete mais ou menos  
dois blocos na memória imediata.

The screenshot shows a web browser window with the URL <http://www.cogsci.princeton.edu/cgi-bin/webwn2.0?stage=1&word=search>. The title bar says "Web WordNet 2.0". The main content area has a large heading "WordNet 2.0 Search". Below it is a search form with a text input field containing "Search word:" and a button labeled "Find senses". A section titled "Overview for 'search'" follows. It states: "The noun 'search' has 5 senses in WordNet." Below this is a numbered list of definitions for the noun "search". At the bottom of the search form, there are checkboxes for "Show glosses" (checked) and "Show contextual help" (unchecked), and a "Search" button.

The verb "search" has 4 senses in WordNet.

1. **search**, hunt, hunting -- (the activity of looking thoroughly in order to find something)
2. **search** -- (an investigation seeking answers; "a thorough search of the ledgers reveal justified the search")
3. **search**, lookup -- (an operation that determines whether one or more of a set of items "they wrote a program to do a table lookup")
4. **search** -- (the examination of alternative hypotheses; "his search for a move that would be unsuccessful")
5. **search** -- (boarding and inspecting a ship on the high seas; "right of search")

Figura 8.14  
Pego no WordNet:  
palavra interconexões  
para pesquisa

WordNet sistema, permitindo que você navegue palavra interconexões. Figura 8.14 mostra os resultados da procura de pesquisa no site WordNet.

O que significa tudo isso para os desenvolvedores usando o Lucene? Com Dave Spencer contribuição para a Sandbox Lucene, o banco de dados pode ser sinônimo WordNet churned em um índice Lucene. Isto permite uma rápida pesquisa sinônimo para exemplo, para injeção sinônimo durante a indexação ou consultar (ver secção 8.6.2 para tal implementação).

### 8.6.1 Construir o índice sinônimo

Para construir o índice sinônimo, siga estes passos:

- 1 Download e expandir o arquivo do site prolog16.tar.gz WordNet em <http://www.cogsci.princeton.edu/~wn>.
- 2 Obter o binário (ou construir a partir da fonte; ver secção 8.10) do Sandbox Pacote WordNet.
- 3 Criar o índice sinônimo usando o `Syns2Index` programa a partir do comando de mando. O primeiro parâmetro aponta para o arquivo `wn_s.pl` obtidos no WordNet de distribuição a partir do passo 1. O segundo argumento especifica o caminho onde o índice Lucene será criado:

```
java org.apache.lucene.wordnet.Syns2Index
⇒ prologwn / wn_s.pl wordnetindex
```

O `Syns2Index` programa converte o banco de dados WordNet sinônimo Prolog em um índice Lucene padrão com um campo indexado `palavra` e os campos não indexados `syn` para cada documento. Versão 1.6 da WordNet produz 39.718 documentos, cada representando uma única palavra, o tamanho do índice é de aproximadamente 2,5 MB, tornando-com-

pacto suficiente para carregar como um `RAMDirectory` para acesso rápido.

Um programa utilitário segundo na área Sandbox WordNet permite olhar para cima `synonyms` de uma palavra. Aqui está uma pesquisa de amostra de uma palavra próximos e queridos aos nossos corações:

```
java busca wordnetindex org.apache.lucene.wordnet.SynLookup

Sinônimos encontrados para "pesquisa":
procurar
pesquisar
pesquisa
pesquisa
olhar
caça
caça
explorar
```

Figura 8.15 mostra esses sinônimos mesmo graficamente usando Luke.

Para usar o índice sinônimo em seus aplicativos, emprestar as peças relevantes a partir de `SynLookup`, Como mostrado na listagem 8.5.

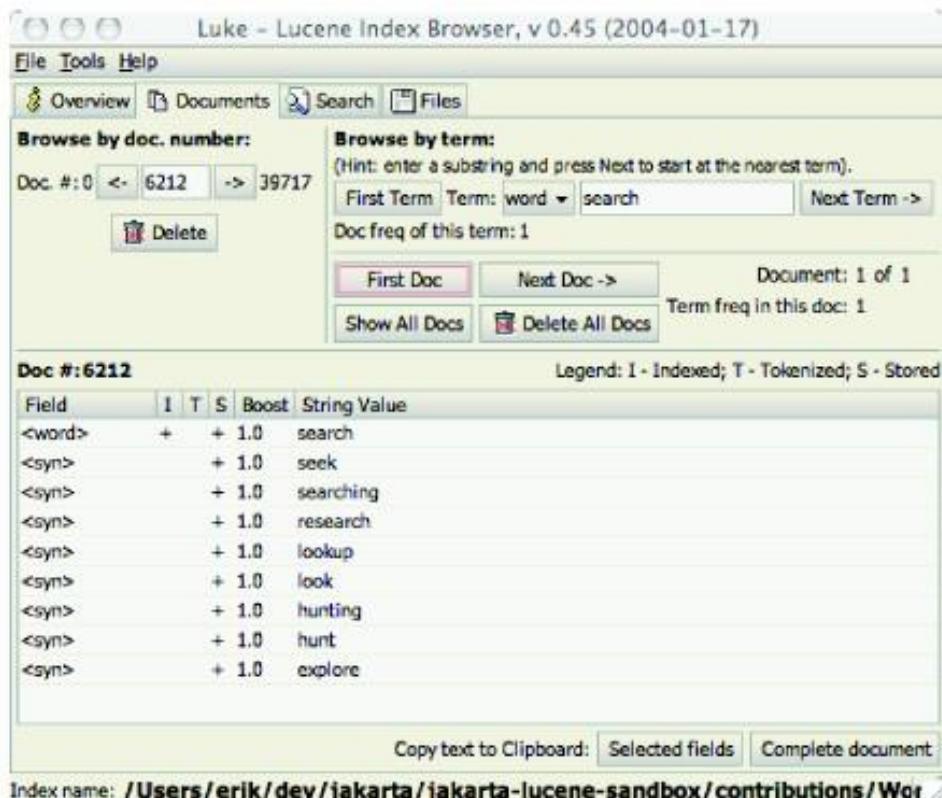


Figura 8.15 Cool app Lucas: sinônimos inspecionando WordNet

#### Listagem 8.5 Procurando sinônimos de um índice baseado em WordNet

```
SynLookup public class {

    public static void (String [] args) throws IOException {principal
        if (args.length! = 2) {
            System.out.println (
                "Java <word> <index org.apache.lucene.wordnet.SynLookup path>");
        }

        Diretório FSDirectory = FSDirectory.getDirectory (args [0], false);
        IndexSearcher searcher = new IndexSearcher (diretório);

        Palavra String = args [1];
        Hits hits = searcher.search (
            nova TermQuery (novo Termo ("palavra", palavra)));

        if (hits.length () == 0) {
            System.out.println ("Sem sinônimos encontrado para" palavra +);
        Else {}
            System.out.println ("Sinônimos encontrado para \" " + palavra + " \":");
        }
    }
}
```

```

for (int i = 0; i <hits.length (); i + +) {
    Documento doc = hits.doc (i);

    String [] valores = doc.getValues ("syn");

    for (int j = 0; j <values.length <; j + +) {
        System.out.println (valores [j]);
    }
}

searcher.close ();
directory.close ();
}
}

```

Enumerar  
sinônimos  
por palavra

AM

O SynLookup programa foi escrito para este livro, mas ele foi adicionado na WordNet codebase Sandbox.

FL

### 8.6.2 Subordinação sinônimos WordNet em um analisador

O costume `SynonymAnalyzer` da secção 4.6 pode facilmente conectar em WordNet sinônimos usando o `SynonymEngine` interface. Listagem 8,6 contém o `WordNet-SynonymEngine`, Que é adequado para uso com o `SynonymAnalyzer`.

#### Listagem 8,6 WordNetSynonymEngine

```

public class WordNetSynonymEngine implements {SynonymEngine
Diretório RAMDirectory;
IndexSearcher pesquisador;

pública WordNetSynonymEngine (índice Arquivo) throws IOException {
    diretório = new RAMDirectory (
        FSDirectory.getDirectory (índice, false));
    searcher = new IndexSearcher (diretório);Sinônimo de carga
}índice na RAM
                                         para acesso rápido
public String [] getSynonyms (String palavra) throws IOException {

    ArrayList synList = new ArrayList ();

    Hits hits = searcher.search (
        nova TermQuery (novo Termo ("palavra", palavra)));

    for (int i = 0; i <hits.length (); i + +) {
        Documento doc = hits.doc (i);

        String [] valores = doc.getValues ("syn");

```

```

        for (int j = 0; j < values.length; j++) {
            synList.add (valores [j]);
        }

        return (String []) synList.toArray (new String [0]);
    }
}

```

---

Ajustando o `SynonymAnalyzerViewer` da seção 4.6 para usar o `WordNetSynonymMotor`, O nosso exemplo de saída é a seguinte:

```

1: [rápida] [agilidade] [fast] [voar] [imediata] [agilidade] [prompt]
   [Prontamente] [rapidamente] [pronto] [rápida] [spry] [imediatamente]
   [Quente]
2: [brown] [marrom] [brownness]
3: [fox] 8 [atormentar] [befuddle] [confundir] [confundir]
   [Discombobulate] [dodger] [fob] [embriaguez] [espartalhão] [jogar]
   [Truque]
4: [salta]
5: [mais] [em] [o]
6: [preguiça] [preguiçoso] [otiose] [indolente] [fainéant]
7: [cães]

```

Curiosamente, sinônimos WordNet que existem para salto e cão (Ver a saída luci na listagem 8.1), mas apenas de forma singular. Talvez decorrentes devem ser adicionados à nosso `SynonymAnalyzer` antes da `SynonymFilter`, Ou talvez o `WordNetSynonymMotor` deve ser responsável pelo decorrentes palavras antes de procurá-los na WordNet índice. Essas são questões que precisam ser abordadas com base no seu ambiente. Isso enfatiza novamente a importância do processo de análise e os fato que merece sua atenção.

O Lucene WordNet código requer uma versão mais antiga (1.6) do WordNet banco de dados. Se você quiser ligar para as versões 2.x mais recentes do WordNet, você precisa se quer ajustar manualmente o código Sandbox Lucene ou empate em JWordNet, um Java API para WordNet alojados em <http://jwn.sourceforge.net/>.

### 8.6.3 Calling em Lucene

Com a penetração cada vez maior de dispositivos móveis e diminuindo seu tamanho, nós necessidade de entrada de texto inteligente de métodos. A interface T9 presentes na maioria dos telefones é muito

---

<sup>8</sup> Nós, aparentemente confuso ou outfoxed banco de dados sinônimo WordNet porque os sinônimos em projetada para raposa não se relacionam com o substantivo animal que se destinam.



Figura 8.16  
Celular-como interface Swing

mais eficiente do que exigindo caráter exato input.<sup>9</sup> como um protótipo de algo potencialmente útil, colocamos Lucene e WordNet Swing sob um celular-como interface, como mostrado na figura 8.16.10

Os botões 2-9 são mapeados para três ou quatro letras do alfabeto cada, idêntico a um celular. Cada clique em um desses números acrescenta o selecionado dígitos para um buffer interno; uma busca Lucene é feita para combinar palavras para esses dígitos. Os botões que não são mapeadas para letras são utilizados para outras capacidades: 1 rola a visão através da lista de palavras correspondentes (barra de status mostra quantas palavras coincidir com os dígitos inseridos), o asterisco (\*) backspaces um dígitos, desfazendo o último número digitado; 0 permite a depuração saída de diagnóstico para o console, e sustenido (#) apaga todos os dígitos, o que lhe permite iniciar uma nova entrada.

### Construção do índice de T9

Nós escrevemos uma classe de utilitário para pré-processar o índice WordNet original em um especial ized T9 índice. Cada palavra é convertido em um t9 campo palavra-chave. Cada palavra, a sua T9 equivalente, e o comprimento do texto da palavra são indexados, como mostrado aqui:

```
NewDoc documento = new Document();
newDoc.add (Field.Keyword ("palavra", palavra));
newDoc.add (Field.Keyword ("t9", t9 (palavra)));
newDoc.add (Campo Novo ("comprimento",
    Integer.toString (word.length ()), false, true, false));
```

<sup>9</sup> T9 é um método de entrada que mapeia cada botão numérico para várias letras do alfabeto. Uma série de números logicamente corresponde a um subconjunto de palavras sensatas. Por exemplo, 732724 feitiços pesquisa.  
10Many graças a Dave Engler para a construção da base framework de aplicações Swing.

O `t9` método não é mostrado, mas pode ser obtido a partir do código fonte do livro distribuição (consulte a secção "Sobre este livro" secção). O comprimento de palavra é indexada como

`Integer.toString()` valor para permitir a classificação por tamanho utilizando o recurso de classificação

discutido na seção 5.1.

Busca de palavras com T9

Para se divertir um pouco com o Lucene, que consulta para uma seqüência de dígitos, com uma `Boolean-`

Pergunta com um ligeiro olhar de frente para um usuário não tem que entrar todos os dígitos

Para

exemplo, se a 73.272 dígitos são inseridos, pesquisa é a primeira palavra mostrada, mas dois ou-  
ers também match (secpar11 e camponesa). A consulta usa uma impulsionou `TermQuery` no  
dígitos exatos (para garantir que as correspondências exatas vêm em primeiro lugar) e uma  
consulta wildcard match-

ing palavras com um ou dois personagens mais mais. Aqui está o `BooleanQuery` código:

```
Query = BooleanQuery BooleanQuery New ();
Termo termo Term = new ("t9", number);
TermQuery termQuery = new TermQuery (prazo);
termQuery.setBoost (2.0f);
WildcardQuery Plus2 = new WildcardQuery (
    Termo de novo ("t9" número, + "??" ));
query.add (termQuery, false, false);
query.add (Plus2, false, false);
```

Os resultados da pesquisa são classificados primeiro pelo escore, em seguida, por extensão, e  
finalmente alphabeti-

mente dentro das palavras do mesmo comprimento:

```
Hits hits = searcher.search (consulta,
    Ordenar novo (SortField new [] {SortField.FIELD_SCORE,
        nova SortField ("comprimento",
            SortField.INT),
        nova SortField ("palavra ")}));
```

Resultados da pesquisa são cronometrados e armazenados em cache. A barra de status exibe o  
tempo a busca

teve (muitas vezes sob 30ms). O cache permite ao usuário percorrer palavras.

Apenas um protótipo

Este protótipo de telefone celular é um desktop compellingly rápida e precisa T9 pesquisa  
implementação. No entanto, o índice Lucene é utilizado mais de 2MB de tamanho e é  
inadequadas dadas as atuais restrições de telefonia móvel de memória. Com um conjunto menor  
de palavras e algumas otimizações de indexação (usando um `UnStored` `t9` campo em vez de  
uma palavra-chave), o índice poderia ser reduzido drasticamente em tamanho. Com persistência,  
rápido,

e conectividade de servidor baratos a partir de dispositivos móveis, algumas pesquisas palavra  
poderia

---

<sup>11</sup> "Uma unidade astronômica de comprimento com base na distância da Terra em que paralaxe estelar é um segundo de arco; equivalente a 3,262 anos luz "(de acordo com um Google define: secpar resultado da WordNet).

talvez ser executadas no servidor ao invés de o cliente. Pesquisar no Google é já uma actividade comum dispositivo móvel!

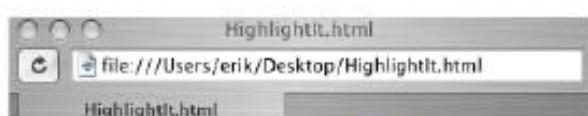
## 8.7 Destacando termos de consulta

Utilizadores dar o seu motor de busca contexto, alguns hits em torno de suas buscas é amigável e, mais importante, útil. Um bom exemplo é a pesquisa do Google resultados. Cada hit, como mostrado na figura 1.1, inclui até três linhas do jongoing documento destacando os termos da consulta. Muitas vezes, uma breve visão da em torno de contexto dos termos de pesquisa é o suficiente para saber se esse resultado vale a pena investigar mais.

Graças à contribuição de Mark Harwood, o Sandbox inclui infra-estrutura para realçar o texto baseado em uma consulta Lucene. Figura 8.17 é um exemplo do uso Marcador em uma amostra de texto com base em uma consulta prazo para ipsum.

O código Highlighter evoluiu recentemente substancialmente em um sofisticado e utilidade flexível. O Highlighter inclui três partes principais: fragmenter, Artilheiro, Formatter e. Estes correspondem às interfaces Java pelos mesmos nomes, e cada um tem uma implementação built-in para facilidade de uso. O exemplo mais simples de Highlighter retorna o melhor fragmento, ao redor de cada termo de correspondência com HTML <B> tags:

```
Texto String = "A ligeira raposa marrom ataca o cão preguiçoso";  
  
TermQuery query = new TermQuery (Termo de novo ("campo", "raposa"));  
Artilheiro artilheiro QueryScorer = new (query);  
Highlighter Highlighter = new (artilheiro);  
  
TokenStream tokenStream =  
    nova SimpleAnalyzer (). tokenStream ("campo",  
    novo StringReader (texto));  
  
System.out.println (highlighter.getBestFragment (tokenStream, texto));
```



Contrary to popular belief, **Lorem Ipsum**... from a **Loorem** **Ipsum** passage, and going through..., discovered the undoubtable source. **Loorem** **Ipsum** comes... the Renaissance. The first line of **Loorem** **Ipsum**, "**Loorem ipsum dolor sit amet..**", comes from a line

Figura 8.17  
Destacando termos de consulta

O código anterior produz esta saída:

```
O marrom rápida <B>Fox </B> salta sobre o cão preguiçoso
```

Highlighter requer que você fornecer não somente um marcador eo texto para destacar, mas também um TokenStream. Analisadores de produzir TokenStreams (ver capítulo 4). A sucesso com sucesso destacar termos, os termos do Pergunta precisam corresponder Símbolos emitidos

a partir do TokenStream. O mesmo texto deve ser usado para gerar o TokenStream como é usada para o texto original para destacar. Cada Símbolo emitida por um Token-Córrego contém informações de posição, indicando que no texto original para começam e terminam os destaque.

O Highlighter quebra o texto original em fragmentos, usando um fragmenter. A built-in SimpleFragmenter divide o texto original em fragmentos de mesmo tamanho com o tamanho padrão de 100 caracteres. O tamanho dos fragmentos é controlável, como você verá na listagem 8.6.

QueryScorer Artilheiro é o built-in. O trabalho do Marcador é principalmente para classificar fragmentos. QueryScorer usa os termos da consulta; extrai-los de primitivos termo, frase, e Boolean consultas e pesos-los com base na sua corresponding fator de impulso. A consulta deve ser reescrito em sua forma mais primitiva de Consulta-Marcador para ser feliz. Por exemplo, wildcard, fuzzy, consultas, prefixo e faixa reescrever-se a um BooleanQuery de todos os termos correspondentes. Chamada Consulta.reescrever (IndexReader) reescrever uma consulta antes de passar a Pergunta para Consulta-

Marcador (A menos que, como neste exemplo, você está certo que a consulta é uma primitiva).

Finalmente, o Formatter decora texto prazo. A built-in SimpleHTMLFormatter, salvo indicação em contrário, usa começam e terminam HTML tags em negrito para cercar o destaque de texto prazo. Highlighter usa tanto o SimpleHTMLFormatter e Simple-Fragmenter por padrão. Para cada termo é destaque, o Formatter é entregue um pontuação token. Esta pontuação é, quando se utiliza QueryScorer, É o fator de impulso da consulta

cláusula do termo. Esta pontuação token pode ser usado para afetar a decoração baseada sobre a importância do termo. A Formatter personalizado precisaria ser implementado para aproveitar esse recurso, mas isso está além do escopo desta seção.

### 8.7.1 Destacando com CSS

Utilização <B> tags ao texto surround que serão prestados pelos navegadores é um razoável padrão. Fancier estilo deve ser feito com folhas de estilo em cascata (CSS) em seu lugar.

Nosso próximo exemplo usa personalizado começam e terminam tags para embrulhar termos destaque

com um <span> usando a classe CSS personalizada realçar. Usando atributos de CSS, a cor e formatação de termos destaque é dissociada da valorização,

permitindo um controle muito mais para a web designers que têm a tarefa de embelezar-  
ing nossa página de resultados.

Listagem 8.7 demonstra o uso de fragmenter um costume personalizado, definindo o  
tamanho do fragmento a 50, e um Formatter personalizado aos destaque estilos com CSS. Em  
nossa

primeiro exemplo, apenas um fragmento melhor foi devolvido, mas brilha em Highlighter  
retornando múltiplos fragmentos. HighlightIt, Na listagem 8.7, usa o Highlighter  
método para concatenar os melhores fragmentos com reticências (...) separador; how-  
cada vez que você também pode ter um String [] retornado por não passar em um separador,  
de modo

que seu código pode lidar com cada fragmento individualmente.

**Listagem 8.7 termos Destacando usando folhas de estilo em cascata**

```
public class {HighlightIt
    texto String private static final =
        "Ao contrário da crença popular, Lorem Ipsum é" +
        "Não o texto simplesmente aleatória. Ela tem raízes em um pedaço de" +
        "Literatura clássica Latina a partir de 45 aC, tornando-se mais de" +
        "2000 anos de idade. Richard McClinton, um professor de Latim" +
        "No Hampden-Sydney College, na Virgínia, procurou uma" +
        "Das palavras em Latim mais obscuras consectetur, de" +
        "Lorem Ipsum uma passagem e passar pela cita" +
        "Da palavra na literatura clássica, descobri o" +
        "Fonte de indiscutíveis. Lorem Ipsum vem das seções" +
        "1.10.32 e 1.10.33 do \" de Finibus Bonorum et " +
        "Malorum \" (Os Extremos do Bem e do Mal), por Cicero, " +
        "Escrito em 45 aC. Este livro é um tratado sobre a" +
        "Teoria da ética, muito popular durante o" +
        "Renascimento. A primeira linha de Lorem Ipsum, \" Lorem " +
        "Ipsum dolor sit amet .. \", vem de uma linha em " +
        "Seção 1.10.32." / / A partir http://www.lipsum.com/
```

```
public static void (String [] args) throws IOException {principal
    String nome = args [0];
```

```
if (nome == null) {
    System.err.println ("Uso: HighlightIt <filename>");
    System.exit (-1);
}
```

```
TermQuery query = new TermQuery (Termo de novo ("f", "ipsum"));
Artilheiro QueryScorer QueryScorer = new (query);Customize
SimpleHTMLFormatter formatador =tags redor
    nova SimpleHTMLFormatter ("<span class=\"highlight\">",
    "</ Span>");                                b
Highlighter Highlighter Highlighter = new (artilheiro formatador);
Fragmenter fragmenter SimpleFragmenter = new (50);Reduzir padrão
    tamanho do fragmento highlighter.setTextFragmenter
(fragmenter);                                c
```

```

TokenStream tokenStream = new StandardAnalyzer ()
    . TokenStream ("f", new StringReader (texto));
    d Tokenize texto

String result =
    highlighter.getBestFragments (tokenStream, texto, 5, "...");

Escritor FileWriter = new FileWriter (filename);
writer.write ("<html>");
writer.write ("<style> \ n" +
    ". Destacar {\ \ n" +
    "Background: yellow; \ n" +
    "} \ N" +
    "</ Style>");
writer.write ("<body>");
writer.write (resultado);
writer.write ("</ body> </ html>");
writer.Close ();
}
}

```

d Tokenize texto

e Destaque 5 melhores fragmentos

f Escrever destaque HTML

- b Nós personalizar as marcas envolventes para cada termo em destaque.
- c Este código reduz o tamanho do fragmento padrão 1-500 caracteres.
- d Aqui nós tokenize o texto original, usando StandardAnalyzer.
- e Destacamos os cinco melhores fragmentos, separando-as com reticências (...).
- f Finalmente, escrever o HTML destaque para um arquivo, como mostrado na figura 8.15.

Em nenhum dos nossos exemplos que nós realizar uma pesquisa e destacar hits real. O texto para destacar foi codificado. Isso levanta uma questão importante quando se lida com o Highlighter: onde conseguir o texto para destacar. Esta questão é abordada no próxima seção.

### 8.7.2 Hits Destacando

Se para armazenar o texto original campo no índice é com você (ver secção 2.2 para as opções de indexação de campo). Se o texto original não é armazenado no índice (geralmente para considerações tamanho), caberá a você a recuperar o texto a ser destacado de sua fonte original. Se o texto original é armazenado com o campo, pode ser recuperado diretamente da Documento obtido a partir de Hits, Como mostrado no-following pedaço de código:

```

IndexSearcher searcher = new IndexSearcher (diretório);

TermQuery query = new TermQuery (Termo de novo ("title", "ação"));
Hits hits = searcher.search (query);

```

```

Artilheiro QueryScorer QueryScorer = new (query);
Highlighter Highlighter Highlighter = new (artilheiro);

for (int i = 0; i < hits.length (); i + +) {
    String title = hits.doc (i) get ("title");

    Stream = TokenStream
        nova SimpleAnalyzer (). tokenStream ("title",
            novo StringReader (título));
    Fragmento String =
        highlighter.getBestFragment (título stream);

    System.out.println (fragmento);
}
    
```

Com o nosso índice livro da amostra, a saída é

```

JUnit em Ação </ B>
Lucene in Action </ B>
Tapeçaria em <B> Action </ B>
    
```

Note que ele ainda era nossa responsabilidade tokenize o texto. Este é duplicado esforço, já que o texto original foi tokenized durante a indexação. No entanto, durante indexação, a informação posicional é descartado (isto é, a posição do caractere de cada termo no texto original, mas os deslocamentos posição prazo são armazenados no índice). Por causa das necessidades computacionais de destacar, que só deve ser utilizado para a hits exibida para o usuário.

## 8.8 filtros Chaining

---

Usando um filtro de pesquisa, como já discutido na seção 5.5, é um mecanismo poderoso para seletivamente estreitando o espaço do documento a ser pesquisado por uma consulta. O Sandbox

contém uma interessante meta filtro no projeto misc, contribuiu por Kelvin Tan, que as cadeias outros filtros juntos e executa AND, OR, XOR, e pouco andnot operações entre elas. ChainedFilter, Como o built-in CachingWrapperFilter, não é um filtro de concreto, que combina uma lista de filtros e executa uma desejada bit-wise operação para cada filtro sucessivos, permitindo combinações sofisticadas.

É ligeiramente envolvidos para demonstrar ChainedFilter porque exige uma diversificada conjunto de dados suficientes para mostrar como os vários cenários de trabalho. Nós criamos um índice

com 500 documentos, incluindo um chave campo com valores de 1 a 500; um data campo com dias sucessivos a partir de 01 de janeiro de 2003, e um proprietário campo com o primeiro

metade dos documentos de propriedade de bob e na segunda metade de propriedade da sue:

```

public class ChainedFilterTest estende TestCase {
    MAX int public static final = 500;
    
```

```

privado Diretório RAMDirectory;
privado IndexSearcher pesquisador;
privado Query;
privado DateFilter dateFilter;
privado QueryFilter bobFilter;
privado QueryFilter sueFilter;

setUp public void () throws Exception {
    diretório = new RAMDirectory ();
    Escritor IndexWriter =
        IndexWriter nova (diretório, novo WhitespaceAnalyzer (), true);

    Calendar cal = Calendar.getInstance ();
    cal.setTimeInMillis (104139720000L); // 2003 01 de janeiro

    for (int i = 0; i <MAX; i + +) {
        Documento doc = new Document ();
        doc.add (Field.Keyword ("chave", "" + (i + 1)));
        doc.add (
            Field.Keyword ("dono", (i <MAX / 2) "bob": "Sue"));
        doc.add (Field.Keyword ("data", cal.getTime ()));
        writer.addDocument (doc);

        cal.add (Calendar.DATE, 1);
    }

    writer.Close ();

    searcher = new IndexSearcher (diretório);

    // Consulta para tudo para tornar a vida mais fácil
    BooleanQuery bq = BooleanQuery new ();
    bq.add (novo TermQuery (Termo de novo ("dono", "bob")), false, false);
    bq.add (novo TermQuery (Termo de novo ("dono", "sue")), false, false);
    query = bq;

    // Data de filtro combina com tudo muito
    Data pastTheEnd = ParseDate ("2099 01 de janeiro");
    dateFilter = DateFilter.Before ("data", pastTheEnd);

    bobFilter = new QueryFilter (
        nova TermQuery (novo Termo ("dono", "bob")));
    sueFilter = new QueryFilter (
        nova TermQuery (novo Termo ("dono", "sue")));
}

// ...
}

```

Além do índice de teste, `setUp` define uma consulta abrangente e alguns filtros para os nossos exemplos. As pesquisas de consulta para os documentos de propriedade ou bob ou sue; usado sem um filtro, ele irá corresponder a todos os 500 documentos. Um abrangente

DateFilter é construído, bem como dois QueryFilters, um para filtrar proprietário bob e outro para processar.

Usando um único filtro aninhado em uma ChainedFilter não tem nenhum efeito além de usar o sem filtro ChainedFilter, Como mostrado aqui com dois dos filtros:

```
testSingleFilter public void () throws Exception {
    Cadeia ChainedFilter = new ChainedFilter (
        novo filtro [] {} dateFilter);
    Hits hits = searcher.search (cadeia de consulta);
    assertEquals (MAX, hits.length ());

    cadeia = new ChainedFilter (novo filtro [] {} bobFilter);
    hits = searcher.search (cadeia de consulta);
    assertEquals (MAX / 2, hits.length ());
}
```

O poder real de ChainedFilter veio quando nós cadeia de vários filtros juntos. A operação padrão é OR, combinando o espaço filtrada, conforme mostrado quando o filtering em bob ou processar:

```
Testor public void () throws Exception {
    Cadeia ChainedFilter = new ChainedFilter (
        novo filtro [] {sueFilte, bob Filter});

    Hits hits = searcher.search (cadeia de consulta);
    assertEquals ("OR corresponde a todos ", MAX, hits.length ());
}
```

Ao invés de aumentar o espaço do documento, e pode ser usado para restringir o espaço:

```
testAND public void () throws Exception {
    Cadeia ChainedFilter = new ChainedFilter (
        novo filtro [] {dateFilter, bobFilter}, ChainedFilter.AND);

    Hits hits = searcher.search (cadeia de consulta);
    assertEquals ("E corresponde apenas bob", MAX / 2, hits.length ());
    assertEquals ("bob", hits.doc (0).get ("dono").);
}
```

O testAND caso de teste mostra que o dateFilter é AND'd com o bobFilter, efetivamente restringir o espaço de busca a documentos de propriedade de bob desde o dateFilter é abrangente. Em outras palavras, a interseção do previsto filtros é o espaço de busca de documentos para a consulta.

```
Conjuntos bit filtro pode ser XOR (exclusivamente OR'd, o que significa um ou outro,
mas não ambos):
testXOR public void () throws Exception {
    Cadeia ChainedFilter = new ChainedFilter (
        novo filtro [] {dateFilter, bobFilter}, ChainedFilter.XOR);
```

```

    Hits hits = searcher.search (cadeia de consulta);
    assertEquals ("XOR jogos sue",MAX / 2, hits.length ());
    assertEquals ("processar", hits.doc (0) get ("dono").);
}

```

O dateFilter XOR com bobFilter efetivamente filtros para sue proprietário em nosso teste de dados. E, finalmente, a operação andnot permite que apenas os documentos que correspondem a primeiro filtro, mas não o segundo filtro de passar por:

```

testANDNOT public void () throws Exception {
    Cadeia ChainedFilter = new ChainedFilter (
        novo filtro [] {dateFilter, sueFilter},
        new int [] {ChainedFilter.AND, ChainedFilter.ANDNOT});

    Hits hits = searcher.search (cadeia de consulta);
    assertEquals ("andnot corresponde apenas bob",
        MAX / 2, hits.length ());
    assertEquals ("bob", hits.doc (0) get ("dono").);

}

```

Em testANDNOT, Dada nossos dados de teste, faixa de todos os documentos na data, exceto aqueles propriedade de sue estão disponíveis para a pesquisa, o que estreita-lo para apenas documentos de propriedade da bob.

Dependendo de suas necessidades, o mesmo efeito pode ser obtido através da combinação de consultas cláusulas em um BooleanQuery ou usando o novo FilteredQuery (Ver secção 6.4.1, página 212). Tenha em mente as advertências de desempenho para uso de filtros, e, se você estiver filtros de reutilização sem alterar o índice, certifique-se que você está usando um filtro de cache. Acorrentado-Filter não cache, mas envolvê-lo em um CachingWrappingFilter vontade cuidar desse aspecto.

## 8.9 Armazenando um índice em Berkeley DB

---

A low-key Chandler projeto (<http://www.osafoundation.org>) é um esforço contínuo para construir um open-source Personal Information Manager. Chandler pretende manter tipos de idades variadas de informações, como email, mensagens instantâneas, compromissos, contatos, tarefas, notas, páginas web, blogs, bookmarks, fotos e muito mais. É uma plataforma extensível, e não apenas uma aplicação. Como você suspeita, a pesquisa é uma componente social à infra-estrutura Chandler.

Repositório subjacente Chandler usa Berkeley DB da Sleepycat em um muito diferente forma diferentes do que um banco de dados relacional tradicional, inspirada RDF e associativos bancos de dados. O codebase Chandler usa Python principalmente, com ganchos para nativos código, quando necessário. Vamos pular direto para os desenvolvedores como Chandler usar Lucene; consulte o site Chandler para mais detalhes sobre este projeto fascinante.

Lucene é compilado para a plataforma nativa usando GCJ e é acessado a partir de Python através SWIG. Lupy (a porta do Lucene Python) foi considerada, mas para uma velocidade abordagem mais nativa foi considerado mais apropriado.

Andi Vajda, um dos principais desenvolvedores de Chandler, criou um diretório Lucene implementação que usa Berkeley DB como o mecanismo de armazenamento subjacente. Um interessante efeito colateral de ter um índice Lucene em um banco de dados é a transacional apoio que presta. Andi doou sua implementação para o projeto Lucene, e é mantido na área de contribuições Db do Sandbox. O projeto Chandler também de código aberto seu código PyLucene, que é discutido na seção 9.6.

### 8.9.1 Codificação para DbDirectory

`DbDirectory` está mais envolvido do que usar o built-in `RAMDirectory` e `FSDirectory`, conservador. Ela requer a construção e gestão de dois Berkeley DB objetos Java API, `DbEnv` e `Db`. Listagem 8.8 mostra `DbDirectory` sendo utilizada para a indexação.

#### Indexação de listagem 8.8 com DbDirectory

```
BerkeleyDbIndexer public class {
    public static void main (String [] args)
        throws IOException, DbException {
        if (args.length! = 1) {
            System.err.println ("Uso: dir <index BerkeleyDbIndexer");
            System.exit (-1);
        }
        String indexDir = args [0];

        DbEnv env = new DbEnv (0);
        Db = new índice Db (env, 0);
        Blocos db = new Db (env, 0);
        DbHome arquivo = new File (indexDir);
        int flags = Db.DB_CREATE;

        if (dbHome.exists ()) {
            File [] files = dbHome.listFiles ();

            for (int i = 0; i < files.length; i + +)
                if (arquivos [i]. getName (). startsWith ("__"))
                    arquivos [i] delete ();
            dbHome.delete ();
        }

        dbHome.mkdir ();

        env.open (indexDir, Db.DB_INIT_MPOOL | bandeiras, 0);
        index.open (null, "__index__", null, Db.DB_BTREE, bandeiras, 0);
        blocks.open (null, "__blocks__", null, Db.DB_BTREE, bandeiras, 0);
```

```
Diretório DbDirectory = new DbDirectory (null, índice, blocos, 0);
Escritor IndexWriter IndexWriter = new (diretório,
    nova StandardAnalyzer (),
    true);

Documento doc = new Document ();
doc.add (Field.Text ("Conteúdo", "A ligeira raposa marrom ..."));
writer.addDocument (doc);

writer.optimize ();
writer.Close ();

index.close (0);
blocks.close (0);
env.close (0);

System.out.println ("Indexação Complete");
}
}
```

Depois de ter uma instância de `DbDirectory`, Utilizando-o com Lucene não é diferente de usar o built-in `Diretório` implementações. Pesquisando com `DbDirectory` usa o mesmo mecanismo, mas você usa o bandeiras valor de 0 para acessar um já criado índice.

### 8.9.2 Instalando `DbDirectory`

Erik tinha um tempo difícil obter `DbDirectory` de trabalho, principalmente por causa de problemas

com a construção e instalação de Berkeley DB 4.2.52 no Mac OS X. Depois de muitos e-mails frente e para trás com a Andi, os problemas foram resolvidos, eo índice (e unshown exemplo, uma pesquisa) funcionou.

Siga as instruções para obter e instalar o Berkeley DB. Certifique-se de configurar o Berkeley DB construir com suporte a Java habilitado (`. / Configure -enable-java`). Você precisa `db.jar` Berkeley DB, assim como o `DbDirectory` (E amigos) código do Sandbox no seu classpath. Pelo menos no Mac OS X, a definição a variável de ambiente `DYLD_LIBRARY_PATH` to `/usr/local/BerkeleyDB.4.2/lib` foi também necessárias.

## 8,10 Construindo o Sandbox

O repositório Sandbox tem sido historicamente um "pilhas não incluídas" área. Estão em curso trabalhos para melhorar a visibilidade ea facilidade de usar o Sandbox componentes, e esta área pode mudar a partir do momento da redação deste texto até que você leia

este livro. Inicialmente, cada contribuição para a Sandbox tinha seu próprio arquivo de construção Ant e não foi integrado em uma construção comum, mas esta situação tem melhorado, agora, a maioria das peças Sandbox são incorporados em uma infra-estrutura de construção comum.

A menos que a documentação online mais atual diz o contrário, recomendamos que você obtenha os componentes Sandbox diretamente do CVS anônimo Jacarta acesso e quer criar os arquivos JAR e incorporar os binários em seu projeto ou copiar o código fonte desejada em seu projeto e construí-lo diretamente em seus próprios binários.

#### 8.10.1 Check it out

Usando um cliente CVS, siga as instruções fornecidas no local de Jacarta: <http://jakarta.apache.org/site/cvsindex.html>. Especificamente, este envolve a execução do seguintes comandos na linha de comando:

```
Cvs-d%: pserver: anoncvs@cvs.apache.org: / home / login cvspublic  
password: anoncvs
```

```
Cvs-d%: pserver: anoncvs@cvs.apache.org: / home / cvspublic checkout  
jakarta-lucene-sandbox
```

A senha é anoncvs. Este é somente leitura acesso ao repositório. Em seu cur-aluguel de diretório, você terá agora um subdiretório chamado jakarta-lucene-sandbox. Sob esse diretório é um diretório onde todos os contributos goodies discutido aqui, e mais, residir.

#### 8.10.2 Ant no Sandbox

Em seguida, vamos construir os componentes. Você vai precisar de Ant 1.6.x, a fim de executar o Sand-caixa de criar arquivos. Na raiz do diretório de contribuições é um arquivo build.xml. A partir de linha de comando, com o diretório atual jakarta-lucene-sandbox/contribuições, execute formiga. A maioria dos componentes vai construir, testar e criar uma distri- arquivo JAR capaz no subdiretório dist.

Alguns componentes, como javascript, não estão integrados neste processo de construção, por isso você precisa copiar os arquivos necessários em seu projeto. Alguns contribuições ultrapassada ainda estão lá também (estes são os que nós homens não-contribuições ção neste capítulo), e adicionais provavelmente chegar depois nós escrito isso.

Cada contribuição subdiretório, tais como analisadores e formiga, tem o seu próprio arquivo build.xml. Para construir um único componente, definir o seu diretório de trabalho atual para diretório do componente desejado e execute formiga. Esta é ainda uma maneira bastante incipiente de obter suas mãos sobre estes add-ons para o Lucene, mas é útil ter direto

acesso à fonte. Você pode querer usar o Sandbox para idéias e inspiração, não necessariamente para o código exato.

## Resumo 8.11

---

Não reinventar a roda. Alguém provavelmente encontrou a mesma situação você está lutando com-você precisa de idioma específico de análise, ou você quer construir um índice durante um processo de construção Ant, ou você quer termos de consulta em destaque na resultados de pesquisa. O Sandbox e os outros recursos listados na web Lucene site deve ser o seu primeiro pára.

Se você acabar arregaçando as mangas e criar algo novo e, em geral útil, por favor, considere fazer uma doação para a Sandbox ou tornando-o disponível para o Comunidade Lucene. Estamos todos mais do que gratos pela generosidade Doug Cutting para open-sourcing Lucene em si. Por contribuindo também, você se beneficia de um grande número de desenvolvedores qualificados que podem ajudar a rever, depurar e mantê-la, e, mais importante, você pode descansar fácil sabendo que você fez do mundo um lugar melhor!

# Portas Lucene

---

9

## Este capítulo aborda

- Usando as portas Lucene para outras programações línguas
- APIs portas comparando ', recursos e execução

Ao longo dos últimos anos, a popularidade do Lucene tem crescido dramaticamente. Hoje, Lucene é o padrão de fato open-source Java IR biblioteca. Embora os inquéritos têm mostrado que o Java é atualmente a linguagem de programação mais ampla, nem todo mundo usa Java. Felizmente, um número de portas Lucene estão disponíveis em diferentes

rentes línguas para aqueles cuja língua de escolha não é Java.

Neste capítulo, nós vamos dar-lhe uma visão geral de todas as portas atualmente Lucene disponíveis. Nós vamos fornecer breves exemplos de uso dos portos, mas tenha em mente que cada porta é um projeto independente, com sua documentação listas próprias, tutoriais, usuário e comunidade de desenvolvedores que será capaz de fornecer mais informações detalhadas.

## Relação 9,1 Portos "para Lucene

Tabela 9.1 mostra um resumo dos aspectos mais importantes de cada porto. Como você pode ver, as portas ficam atrás Lucene. Não desanime com isso, embora, todos os os projetos porto Lucene estão ativamente desenvolvido.

Tabela 9.1 O resumo de todos os portos existentes Lucene

	CLucene	dotLucene	Plucene	Lupy	PyLucene
Port linguagem	C ++	C #	Perl	Píton	GCJ SWIG +
A versão atual	0.8.11	1,4	1,19	0.2.1	0.9.2
Java versão	1,2	1.4-final	1,3	1.2 (parcial)	1.4 (parcial)
Índice compatível	Sim (1.2)	Sim (1.4)	Sim (1.3)	Sim (1.2)	Sim

Cada uma das portas destaque é atualmente um projeto independente. Isso significa que cada porta tem o seu próprio Web site, listas de discussão, e tudo mais que vai tipicamente juntamente com projetos open-source. Cada porta também tem o seu próprio grupo de fundadores e desenvolvedores.

Apesar de cada porto tenta permanecer em sincronia com a última versão do Lucene, eles todos ficam para trás um pouco. Além disso, a maioria das portas são relativamente jovens, e pelo que pudemos reunir, não há sobreposições comunidade de desenvolvedores. Cada porta leva algum e omite alguns dos conceitos de Lucene, mas porque Lucene foi bem concebido, todos eles imitam sua arquitetura. Há também com-pouca comunicação entre os desenvolvedores dos portos e desenvolvedores Lucene, embora estamos todos conscientes da existência de cada projeto. Isso pode mudar com o tempo, especialmente

pois os autores deste livro gostaria de ver todas as portas se reuniram em torno Lucene , a fim de assegurar o desenvolvimento paralelo, uma comunidade mais forte, API mínima mudanças, um formato de índice compatível, e assim por diante. Com isto dito, vamos olhar para cada porta, começando com CLucene.

## 9.2 CLucene

---

CLucene é a porta Ben van Klinken é open-source da Apache Jakarta Lucene para C + +. É lançado sob a licença LGPL e hospedado no <http://sourceforge.net/projects/clucene/>. Ben é um australiano buscando um mestrado em International Relações e Politics asiáticos. Embora os estudos não estão em uma tecnologia relacionada com campo, ele tem forte interesse em Recuperação de Informação. Ben teve a gentileza de fornecer esta visão geral do CLucene.

A versão atual do CLucene é 0.8.11, é baseado em Lucene versão 1.2. Devido a problemas de Unicode (descrito mais tarde), existem alguns problemas de compatibilidade

Linux entre não-Unicode índices e índices Unicode. Baseado em Linux CLucene lerá índices Unicode, mas pode produzir resultados estranhos. A versão compilada para a plataforma Microsoft Windows não tem problemas com o suporte Unicode.

O pacote de distribuição de CLucene inclui muitos dos mesmos componentes como Lucene, tais como testes e exemplos demo. Ele também contém wrappers que permitem CLucene para ser usado com outras linguagens de programação. Atualmente, existem wrappers para PHP., NET (somente leitura), e uma biblioteca de vínculo dinâmico (DLL) que pode ser compartilhada entre os diferentes programas, e wrappers desenvolvidos separadamente para Python e Perl.

### 9.2.1 As plataformas suportadas

CLucene foi inicialmente desenvolvido em Microsoft Visual Studio, mas agora também compilhas no GCC, mingw32, e (alegadamente) o Borland C + + compilador (embora não construir scripts estão sendo distribuídos). Além do Windows MS plataforma, CLucene também tem sido construído com sucesso no Red Hat 9, Mac OS X, e Debian. A equipe CLucene está fazendo uso de multiplataforma SourceForge é com-pilha fazenda para garantir que CLucene compila e roda em tantas plataformas quanto possíveis. A atividade nas listas de desenvolvedores CLucene 'discussão indica que o apoio para AMD64 arquitetura e FreeBSD está sendo adicionado.

### 9.2.2 compatibilidade API

A API CLucene é semelhante ao Lucene. Isto significa que o código escrito em Java pode ser convertido para C + + com bastante facilidade. A desvantagem é que CLucene não segue

o geralmente aceitos C++ padrões de codificação. No entanto, devido ao número de classes que teria de redesenhada, CLucene continua a seguir a "Javaesque" codificação padrão. Esta abordagem permite também que grande parte do código a ser convertido

uso de macros e scripts. Os invólucros CLucene para outros idiomas, que são incluídos na distribuição, todos têm APIs diferentes.

Listagem 9.1 mostra um programa de linha de comando que ilustra a indexação e pesquisar API e sua utilização. Este programa cria índices primeiros de vários documentos com um único conteúdo de campo. Depois disso, ele executa algumas pesquisas contra o gerado índice e imprime os resultados da pesquisa para cada consulta.

#### Listing 9.1 Usando CLucene de IndexWriter e IndexSearcher API

```
int main (int argc, char ** argv) {

    try {
        SimpleAnalyzer * analisador = new SimpleAnalyzer ();
        Escritor IndexWriter (_T ("testIndex"), * analisador, true);

        * wchar_t docs [] = {
            _T ("A, bc d e "),
            _T ("A, bc d e a b c d e "),
            _T ("A, bc d e f g h i j "),
            _T ("Um ce "),
            _T ("E c a "),
            _T ("Um ce um e-c "),
            _T ("Um ce um c b ")
        };

        for (int j = 0; j < 7; j++) {
            Documento * d = new Documento ();
            Field & f = campo:: Texto (_T ("conteúdo"), docs [j]);
            d-> add (f);

            writer.addDocument (* d);
            // Não há necessidade de excluir campos - documento toma posse
            // excluir d;
        }
        writer.Close ();

        IndexSearcher pesquisador (_T ("testIndex"));
        wchar_t * consultas [] = {
            _T ("A b"),
            _T ("\\" A, b \ ""),
            _T ("\\" A b c \ ""),
            _T ("Um c"),
            _T ("\\" Um c \ ""),
            _T ("\\" E um c \ ""),
        };
    }
}
```

```

Hits * hits = NULL;
Parser QueryParser (_T ("conteúdo"), analisador *);

parser.PhraseSlop = 4;
for (int j = 0; j < 6; j++) {

    * Consulta query = & parser.parse (consultas [j]);
    const wchar_t * qryInfo = query-> toString (_T ("Conteúdo"));
    _cout << _T ("Query") << qryInfo << endl;
    excluir qryInfo;

    Hits * hits = & searcher.search (* consulta);
    _cout << hits-> Length () << _T (" resultados total") << endl;
    _T AM
    for (int i = 0; i < hits-> Length () & & i < 10; i++) {
        Documento * d = & hits-> doc (i);
        cout << i << _T ("") << hits-> score (i) <<
        _T ("") << D-> get (_T ("Corpo"));
    }
    excluir hits;
    excluir da consulta;
}

searcher.close ();
if (analisador)
    excluir analisador;
} Catch (THROW_TYPE e) {
    _cout << _T ("pegou uma exceção:") <<
    e.what () << _T ("\n");
} Catch (...){
    _cout << _T ("pego uma exceção desconhecida \n");
}

```

Muitas aplicações têm de lidar com caracteres fora do intervalo ASCII. Vamos olhar para algumas questões relacionadas com o Unicode mencionados anteriormente.

### 9.2.3 suporte a Unicode

CLucene foi originalmente escrito para ser tão rápido e leve possível. No interesse de velocidade, a decisão foi tomada para não incorporar qualquer bibliotecas externas para manipulação de string e contagem de referência. No entanto, existem algumas desvantagens

para isso. Linux sofre com a falta de suporte a Unicode bom, e desde CLucene não usa bibliotecas externas, Linux constrói teve que ser construída sem Unicode. Este levou a CLucene usando o `_UNICODE` pré-processador directiva: Quando é especificado, os caracteres Unicode são usados, caso contrário, não-Unicode (estreito) personagens são utilizada. No entanto, suporte para Unicode está incluído no CLucene e pode ser ativado

em tempo de compilação. Versão futura também pode resolver este problema, opcionalmente, inclusão de uma biblioteca Unicode.

#### 9.2.4 Performance

De acordo com alguns relatórios capturados nos arquivos dos Desenvolvedores Lucene mailing list, documentos CLucene índices mais rápido do que Lucene. Nós não fizemos qualquer benchmarks nós mesmos, pois isso exigiria voltar para a versão 1.2 de Lucene (não algo que um usuário novo Lucene faria).

#### 9.2.5 Usuários

Embora a porta CLucene tem sido em torno de um tempo e tem um usuário ativo lista de discussão, não temos sido capazes de localizar muitos usuários reais CLucene para listar aqui.

Isto pode ser devido ao fato de que a equipe de desenvolvimento CLucene é pequeno e tem um tempo duro manter-se com recursos que estão sendo adicionados ao Lucene. Nós encontramos para fora sobre Awasu, uma ferramenta de gestão do conhecimento pessoal que usa CLucene sob as tampas (<http://www.awasu.com/>).

### 9.3 dotLucene

---

Quando escrevi este capítulo, discutimos uma porta .NET chamado de Lucene Lucene.Net. Infelizmente, as pessoas por trás Lucene.Net decidiu retirar seu porto e seu código-fonte do site SourceForge, onde o projeto foi hospedado. No entanto, Lucene.Net foi liberado sob a licença Apache Software (ASL), que tornou possível para um novo grupo de desenvolvedores para assumir a projeto. Esta nova encarnação do porto .NET é dotLucene, e você pode encontrá-lo em <http://www.sourceforge.net/projects/dotlucene/>. O pacote de distribuição de dotLucene consiste dos mesmos componentes que o pacote de distribuição de Lucene. Ela inclui o código fonte, testes e exemplos demo poucos.

Além dotLucene, há uma outra porta do Lucene para o .NET plataforma: NLucene, que está hospedado no <http://www.sourceforge.net/projects/nlucene/>. No entanto, essa porta parece irremediavelmente fora da versão data-o último foi lançado no verão de 2002-e assim não merece uma cobertura completa.

#### 9.3.1 compatibilidade API

Embora seja escrito em C #, dotLucene expõe uma API que é quase idêntico ao a do Lucene. Conseqüentemente, o código escrito para Lucene pode ser portado para C # com o mínimo esforço. Esta compatibilidade também permite que os desenvolvedores .NET para utilizar documento para a versão Java, como este livro.

A diferença é limitada aos estilos de Java e C # nomeação. Considerando Java nomes de métodos começam com letras minúsculas, a versão .NET C # usa o naming estilo em que nomes de métodos tipicamente começam com letras maiúsculas.

### 9.3.2 Índice de compatibilidade

dotLucene é compatível com o Lucene ao nível do índice. Ou seja, um índice criado por Lucene pode ser lido por dotLucene e vice-versa. Claro que, como Lucene evolui, os índices entre as versões do Lucene em si não pode ser portátil, de modo que este compatibilidade é atualmente limitado a versão 1.4 do Lucene.

### 9.3.3 Performance

Os desenvolvedores do dotLucene não têm qualquer números de desempenho, neste momento, e eles estão focados em adicionar funcionalidades ao seu porta para garantir que ele fica tão perto

Lucene possível. No entanto, seria seguro assumir que dotLucene do desempenho é semelhante ao de seu precursor, de acordo com autor Lucene.Net 's, a sua desempenho foi comparável ao do Lucene.

### 9.3.4 Usuários

No decorrer de nossa pesquisa de Lucene.Net, encontramos vários usuários interessante que a porta Lucene. O usuário mais notável é Lucene.Net Lookout Software (<http://www.lookoutsoft.com/Lookout/>), que foi recentemente adquirida pela Microsoft Corporação. É o criador de Lookout, o Microsoft Outlook popular add-on que provides funcionalidade de pesquisa superior ao do recurso do Outlook de busca embutido.

Outro usuário interessante Lucene.Net é Beagle (<http://www.gnome.org/beagle/>), um componente do GNOME para indexação e pesquisa de todos os tipos de arquivos, incluindo fotos. Beagle ainda está em fases muito iniciais de desenvolvimento.

Porque o projeto dotLucene é tão novo, não olhar para os usuários deste novo porto. No entanto, desde a última versão de Lucene.Net foi usado para iniciar o dotLucene projeto, tanto Lookout Software e Beagle são efetivamente usando dotLucene. Furdido, estamos certos de que com o tempo, todos os usuários do porto Lucene.Net migrarão para dotLucene.

## 9.4 Plucene

---

Plucene é uma porta de Perl Lucene, você pode encontrá-lo no CPAN (<http://search.cpan.org/dist/Plucene/>). Versão 1.19 do Plucene foi lançado em Julho de 2004, e é uma linha reta porta da versão 1.3 do Lucene. A maioria dos trabalhos foi feita por Simon Cozens, e

apesar de seu envolvimento com o desenvolvimento Plucene diminuiu, ele permanece envolvidos e ativos na lista de discussão Plucene.

#### 9.4.1 compatibilidade API

Sendo uma porta direta do Lucene, Plucene preserva a API, em grande medida. A única diferença óbvia é no estilo do código de nomenclatura, que segue as normas para a nomenclatura e estrutura dos módulos Perl, classes, métodos, e tal. Na listagem 9.2, você pode ver um exemplo de `IndexWriter` e `IndexSearcher` uso em Plucene.

##### Listagem 9.2 Utilização de Plucene IndexWriter e IndexSearcher API

```

o meu escritor $ = Plucene:: Índice:: Writer-> new ("index / tmp /",
Plucene:: Plugin:: Analyzer:: PorterAnalyzer-> new (), 1);
$ Escritor-> set_mergefactor (100);
while (( $ key, $ value) = each (% hash)) {
    $ Doc = Plucene:: Document-> new;
    $ Doc-> add (Plucene:: Documentos:: Campo-> palavra-chave (id = key> $));
    $ Doc-> add (Plucene:: Documentos:: Campo-> UnStored ('text' => $ value));
    $ Escritor-> add_document ($ doc);
};
$ Escritor-> otimizar;
undef $ escritor;
meu analisador $ = Plucene:: QueryParser-> new ({
    analisador => Plucene:: Plugin:: Analyzer:: PorterAnalyzer-> new (),
    default => "text"
});
my $ queryStr = "+ manga + ginger";
my $ query = $ parser-> parse ($ queryStr);
my $ searcher = Plucene:: Pesquisar:: IndexSearcher-> new ("tmp / index");
my $ hc = Plucene:: Pesquisar:: HitCollector-> new (a cobrar => sub {
    my ($ self pontuação, doc $, $) = @_;
    push @ docs, $ pesquisador-> doc ($ doc);
});
$ Pesquisador-> search_hc ($ query, $ hc);

```

Como você pode dizer a partir da listagem, se você estiver familiarizado com Perl, você será capaz de transpor com facilidade.

Embora a API Plucene se assemelhe ao de Lucene, existem alguns internos diferenças de implementação entre as duas bases de código. Uma diferença é que Lucene usa o método de sobrecarga, enquanto Plucene usa nomes de método diferentes na maioria dos casos. A outra diferença, de acordo com os desenvolvedores Plucene, é que Java usa inteiros de 64 bits de comprimento, mas a maioria das versões Perl usar 32 bits.

#### 9.4.2 Índice de compatibilidade

Segundo o autor Plucene, o índice criado pelo Lucene 1.3 e Plucene 1.19 são compatíveis. Um aplicativo Java que utiliza o Lucene 1.3 será capaz de ler e digerir um índice criado por Plucene 1.19 e vice-versa. Como é o caso de outras portas com índices compatíveis, índices entre as versões do Lucene em si pode não ser portátil Lucene como evolui, assim que esta compatibilidade é restrita a Lucene versão 1.3.

#### 9.4.3 Performance

Versão 1.19 do Plucene é significativamente mais lento do que a versão Java. Um Plucene desenvolvedor atribuiu isso às diferenças em vantagens e desvantagens entre as línguas implementação. Plucene porque é um porto bastante direta, muitos dos Pontos fortes Java hit pontos fracos do Perl. No entanto, segundo a mesma fonte, as correções para problemas de desempenho estão em obras. Alguma atividade recente em Plucene de listas de discussão também sugere que os desenvolvedores estão a abordar questões de desempenho.

#### 9.4.4 Usuários

De acordo com consultores Plucene, Plucene é usado por Gizmodo (<http://www.gizmodo.com/>), um site que analisa os dispositivos de ponta de consumo eletrônicos. É também usado por Twingle (<http://www.twingle.com>), um site web-mail dirigido por Kasei, a empresa que patrocinou o desenvolvimento de Plucene. Plucene também foi integrado no Movable Type, um software de blogging popular.

### 9.5 lupy

---

Lupy é um porto Python puro de Lucene 1.2. Os principais desenvolvedores de lupy são Amir Bakhtiar e Allen Short. Algumas funcionalidades do Lucene core está faltando lupy, tal como `QueryParser`, Alguns dos analisadores, o índice de fusão, de bloqueio, e algumas outros pequenos itens. Embora lupy é uma porta de uma versão Lucene muito antigo, a sua desenvolvedores estão ocupados características acrescentando que deve aproxima-la Lucene 1.4. O versão atual do lupy é 0.2.1, você pode encontrá-lo em <http://www.divmod.org/Home/Projetos/lupy/>.

#### 9.5.1 compatibilidade API

Sintaxe Python lado, API lupy se assemelha a do Lucene. Na listagem 9.3, que mostra como um índice `Documento` com lupy, você vê familiar classes e métodos. No entanto, note que podemos criar `IndexWriter` sem especificar o analisador de que é algo que não podemos fazer em Lucene.

**Indexação de listagem um arquivo com 9,3 lupy, e demonstrando lupy API de indexação**

```
IndexWriter de importação lupy.index.IndexWriter
de documento de importação lupy

índice # aberto para escrita
indexador IndexWriter = ('/tmp/index', True)

# Criar documento
d = document.Document()

# Adicionar campos para documentar
f = document.Keyword('filename', fname)
d.add(f)
f = document.Text('title', título)
d.add(f)

# False Pass como o arg 3 para garantir que
# O texto real de s não é armazenado no índice
f = document.Text('text', s, False)
d.add(f)

# Adicionar documento para otimizar o índice, e perto do índice
indexer.addDocument(d)
indexer.optimize()
indexer.close
```

Listagem 9.4 mostra como podemos usar lupy para pesquisar o índice que criamos com o código da listagem 9.3. Depois de abrir o índice com `IndexSearcher`, Criamos um `Prazo` e então um `TermQuery` da mesma forma que faria com Lucene. Depois de executing a consulta, nós loop através de todos os hits e imprimir os resultados.

**Listagem 9,4 Pesquisando um índice com lupy, e demonstrando API em busca de lupy**

```
Prazo de importação lupy.index.Term
lupy.search.IndexSearcher de importação IndexSearcher
TermQuery de importação lupy.search.Term

índice # aberta para pesquisa
Pesquisa IndexSearcher = ('tmp/index/')

# Procurar por 'mango' a palavra no 'texto' campo
t = Prazo('text', 'manga')
q = TermQuery(t)

# Executar a consulta e obter hits
hits = searcher.search(q)
```

```
# Loop através de hits e imprimi-los
para bater em hits:
print 'Encontrado no documento% s (% s)'% (hit.get ('filename'),
hit.get ('title'))
```

---

Como você pode ver, a API lupy sente apenas um pouco diferente do Lucene. Que deve ser desenvolvedores espera-lupy são grandes fãs Python. Independentemente disso, a API é simples e se assemelha a API Lucene de perto.

### 9.5.2 Índice de compatibilidade

Como é o caso com dotLucene e Plucene, um índice criado com lupy é compatível com a do Lucene. Mais uma vez, que a compatibilidade é limitada a um particular versão. Em caso de lupy, os índices são compatíveis com os índices Lucene 1.2's.

### 9.5.3 Performance

Como Plucene, lupy é uma porta direta do Lucene original, o que afeta a sua performance. Não há truques Python específica em lupy para garantir melhor desempenho do porto Python. No entanto, falamos com os desenvolvedores lupy, e em Além de adicionar novas características do Lucene para lupy, eles também estarão abordando problemas de desempenho em versões futuras.

### 9.5.4 Usuários

O usuário principal do lupy é divmod (<http://www.divmod.com/>). Como você pode dizer a partir do URL, este site está relacionado com o site que hospeda projetos lupy.

## 9,6 PyLucene

---

PyLucene é o porto mais recente Lucene, é lançado sob a licença MIT e liderada por Andi Vajda, que também contribuiu Berkeley DbDirectory (ver secção 8.9) para codebase Lucene. Começou como um componente de indexação e busca de Chandler (brevemente descritos na seção 8.9), um open-source extensível PIM, mas foi dividido em um projeto separado em junho de 2004. Você pode encontrar PyLucene em <http://pylucene.osafoundation.org/>.

Tecnicamente falando, não é verdade PyLucene porta. Em vez disso, ele usa o GNU Java Compiler (GCJ) e SWIG para exportar a API Lucene e disponibilizá-lo para um Interpretador Python. GCJ é distribuído como parte da caixa de ferramentas GCC, que pode ser usados para compilar o código Java em uma biblioteca nativa compartilhada. Uma biblioteca compartilhada expõe classes Java como C + + classes, o que torna simples a integração com Python.

SWIG (<http://www.swig.org>) é uma ferramenta de desenvolvimento de software que conecta programas escritos em C e C++ com uma variedade de linguagens de programação de alto nível tais como Python, Perl, Ruby, e assim por diante. PyLucene é essencialmente uma combinação de a saída do GCJ aplicado ao código-fonte e Lucene "ginástica SWIG", como Andi Vajda colocá-lo.

### 9.6.1 compatibilidade API

PyLucene porque era originalmente um componente de Chandler, os seus autores expostos apenas as classes Lucene e métodos que eles precisavam. Conseqüentemente, nem todos os Lucene funcionalidade está disponível em PyLucene. No entanto, como tem PyLucene tornar-se um projeto separado, os usuários começaram a pedir mais dele, então Andi e sua equipe estão lentamente expondo mais do API Lucene via SWIG. Com o tempo, eles pretende expor todas as funcionalidades. Porque adicionar mais recentes recursos Lucene para PyLucene é simples e rápido, a equipe acredita PyLucene PyLucene sempre será ser capaz de permanecer em sincronia com Lucene, o que foi uma das razões de seus desenvolvedores embarcou nele em vez de tentar usar lupy.

Na medida em que sua estrutura está em causa, a API é praticamente o mesmo, o que torna mais fácil para os usuários do Lucene para aprender como usar PyLucene. Outra conveniente efeito colateral é que toda a documentação Lucene existentes podem ser utilizados para o programing com PyLucene.

### 9.6.2 Índice de compatibilidade

Devido à natureza de PyLucene ("compilador e ginástica SWIG"), seus índices são compatíveis com os do Lucene.

### 9.6.3 Performance

O objetivo do projeto PyLucene não é para ser o porto mais rápido Lucene, mas para ser o porto mais próximo. Por causa da abordagem GCJ e SWIG, isso não deve ser difícil conseguir, porque exige menos esforço do que escrevendo manualmente uma porta para outro linguagem de programação. Apesar do fato de que o alto desempenho não é a prioridade, PyLucene supera Lucene, embora não coincide com o desempenho de CLucene.

### 9.6.4 Usuários

Sendo uma porta Lucene muito recente, PyLucene não tem muitos usuários pública ainda. Assim agora, o único projeto sério que sabemos que usa PyLucene é Chandler (<http://www.osafoundation.org/>).

## 9.7 Resumo

---

Neste capítulo, discutimos todos os portos existentes atualmente Lucene conhecido por nós: CLucene, dotLucene, Plucene, Iupy e PyLucene. Olhamos suas APIs, recursos portados, Lucene compatibilidade e desempenho em relação ao Lucene, bem como alguns dos usuários de cada porta. O futuro pode trazer Lucene adicionais portos; os desenvolvedores Lucene manter uma lista na Wiki Lucene em <http://wiki.apache.org/jakarta-lucene/>.

Cobrindo as portas Lucene, temos intensificado fora dos limites do núcleo Lucene. No próximo capítulo vamos ainda mais longe ao examinar várias juros caso ing estudos Lucene.

# 10

## Estudos de caso

### Este capítulo aborda

- Usando Lucene no mundo real
- Empresa projeto arquitetônico
- Resposta às preocupações de linguagem
- Manipulação de preocupações de configuração e threading

Uma imagem vale mais que mil palavras. Exemplos de Lucene verdadeiramente "em ação" são inestimável. Lucene é a força motriz por trás de muitas aplicações. Há incontáveis usos de propriedade ou ultra-secreto de Lucene que nunca pode saber, mas também há inúmeras aplicações que podemos ver em acção online. Wiki Lucene tem uma seção intitulada PoweredBy, no [http://wiki.apache.org/jakarta-lucene / PoweredBy](http://wiki.apache.org/jakarta-lucene/PoweredBy), o que listas de muitos sites e produtos que utilizam o Lucene.

API Lucene é simples, quase trivial, para usar. A mágica acontece quando Lucene é usado inteligentemente. Os estudos de caso que se seguem são exemplos de muito inteligente utiliza de Lucene. Ler nas entrelinhas dos detalhes de implementação de cada um deles, e pedir as jóias dentro. Por exemplo, oferece uma Nutch open-source, altamente escalável, a solução de busca full-Internet que deve ajudar a manter Google honesto e em seus deuses. jboss está focada em um único domínio em Java e afinou seu motor de busca esféricamente para a sintaxe Java. SearchBlox oferece uma produto (versão gratuita limitada disponível) com base no Lucene, proporcionando intranet soluções de pesquisa. LingPipe estudo de caso é intensamente acadêmico e assustadoramente poderosa para a análise de domínio com foco linguística. Mostrando o fator inteligência, Michaels.com usa Lucene para indexar e procurar cores. E, finalmente, TheServer-Lado inteligente envolve Lucene com infra-estrutura facilmente configurável, permitindo você facilmente encontrará artigos, análises e discussões sobre temas Java.

Se você é novo no Lucene, ler esses estudos de caso em um nível alto e encobrir todos os detalhes técnicos ou listagens de código; ter uma idéia geral de como está sendo Lucene

usado em um conjunto diversificado de aplicações. Se você é um desenvolvedor experiente Lucene

ou você já digeriu os capítulos anteriores deste livro, você vai desfrutar da técnica detalhes, talvez alguns são empréstimos no valor diretamente para suas aplicações.

Nós estamos muito endividados para os contribuintes desses estudos de caso que Levou tempo fora dos seus horários ocupados para escrever o que você vê no restante do neste capítulo.

## 10.1 Nutch: "O NPR dos motores de busca"

Contribuição de Michael Cafarella

Nutch é um motor de busca open-source que utiliza o Lucene para pesquisar todo o web de documentos, ou em uma forma personalizada para uma intranet ou um subconjunto de na web. Queremos construir um motor de busca que é tão boa quanto qualquer outra coisa que é capaz de: Nutch precisa processar, pelo menos, tantos documentos, busca-los pelo menos como rápido, e ser pelo menos tão confiável, como qualquer motor de busca você já usou.

Há um monte de código em Nutch (o fetcher HTTP, o banco de dados de URL, e assim on), mas pesquisa de texto é claramente no centro de qualquer motor de busca. Grande parte do código e esforço colocado em Nutch existem para apenas duas razões: para ajudar a construir uma Lucene índice, e para ajudar a índice de consulta que.

Na verdade, Nutch usa muitos índices Lucene. O sistema é projetado para escalar conjuntos processo de Web-escala documento (em algum lugar entre 1 e 10 bilhões de documentos). O conjunto é tão grande que ambos indexação e consulta deve ter lugar através lotes de máquinas simultaneamente. Além disso, o sistema no momento da consulta precisa processar Buscas rapidamente, e que necessita para sobreviver, se alguns crash máquinas ou são destruídas.

A arquitetura de consulta Nutch é bastante simples, eo protocolo pode ser descrito em apenas alguns passos:

- 1 Um servidor HTTP recebe a solicitação do usuário. Existe algum código Nutch correndo lá como um servlet, chamado de manipulador de Consulta. O Han-Query manobrador é responsável por devolver o HTML página de resultados em resposta à pedido do usuário.
- 2 O Handler Query faz algum processamento luz da consulta e encaminha os termos de busca de um grande conjunto de máquinas Índice Searcher. O Nutch sistema de consulta pode parecer muito mais simples do Lucene, mas isso é em grande parte porque os usuários motor de busca tem uma idéia forte de que tipo de consultas eles gostam de executar. Lucene sistema é muito flexível e permite que para muitos diferentes tipos de consultas. O de aparência simples consulta Nutch é convertido em um Lucene muito específicas um. Isto é discutido mais adiante. Cada Index Searcher funciona em paralelo e retorna uma lista ordenada de IDs documento.
- 3 Existem hoje muitas correntes de resultados de pesquisa que voltar ao Handler consulta. O Handler Query organiza os resultados, encontrar o melhor ranking em todos eles. Se qualquer Searcher Índice não retornar resultados depois de um segundo ou dois, ele é ignorado, e da lista de resultados é composta de o repliers sucesso.

### 10.1.1 Mais de profundidade

O Handler Query faz algum processamento muito leve da consulta, como reposição ing fora stop palavras como o e dos. Em seguida, realiza algumas operações, para que Nutch pode funcionar bem em grande escala. É muitos contatos Índice Searchers simultaneamente, porque o conjunto de documentos é muito grande para ser pesquisado por qualquer um.

Na verdade, para todo o sistema robustez, um único segmento do conjunto de documentos será copiado para várias máquinas diferentes. Para cada segmento no conjunto, o Query Manipulador de contatos aleatoriamente um dos Searchers índice que pode pesquisá-lo. Se um

Índice Searcher não pode ser contatado, o manipulador de Consulta marca como indisponíveis para pesquisas futuras. (The Handler Query irá verificar para trás de vez em quando, em caso, a máquina vem novamente disponível.)

Uma pergunta comum de pesquisa projeto do motor é se dividir o total índice de texto por documento ou por termo de pesquisa. Se um único ser Searcher Index responsável por, digamos, todas as ocorrências de papagaio? Ou deve lidar com todas as possíveis

consultas que atingiu o <http://nutch.org> URL?

Nutch decidiu por este último, que definitivamente tem algumas desvantagens.

Segmentação baseada em documento significa que cada pesquisa tem que acertar todos os segmentos;

com prazo baseado em segmentação, o manipulador de consulta poderia simplesmente a frente a uma-sin

gle Índice Searcher e pular a integração step.1

A maior vantagem de segmentar por documento é quando se considera

falhas da máquina. E se um único termo segmento torna-se indisponível? Motor utilizadores de repente não pode obter nenhum resultado para um número não trivial de termos.

Com o

baseados em documentos técnica, uma máquina de mortos simplesmente significa alguma porcentagem de

os documentos indexados serão ignorados durante a busca. Isso não é grande, mas é não catastrófico. Baseados em documentos de segmentação permite que o sistema para manter-chug

ging em face do fracasso.

#### 10.1.2 Outros recursos Nutch

- O Handler Query pede a cada Searcher índice para apenas um pequeno número de documentos (geralmente 10). Como os resultados são integrados a partir de Índice de muitos Pesquisadores, não há necessidade de uma grande quantidade de documentos a partir de qualquer fonte,
- ~~Especialmente quando os navegadores modernos sempre mostram a primeira página de resultados.~~  
complicado antes que seja processed.<sup>2</sup> Cada documento indexado contém três campos: o conteúdo da página web em si, o texto da página URL, e um sintético documento que consiste em todos a âncora texto encontrado em hiperlinks que levam à página web. Cada campo tem um peso diferente. O Handler Query Nutch gera uma consulta Lucene boolean que contém o usuário motor de busca texto em cada um dos três campos.
- Nutch também especialmente combinações de índices de palavras que ocorrem extremamente freqüentemente na web. (Muitos deles são HTTP relacionados frases.) Estes seqüências de palavras ocorrem com tanta freqüência que é a sobrecarga desnecessária para pesquisar

<sup>1</sup> Exceto no caso de consultas de várias palavras, o que exigiria uma quantidade limitada de integração.

<sup>2</sup> Observam os autores ': Veja mais sobre essa expansão consulta na seção 4.9.

cada componente da seqüência de forma independente e, em seguida, encontrar a intersecção. Ao invés de procurar por esses termos como pares de palavras separadas, podemos procurá-los como uma única unidade Nutch deve detectar no índice de tempo. Além disso, antes de contactar o Searcher Index, o manipulador de Query olha para qualquer um dos estas combinações em seqüência de consulta do usuário. Se tal seqüência não ocorrer, palavras que o compõem são aglomerados em um único termo de pesquisa especial.

- O fetcher Nutch / indexador prepara documentos HTML antes da indexação -los com Lucene. Ele usa o analisador NekoHTML para retirar a maioria HTML conteúdo e índices apenas o texto nonmarkup. NekoHTML também é útil para extrair o título de um documento HTML.
- Nutch não usa ou decorrentes aliasing termo de qualquer espécie. Motores de busca historicamente não fez muito decorrentes, mas é uma questão que vem regularmente.
- O Nutch comunicação entre camada de rede (IPC) mantém uma duradoura conexão TCP / IP entre cada consulta e cada Handler Índice Searcher. Há muitos threads simultâneos no Handler Query lado, qualquer das quais pode enviar um convite para o servidor remoto em um determinado endereço.  
O servidor recebe cada pedido e tenta encontrar um serviço registrada sob a corda dada (que é executado em seu próprio segmento). -Request do cliente blocos thread ing até ser notificado pelo código IPC que a resposta do servidor chegou. Se a resposta demora mais do que o tempo limite IPC, o IPC código irá declarar o servidor morto e lançar uma exceção.

## 10,2 Lucene Usando a jGuru

---

Contribuição de Terence Parr

jGuru.com é um site orientado pela comunidade de desenvolvedores Java. Os programadores podem encontrar respostas entre os nossos 6.500 entradas FAQ e fazer perguntas em nossos fóruns. Cada tópico é gerido por um guru (um especialista tópico selecionado por jGuru gestão) que as minas do forum perguntas e respostas procurando tópicos interessantes que ele ou ela pode noivo em uma entrada de FAQ boa. Por exemplo, os autores deste livro, Erik Hatcher e Otis Gospodneti, são gurus da Formiga e Lucene topics, respectivamente, em jGuru. Lançado em dezembro de 1999, agora tem mais jGuru de 300.000 visitantes únicos por mês, cerca de 300.000 usuários registrados, e mais de 2.000.000 de page views por mês.

Embora o site parece bastante simples na parte externa, o servidor é um 110k linha pure-Java behemoth contendo todos os tipos de guloseimas interessantes, tais como a sua

StringTemplate motor (<http://www.antlr.org/stringtemplate/index.tml>) para gerar colaborante páginas web multiskin dinâmico. Apesar de seu tamanho e complexidade, jGuru apenas exerce um baseado em Linux dual-headed 800Mhz servidor Pentium com 1Gb RAM rodando JDK 1.3. Vou limitar minha discussão aqui, no entanto, para usar jGuru de Lucene e outros mecanismos de processamento de texto.

Antes Lucene se tornaram disponíveis, foi utilizada uma pesquisa comercialmente disponível motor que essencialmente necessários para o seu servidor spider site próprio, em vez de diretamente preencher o banco de dados de pesquisa do banco de dados do servidor principal. Spidering levou muitos dias para terminar, mesmo quando o nosso site teve poucas entradas FAQ e usuários. Ao construir índices de pesquisa ing com Lucene diretamente do nosso banco de dados, em vez de spidering, o tempo caiu para cerca de 30 minutos. Além disso, o motor de pesquisa anterior havia para ser instalado separadamente e tiveram seus próprios programação baseada em XML bizarro lange (Veja meu artigo "Os seres humanos não deveriam ter que grok XML" [<http://www-106.ibm.com/developerworks/xml/library/x-sbxm.html>] para minhas opiniões sobre este), tornando o sistema mais complicado e pouco confiável. Lucene, em contraste, é apenas mais um arquivo JAR implantado com nosso servidor.

Esta descrição é uma descrição porcas e parafusos de como usa jGuru Lucene e outras instalações de processamento de texto para fornecer uma boa experiência do usuário.

### 10.2.1 Tópico léxicos e categorização de documentos

Um dos objetivos do projeto de jGuru é fazê-lo provavelmente você receberá uma resposta a sua pergunta. Para fazer isso, tentar aumentar a relação sinal-ruído em nossa fóruns, artigos de aranha a partir de outros sites, e permitir aos usuários filtro de conteúdo de acordo com as preferências tópico. Tudo isso depende de saber algo sobre o tema terminology empregados pelos usuários.

Por exemplo, considere o nosso procedimento de redução de ruído para postagens de fórum. Não há nada pior do que uma questão já respondida, uma questão de banco de dados no fórum Swing, ou uma thread onde as pessoas dizem "Você é um idiota." "Não, \* Você \* re 'um idiota. "Temos bastante sucesso resolveu este problema de seguir a cesso de:

- 1 Se não houver palavras-chave relacionadas com Java no post, perguntar ao usuário para reformular.
- 2 Se o post usa terminologia mais provável a partir de um tema diferente, sugerem o outro tópico provavelmente (s) e deixá-los clique para mover o post para o adequadas fórum.
- 3 Use Lucene para pesquisar entradas existentes FAQ para ver se a questão tem já foi respondida. Se o usuário não vê a resposta certa, ele ou ela manualmente deve clicar em Continuar para realmente apresentar algo para o fórum.

Como sabemos que o léxico (isto é, o vocabulário ou terminologia) para um tópico em particular é? Felizmente, jGuru é um site de domínio específico. Sabemos que Java é o tema principal e que há subtópicos, como JSP. Primeiro, eu o spidered New York Times e outros sites, coletando um pool de genéricos palavras em Inglês. Então eu coletados palavras do nosso sistema de FAQ, imaginando que era Inglês + Java. Fazendo uma diferença conjunto fuzzy, (Java + Inglês)-Inglês, deve resultar em um conjunto de Java

palavras específicas. Usando algo como TFIDF (freqüência do termo, o documento inversa freqüência), eu raciocinei que a maior freqüência foi uma palavra em nossa FAQs e os com menos freqüência que foi utilizada no texto puro Inglês, o mais provável era ser um Java palavra-chave (e vice-versa). Um método semelhante você recebe o Java subtópico lexic-contras. Enquanto o tempo avança, existentes tópico drift léxicos com cada entrada nova FAQ. O léxico correspondente é atualizado automaticamente com qualquer novas palavras e suas freqüências de ocorrência, o operador do servidor não precisa fazer nada , a fim de acompanhar as mudanças no uso da palavra programador.

jGuru snoops outros sites relacionados com Java para artigos, tutoriais, fóruns, e assim por diante

que podem ser de interesse dos usuários jGuru. Não são apenas estes itens indexados pelo Lucene, mas usamos vocabulários nosso tópico para calcular o tópico muito provavelmente (s). Os usuários podem filtrar apenas, digamos, snooped conteúdo JDBC.

### 10.2.2 estrutura de banco de dados da pesquisa

Para Lucene, jGuru tem 4 principais bancos de dados de busca Lucene armazenados em diretórios:

- /Var / data / pesquisa / faq Content-de jGuru FAQs
- / Var / data / pesquisa / forum Content-de jGuru fóruns
- / Var / data / pesquisa / Content-spidered estrangeira de não-jGuru fontes
- / Var / data / pesquisa / guru relacionadas com o conteúdo aos usuários jGuru

Dentro do software de servidor, cada banco tem um nome de recurso de busca similar à uma URL:

- jGuru: forum
- jGuru: faq
- estrangeiro
- jGuru: guru

A razão pela qual temos bancos de dados de pesquisa separada é que podemos reconstruir e pesquisar -los separadamente (mesmo em uma máquina diferente), a corrupção em um banco de dados não afeta os outros, e as pesquisas altamente específicas são frequentemente mais rápido devido à particionamento (procura de FAQs apenas, por exemplo).

O software também tem jGuru grupos de recursos, tais como `universo` isso significa todos os recursos de busca. Recursos de pesquisa também pode ter tópicos. Por exemplo, `jGuru:faq / Lucene` indica apenas as entradas Lucene FAQ armazenados no `jGuru` banco de dados.

Dentro dos recursos estrangeiros são sites como o

- estrangeira: devworks
- estrangeira: JavaWorld

As caixas de pesquisa são sensíveis ao contexto, para que ao visualizar uma página de JDBC, você verá o seguinte no formulário HTML para a caixa de pesquisa:

```
<INPUT Type=hidden NAME=resource VALUE="jGuru:faq/JDBC">
<INPUT Type=hidden NAME=resource VALUE="jGuru:forum/JDBC">
```

Isto indica jGuru deve pesquisar somente os associados FAQ / forum com JDBC. Se você está na página inicial ou FAQ Forum zona, você vai ver

```
<INPUT Type=hidden NAME=resource VALUE="jGuru:faq">
<INPUT Type=hidden NAME=resource VALUE="jGuru:forum">
```

Partir da página inicial, você verá:

```
<INPUT Type=hidden NAME=resource VALUE="universe">
```

Além disso, os tópicos relacionados são agrupados de modo que solicitando uma pesquisa, digamos, Servlets

também procura tópicos JSP e Tomcat. O gerente de pesquisa tem defini-predefinidos ções, tais como

```
nova SearchResourceGroup ("jGuru: faq / Servlets",
    "Servlets e FAQs relacionados",
    new String [] {"jGuru: faq / Servlets",
        "JGuru: faq / JSP",
        "JGuru: faq / Tomcat"
    }
)
```

jGuru vai lançar buscas recurso mais múltiplos em paralelo para aproveitar do nosso servidor dual-headed menos que os resultados devem ser fundidos em um único resultado.

Finalmente, é importante notar que os recursos de busca não estão limitados a Lucene dados bases. jGuru tem um número de bisbilhoteiros que raspar resultados sob demanda a partir motores de busca em outros sites. O jGuru consulta e resultados de pesquisa display software não importa onde uma lista de resultados de pesquisa vem.

### 10.2.3 campos Index

Todos os bancos jGuru Lucene têm a mesma forma de consistência, embora alguns campos não são usados dependendo do tipo de entidade indexados. Por exemplo, a foreign-estrangeiras do banco de dados armazena uma pesquisa ID site, mas não é usado no regular jGuru Lucene

banco de dados. Alguns campos são utilizados para exibição, e alguns são usados para pesquisa. O lista completa dos campos é mostrado na tabela 10.1.

Tabela 10.1 Índice jGuru campos Lucene

Nome do campo	Descrição
EID	Palavra-chave utilizada como identificador único
local	Palavra-chave usada por estrangeiros só db
data	Palavra-chave formato ( <code>DateField.dateToString (...)</code> )
tipo	Palavra-chave (uma palavra) em conjunto {forum, artigo, curso, livro, doc, código, faq, as pessoas}
título	Texto (como o FAQ causa, dentro do Fórum, título do artigo)
link	<code>UnIndexed</code> em jGuru; palavra-chave no db estrangeiros (link para entidade)
descrição	<code>UnIndexed</code> (Para exibição)
tópico	Texto (um ou mais tópicos separados por espaços)
conteúdo	<code>UnStored</code> (O campo de pesquisa principal)

Quando uma entrada é retornada como parte de uma pesquisa, o título, link, data, tipo e campos de descrição são exibidos.

Todas as entradas FAQ, fóruns, artigos estrangeiros, guru bios, e assim por diante usar o `content` campo para armazenar texto indexado. Por exemplo, uma entrada de FAQ fornece a questão, resposta, e quaisquer comentários relacionados como `conteúdo` (Isto é, o texto indexado). O título é definido como a questão FAQ, o link é definido como / faq / view.jsp? EID = n para ID n, e assim por diante. O software de exibição de pesquisa não precisa saber o tipo de uma entidade - ele pode simplesmente imprimir o título, link e descrição.

#### 10.2.4 indexação e elaboração de conteúdo

Há duas coisas que você precisa saber para criar um banco de dados de busca Lucene: como você vai obter informações para aranha, e que o processamento que você vai fazer no texto para aumentar a probabilidade de uma consulta bem sucedida.

Você nunca deve construir um banco de dados de pesquisa por rastreamento do seu próprio site. Usando o

Porta HTTP para obter informações e depois remover cruft HTML quando você tem acesso direto ao banco de dados é insanidade. Não é só transferência direta de informações muito mais rápido, você tem mais controle sobre o que parte do conteúdo é indexado.

índices jGuru novos conteúdos, pois é acrescentada para que você pode postar uma pergunta e depois imediatamente procurar e encontrá-lo ou registo e logo em seguida encontrar o seu nome.

Depois de um banco de dados de pesquisa é construída, é dinamicamente actualizada. Não há nunca um necessidade de aranha a menos que o banco de dados não existe. A automação útil é ter seu senso servidor ausente bancos de dados de pesquisa e construí-los durante a inicialização.

jGuru processos altamente conteúdo antes de deixá-lo índice Lucene. O mesmo processamento ocorre para operações de índice e consulta, caso contrário, as consultas provavelmente

Não é possível encontrar bons resultados. jGuru converte tudo em letras minúsculas, plurais tira, faixas de pontuação, tira de tags HTML (com exceção de trechos de código em `<pre>` tags), e tira Inglês palavras de parada (discutido mais tarde).

Porque conhece o léxico jGuru Java, eu experimentei com a remoção não-Java palavras durante a indexação / consulta. Como se constata, os usuários querem ser capazes de encontrar não-

Palavras-chave, tais como Java quebrado bem como palavras-chave Java, de modo que este recurso foi removido.

Plurais Stripping definitivamente melhorou a precisão das consultas. Você não quer janela e Windows a ser consideradas palavras diferentes, e também os parafusos Lucene informação de freqüência calcula durante a indexação. Eu gradualmente construída a seguinte rotina utilizando a experiência e algumas simples humano e computador análise aplicada ao nosso corpus de entradas FAQ:

```
/ ** A stripper plural útil, mas não é particularmente eficiente * /
stripEnglishPlural public static String (String palavra) {
    // Muito pequena?
    if (word.length () <STRIP_PLURAL_MIN_WORD_SIZE) {
        retorno palavra;
    }
    // Casos especiais
    if (word.equals ("tem") || |
        word.equals ("era") || |
        word.equals ("faz") || |
        word.equals ("vai") || |
        word.equals ("morre") || |
        word.equals ("sim") || |
        word.equals ("pega") || | // significa muito em java / JSP
        word.equals ("seu"))
    {
        retorno palavra;
    }
    String newWord = palavra;
    if (word.endsWith ("sses") || |
        word.endsWith ("xes") || |
        word.endsWith ("hes")) {
        // Remove 'es'
        newWord word.substring (0 word.length, () -2);
    }
    else if (word.endsWith ("s")) {
        // Remove 's', substitua por 'y'
        newWord = word.substring (0 word.length, () -3) + 'y';
    }
}
```

```

        else if (word.endsWith ("s") & &
                  ! Word.endsWith ("ss") & &
                  ! Word.endsWith ("é") & &
                  ! Word.endsWith ("nós") & &
                  ! Word.endsWith ("pos") & &
                  ! Word.endsWith ("ses")) {
            // Remove 's'
            newWord = word.substring (0 word.length, () -1);
        }
        retorno newWord;
    }

}

```

Depois de olhar para o histograma de cerca de 500.000 palavras Inglês Agarrei a partir de vários sites, encontrei a seguinte lista para ser eficaz na redução ruído de indexação:

```

String public static final [] = {EnglishStopWords
    "I", "sobre", "também", "an", "and", "qualquer", "são", "aren", "arent",
    "Ao redor", "como", "at", "ser", "porque", "sido", "antes", "ser",
    "Entre", "ambos", "mas", "por", "pode", "não pode", "não posso", "vir",
    "Poderia", "dia", "fez", "fazer", "doe", "faz", "doesn", "doesnt",
    "Dont", "qualquer um", "mesmo", "cada", "for", "de", "get",
    "Grande", "tinha", "tem", "hasn", "hasnt", "ter", "havn",
    "Havnt", "ele", "help", "seu", "aqui", "ele", "seu", "como",
    "In", "info", "em", "é", "it", "seu", "apenas", "let", "vida",
    "Ao vivo", "muitos", "pode", "me", "a maioria", "muito mais", "deve", "meu",
    "Necessidade", "Não", "de", "on", "um", "apenas", "ou", "outros", "nosso",
    "Por favor", "pergunta", "re", "realmente", "respeito", "disse", "dizer",
    "Ver", "ela", "deverá", "uma vez que", "so", "alguns", "ainda", "história",
    "Such", "tome", "do que", "obrigado", "que", "a", "seu", "eles",
    "Então", "lá", "estes", "eles", "coisa", "aqueles", "pensamento",
    "A", "a", "assim", "a", "dito", "demasiado", "use", "usados",
    "Usa", "usando", "ve", "muito", "quero", "era", "caminho", "nós",
    "Bem", "eram", "o que", "quando", "onde", "que", "quem", "porquê",
    "Irá", "com", "sem", "ganhou", "costume", "iria", "você", "seu"
};

```

Palavras como hasn são o resultado de não sendo despojado de pontuação.

### 10.2.5 Consultas

jGuru trabalha duro para oferecer bons, resultados de pesquisa consistente. De cuidado pré-prepared índices, jGuru noivos palavras de pesquisa e os traduz para Lucene específicas consultas. Esta seção resume como os resultados são exibidos, descreve como consultas são geradas, proporciona uma stripper plural Inglês, e, finalmente, caracteriza jGuru pesquisar palavras do ano de 2002.

### Mostrar resultados de pesquisa e tipos de pesquisa

Independentemente da fonte ou o tipo de entidade, todos os resultados são normalizados para apresentar o título,

link, data e descrição. jGuru pode fornecer resultados mesclados a partir de múltiplas bases de dados e pode fornecer resultados por fonte, tais como entradas no FAQs, entradas nos Fóruns, e assim por diante. Isto é frequentemente útil porque alguns o conteúdo é editado e alguns não é. Você pode querer ignorar o conteúdo inédito, como fóruns ocasionalmente.

As consultas também podem ser limitadas às bases de dados específicos, tópicos, ou sites, especificando um

nome do recurso, como `jGuru: faq / Lucene`.

Manipulação de várias páginas de resultados de pesquisa é um problema interessante. Você faz

Não quero ter que guardar e gerir os resultados da pesquisa em uma variável de sessão para que Página 2 pode ser exibida quando um usuário clica no link da página seguinte. Felizmente, este problema é facilmente resolvido: verifica-se que Lucene é rápido o suficiente apenas para repetir a consulta

cada página de resultados e depois pedir para página nth de resultados.

**NOTA** Gostaríamos de enfatizar a última frase Terrence está aqui: "... Lucene é rápido apenas o suficiente para repetir a consulta em cada página de resultados ...".

Mencionamos esta anterior-  
ly no ponto 3.2.2.

### Consultas de computação Lucene de palavras de pesquisa

jGuru primeira processos de consultas da mesma forma que utiliza para a preparação de texto para

índice. Então, porque Lucene assume uma lógica OR-padrão como os usuários esperam e E a lógica, insere jGuru E entre as palavras da consulta após normal processamento de texto, tais como parar de palavra-remoção. Uma cadeia de busca de "fechar a data-

base "deve encontrar apenas os documentos que contenham ambos os fechar e banco de dados. Se,

No entanto, o número de palavras é maior que um limiar, a consulta é deixada como está.

Como aumenta o número de palavras, a probabilidade de uma correspondência E-condição nada zero abordagens.

Para melhorar ainda mais a precisão de busca, consultas enviadas para Lucene conter termos de

o título e conteúdo. Mais de um campo que corresponde uma consulta, o mais provável você tem um bom jogo. Então, se você procurar por "não a classe found" e há um FAQ título de entrada "Por que eu recebo de classe não encontrada", esta entrada deve obter uma boa pontuação. Se

você só procurou o conteúdo indexado, a entrada FAQ incorretamente obter uma pontuação muito inferior.

jGuru usa um truque para melhorar os resultados finais da pesquisa. Palavras-chave encontradas em uma

consulta, tais como métodos e nomes de classe a partir da API Java são impulsionados para indicar

sua importância. Pegando a lista de palavras-chave da API foi uma simples questão de usar javap nos arquivos. classe e analisar os resultados com ANTLR (<http://www.antlr.org>).

### Características de consulta

jGuru registra todas as consultas de pesquisa, porque ele acabará por usar esse feedback para automaticamente melhorar os resultados de pesquisa (por exemplo, observando que entradas FAQ usuários visitam depois de uma pesquisa-as entradas poderiam ser impulsionados para consultas semelhantes no futuro). Nesta seção, eu coletei algumas estatísticas, o leitor pode achar interessante.

No último ano completo de estatísticas, de 2002, havia 1.381.842 procura total, 554.403 dos quais eram únicos (vis-à-vis `equals()`). Havia 820.800 multi-palavra e 561.042 única palavra pesquisas (cerca de 40%). 829.825 consultas referenciadas bases de dados jGuru especificamente (versus as estrangeiras como developerWorks), com 597.402 buscas em um determinado tópico.

### Strings de busca mais popular Java

Tabela 10.2 mostra os 35 principais termos de um histograma de busca (freqüência de contagem de 1.381.842 pesquisas). Ele fornece uma medida dos termos mais populares em 2002.

Tabela 10.2 Top 35 termos de busca mais popular Java

Freqüência	String de consulta completa	Freqüência	String de consulta completa	Freqüência	String de consulta completa
4527	struts	1796	imprimir	1442	log4j
3897	tomcat	1786	data	1414	jarra
3641	JTable	1756	upload	1403	mod_jk
3371	JTable	1720	classpath	1369	mod_webapp
2950	sessão	1683	imagem	1330	gota
2702	jboss	1650	JTree	1323	apache
2233	jsp	1627	applet	1320	weblogic
2116	jdbc	1559	javascript	1267	formiga
2112	xml	1536	servlet	1246	ejb
1989	JTree	1526	ftp	1229	pool de conexão
1884	javamail	1525	fio	1217	upload de arquivo
1827	web.xml	1476	biscoito		

Considerando apenas as consultas multiword, tabela 10.3 mostra os 35 mais popular pesquisas.

Tabela 10.3 35 mais populares multi-word consultas

Freqüência	String de consulta completa	Freqüência	String de consulta completa	Freqüência	String de consulta completa
1229	pool de conexão	376	arquivo de propriedades	276	botão voltar
1217	upload de arquivo	362	jsp incluem	270	vazamento de memória
741	fluxo de entrada	360	serviços web	266	arquivo de propriedade
618	tempo limite da sessão	360	cópia de arquivo	264	coleta de lixo
603	apache tomcat	355	nt serviço	253	classe interna
553	o pool de conexões	342	download de arquivos	243	chave primária
502	upload de arquivos	336	host virtual	242	sessão jsp
500	ler o arquivo	329	tomcat 4	242	classe não encontrada
494	apache tomcat	327	sem memória	239	JDK 1.4
428	procedimento armazenado	322	página de erro	239	servlet applet
422	java mail	301	http post	233	jsp para a frente
384	arrastar e soltar	281	formato de data		

Quanto aos temas (ver tabela 10.4), os registros revelam o histograma a seguir (truncado a 35 entradas) de 597.402 FAQ total ou tópico do Fórum pesquisas específicas. Naturalmente, tópicos introduzido parcialmente a 2002 são artificialmente menos popular nesta lista.

Tabela 10.4 Top 35 tópicos

Freqüência	FAQ específicos ou Tópico no fórum	Freqüência	FAQ específicos ou Tópico no fórum	Freqüência	FAQ específicos ou Tópico no fórum
191013	Tomcat	22118	AWT	9120	Coleções
129035	JSP	21940	Applets	8242	Tópicos
96480	Servlets	21288	Networking	8183	IntelliJ IDEA
84893	Struts	18100	VAJ	7764	Ferramentas
72015	Balanço	17663	AppServer	7446	JMS
51871	JDBC	17321	XML	7428	I18N
47092	JavaMail	14603	JNI	7269	J2ME

continua na página seguinte

Tabela 10.4 Top 35 tópicos (Continuação)

Freqüência	FAQ específicos ou Tópico no fórum	Freqüência	FAQ específicos ou Tópico no fórum	Freqüência	FAQ específicos ou Tópico no fórum
46471	JavaScript	14373	JBuilder	6828	Linux
38083	EJB	13584	Segurança	5884	Mídia
33765	JavaLanguage	10560	ANTLR	4886	CORBA
33546	Formiga	10090	RMI	4876	Serialização
24075	IO	9395	JNDI		

### 10.2.6 JGuruMultiSearcher

Lucene não tem um objeto padrão para pesquisar vários índices, com diferentes consultas. Porque jGuru precisa pesquisar no banco de dados estrangeiras versus a sua bases de dados de pesquisa interna com os termos de consulta um pouco diferente, fiz uma subclasse de `Lucene MultiSearcher`, `JGuruMultiSearcher` (Mostrado na listagem 10.1), para corrigir a situação.

**NOTA** `JGuruMultiSearcher` usa um pouco de baixo nível API Lucene interna que é não abordados neste livro. Por favor, consulte Javadocs Lucene para mais detalhes sobre `TopDocs` e `ScoreDoc` bem como a `Searcher` interface.

#### Índices de listagem 10.1 Searching múltiplos com diferentes consultas

```

/ ** Desde multisearcher lucene era final, 3 eu tinha no atacado
 * Copiá-lo para corrigir uma limitação que não se pode
 * Tem várias consultas, por isso, não lucene heterogêneos
 * Busca db.
 */
public class JGuruMultiSearcher estende MultiSearcher {
    Consulta [] consultas = null;

    / ** Cria um buscador que procura <i> pesquisadores </ i>. * /
    pública JGuruMultiSearcher (Searcher [] pesquisadores,
                                Consulta [] consultas) throws IOException {
        super (pesquisadores);
        this.queries = consultas;
    }
}

```

<sup>3</sup> Isto já não é o caso (a partir das Lucene 1.4). `MultiSearcher` foi aberto, e um `Paralelo MultiSearcher` subclasse tenha sido adicionado ao núcleo. No entanto, nada está construído na medida em que performances uma consulta diferente em cada índice e mescla os resultados como este `JGuruMultiSearcher`.

```

    protegidos TopDocs de busca (Query / * ignorado * /,
                                Filtro filtro, int nDocs)
    throws IOException {
    HitQueue hq = new HitQueue (nDocs);
    float minScore = 0.0f;
    int TotalHits = 0;

    / / Pesquisa de cada pesquisador
    for (int i = 0; i < searchers.length; i + +) {
        if (queries [i] == null || searchers [i] == null) {
            continuar;
        }
        TopDocs docs =
            pesquisadores [i] de busca (queries [i], filtro, nDocs).;
        TotalHits + = docs.totalHits; / update / TotalHits
        ScoreDoc [] = scoreDocs.docs.scoreDocs;
        for (int j = 0; j < scoreDocs.length; j + +) {
            / / Merge scoreDocs em hq
            ScoreDoc scoreDoc = scoreDocs [j];
            if (scoreDoc.score > = minScore) {
                scoreDoc.doc + = começa [i] / / converter doc
                hq.put (scoreDoc); fila hit / update /
                if (hq.size () > nDocs) { / / se overfull fila hit
                    hq.pop (); // menor remover hit na fila
                    / / Reset minScore
                    minScore = ((ScoreDoc) hq.top ()) pontuação.;
                }
            Else {}
            break; / / não pontuação mais > minScore
        }
    }
}

ScoreDoc [] = new scoreDocs ScoreDoc [hq.size ()];
for (int i = hq.size () - 1; i > = 0; i -) { / / coloque docs em ordem
    scoreDocs [i] = (ScoreDoc) hq.pop ();
}

return new TopDocs (TotalHits, scoreDocs);
}
}

```

### 10.2.7 Diversos

Lucene faz um monte de arquivos antes de executar um `otimizar ()` às vezes. Nós teve que se nossas descrições de arquivos do Linux para max 4000 com `ulimit-n 4000` de pré-desafogar o sistema de busca de ir insane.<sup>4</sup>

Eu costumava executar uma tarefa agendada no servidor para otimizar o Lucene vários dados bases (tendo o cuidado de sincronizar com inserções de banco de dados). Antes que eu descobri

a questão descritor de arquivo mencionado anteriormente, eu me mudei de otimização para a inserção ponto, ou seja, eu otimizado em cada inserção. Isso não é mais necessário e faz inserções artificialmente lento.

O analisador Lucene string de consulta não é exatamente robusto. Por exemplo, consultando "E o arrastar" com parafusos o primeiro porque é uma palavra parar. O relatório de bug status foi alterado para "não vai resolver" no site web, estranhamente enough.<sup>5</sup> Eventualmente eu construí o meu próprio mecanismo.

## Usando 10,3 Lucene em SearchBlox

---

Contribuição de Robert Selvaraj, SearchBlox Software Inc.

Quando começamos a projetar SearchBlox, tínhamos um objetivo para desenvolver uma aplicação Java 100% ferramenta de busca que é simples de implementar e fácil de gerenciar. Existem inúmeros ferramentas de busca disponíveis no mercado, mas poucos foram projetados com a gestão capacidade da ferramenta em mente. Com a procura de informações tornando-se um crescendo parte de nossas vidas diariamente, é nossa opinião que o gerenciamento é a chave para o adoção generalizada de ferramentas de busca, especialmente em empresas onde a complexidade das ferramentas existentes é a pedra de tropeço importantes na execução de pesquisa aplicações, para não mencionar o custo. Empresas devem ser capazes de implantar pesquisa funcionalidade em questão de minutos, não em meses.

### 10.3.1 Por que escolher o Lucene?

Ao selecionar um mecanismo de indexação e busca de SearchBlox, fomos confrontados com duas opções: ou usar um dos vários toolkits open-source que estão dispostos ou construir o nosso kit de ferramentas de busca própria. Depois de olhar vários toolkits promissor, decidimos usar Lucene. As razões por trás dessa decisão foram

- Desempenho Lucene- oferece o desempenho da pesquisa incrível. Busca típica os tempos são em milisegundos, mesmo para grandes coleções. Isto apesar do fato de que é 100% Java, que é lento em comparação com linguagens como C ++. No setor de pesquisa da indústria, é extremamente importante ter rápidos e relevantes resultados de pesquisa.
- Escalabilidade-Even embora SearchBlox é otimizado para pequenas e médias coleções de documentos de tamanho (<250 mil documentos), a escalabilidade também foi um

<sup>4</sup> Lucene 1.3 adicionado o formato índice composto, tornando a situação lidar com arquivos muito menos de um problema.

<sup>5</sup> Há ainda questões em aberto sobre as palavras de parada e `QueryParser`.

critérios críticos na escolha Lucene. Queríamos manter aberta a opção de apoio maior em coleções SearchBlox em uma data posterior. Lucene é certamente à altura da tarefa em termos de escalabilidade. Nós estamos cientes de um particular projeto onde Lucene está sendo usado para 4.000.000 índice de documento com <100 vezes busca milissegundo.

- Extensa adoção - Usage- de Lucene tem crescido muito nos últimos par de anos. Tornou-se altamente popular com Recuperação de Informação (IR) especialistas que usaram Lucene como o kit de ferramentas de busca para seus projetos.

Isso resultou em uma grande quantidade de Lucene add-on código open-source sendo disponível para realizar várias tarefas especializadas IR. Esta pode ser uma grande bônus quando você deseja oferecer a seus usuários / clientes novos recursos em muito ciclos curtos de desenvolvimento.

### 10.3.2 arquitetura SearchBlox

Figura 10.1 mostra a arquitetura geral de SearchBlox. Em comparação com Lucene, que é uma indexação de texto e busca API, SearchBlox é uma ferramenta de busca completa. Ele

características crawlers integrada, suporte para diferentes tipos de documentos, provisão para

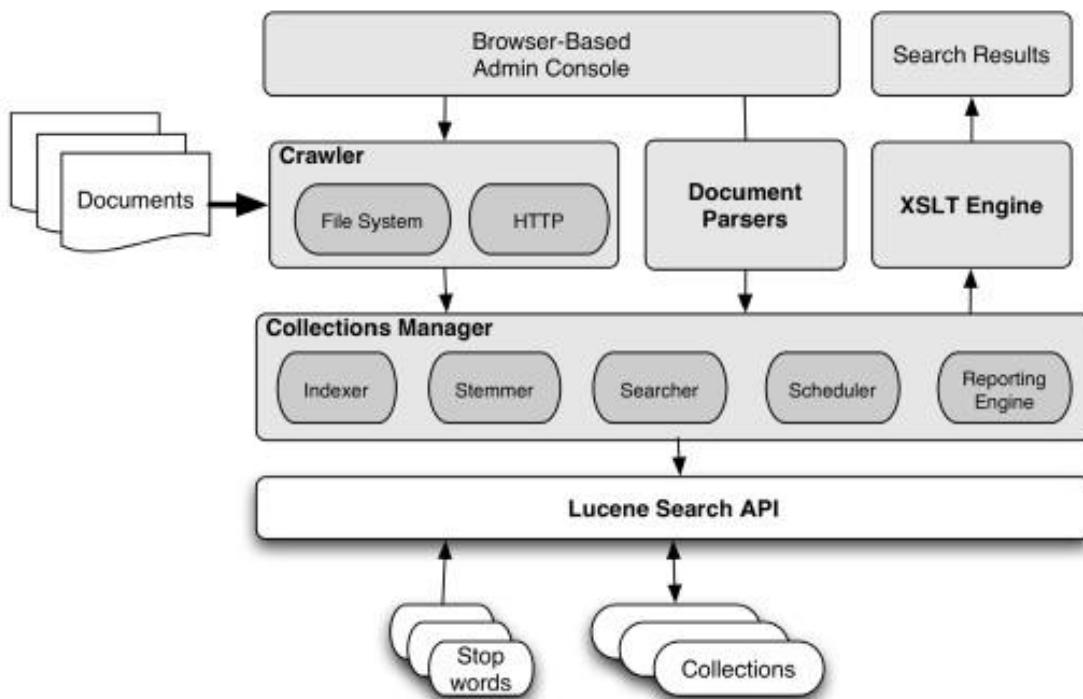


Figura 10.1    SearchBlox sistema de arquitetura

várias línguas, e resultados de pesquisa personalizados, tudo controlado a partir de um navegador

Admin Console baseado. Como uma solução Java puro, SearchBlox podem ser implantados para qualquer container Servlet / JSP, dando ao cliente total flexibilidade na escolha o hardware, sistema operacional e servidor de aplicativos.

### 10.3.3 Resultados da busca

SearchBlox fornece ao usuário a opção de ver os resultados da pesquisa classificadas por relevance, classificadas por data ou ordem alfabética sobre o título. Resultados da pesquisa em SearchBlox são totalmente personalizáveis usando folhas de estilo XSL. A busca resultados também podem ser entregues como XML para ser consumido por um aplicativo externo.

Isto é conseguido através da geração de um documento XML para cada página de resultados de pesquisa. Em página de resultados, cada resultado de pesquisa é representado como mostrado aqui:

```
<resultado no="1">
  <score> 27 </ pontuação>
  <url> http://www.searchblox.com/faqs/question.php?qstId=22 </ url>
  <lastmodified> 24 de novembro de 2003 07:40:15 EST </ LastModified>
  <indexdate> 24 de novembro de 2003 07:40:15 EST <indexdate />
  <tamanho> 7408 </ size>
  <title> SearchBlox <highlight> FAQs <destacar /> </ title>
  <keywords />
  <contenttype> HTML </ contenttype>
  <description> <highlight> FAQs </ destaque>
  Home / Browse Categories / Coleções /
  Porque é que existe um limite no número de coletas?
  Porque é que existe um limite no número de coletas?
  Há um limite no número de coleções, devido à
  motivos de desempenho. Comentários do usuário Porque é que t. ...
  </ Description>
  <idioma> en </ language>
</ Result>
```

Este segmento XML é gerado a partir do Lucene `Hits` objeto. Os dados para o título e descrição campos é então passado através de um `Highlighter` classe para destacar

os termos da consulta. Os termos destacados são marcado com o `<highlight>` tag.

Esse mecanismo dá ao desenvolvedor flexibilidade completa para a personalização de resultados de pesquisa, escolhendo apenas os elementos XML que são de importância para o usuário final.

### 10.3.4 O suporte de idiomas

SearchBlox atualmente suporta 17 idiomas, incluindo japonês, (chinês Simplificado e tradicional) e coreano. Existem dois principais desafios na criação de uma ferramenta de pesquisa que oferece suporte à pesquisa em várias coleções em vários idiomas

- Indexação de documentos com diferentes codificações-A solução para este problema é normalizar a codificação do documento. No caso de SearchBlox, todo o conteúdo é convertidos para UTF-8 antes da indexação. SearchBlox usa vários mecanismos para detectar a codificação do documento que está a ser indexado.
- Detectar o idioma do conteúdo do A- idioma do conteúdo é necessário para duas finalidades quando indexação: para escolher o analisador correta e usar a lista stop-palavras corretas. SearchBlox usa a configuração de idioma especificado no momento da criação coleção como o idioma do conteúdo.

#### 10.3.5 Engine Reportagem

Um elemento chave do SearchBlox é o Mecanismo de Reporting. É fundamental saber o que os usuários finais estão procurando. Ferramentas de busca mais comercial fornecer um relatório ferramenta, que pode ser um analisador de log ou uma ferramenta de banco de dados baseado. Em SearchBlox, o

Motor de relatório é baseado em Lucene. Detalhes de cada consulta de pesquisa são indexados como um documento Lucene. Buscas predefinidos são executados neste índice Lucene para recuperar as estatísticas de relatórios diversos. Este mecanismo de relatórios baseados em Lucene

oferece todas as vantagens de um sistema de comunicação baseado em banco de dados sem-over cabeça de usar um banco de dados.

#### 10.3.6 Resumo

SearchBlox aproveita a API Lucene para oferecer uma ferramenta de pesquisa pura Java. Utilização

Lucene tem permitido SearchBlox para concentrar-se na concepção da usabilidade do ferramenta de busca em vez de desenvolver uma API de busca a partir do zero. Com Java havendo-se um padrão generalizado da empresa ea necessidade crescente de pesquisa, SearchBlox irá fornecer uma ferramenta Lucene realmente utilizável busca incorporar.

### 10,4 inteligência competitiva com Lucene em XM-InformationMinder XtraMind™ da

---

Contribuição de Karsten Konrad, Steinbach Ralf e Stenzhorn Holger

Conhecimento detalhado sobre os concorrentes, mercados, clientes e produtos é uma vantagem estratégica vital para qualquer empresa. Mas na inundação crescente de informação disponível hoje, a informação verdadeiramente relevante não pode ser pesquisado com as metodologias de pesquisa mais comuns, mesmo em um particular, estreita domínio. A este respeito, agregando informações tornou-se por muito mais tempo demorada do que a sua avaliação focada. Sistematica e eficientemente procurar

e identificar todas as correlações importantes, as tendências actuais ea sua evolução a quantidade de dados que entram a cada dia tem se tornado cada vez mais difícil, se não impossível. No entanto, a tarefa primordial de grupos de inteligência nas empresas é exatamente isso: manter o controle de todas as notícias potencialmente importantes atuais e subse-

temente informar marketing, vendas, ou os grupos de planejamento estratégico de qualquer-devel  
volvimentos que podem alterar o ambiente de negócios da empresa ou direção.

Por isso, há alguma necessidade real de sofisticadas ferramentas especializadas que fornecem ajuda na coleta e avaliação de informações para os trabalhadores do conhecimento na departamentos de inteligência. XtraMind Technologies-uma tecnologia e solução provedor para linguagem natural e inteligência artificial orientado para tais sistemas como aqueles usados para a classificação automática de texto, agrupamento de documentos, detec-topic

ção, e assim por diante, desenvolveu o XM-InformationMinder para atingir exatamente essas necessidades em um aplicativo baseado na web cliente / servidor. Uma das tecnologias de grampo

gias que usamos neste aplicativo é Lucene.

XM-InformationMinder foi inicialmente desenvolvido como uma solução personalizada para uma fabricante de medicamentos genéricos grandes alemão. Eles queriam uma ferramenta simples que poderia por um lado supervisionar notícias sobre si mesmos, seus concorrentes, e sua produtos na Internet e, por outro lado recolher informações sobre produtos químicos compostos e formulações utilizadas para medicamentos da empresa em relação pendentes patentes e contra-indicações, por exemplo. O produto final se tornou um aplicação multiuso destinados a inteligência competitiva e produto em qualquer área de negócio.

XM-InformationMinder pode ser basicamente dividido em dois subapplications: primeiro, um web-based portal de acesso a informação que permite ao usuário realizar pesquisas para documentos particulares ou temas, a gestão de documentos encontrados através de categorias, a geração de relatórios para executivos e assim por diante e, segundo, um agente que percorre todo o Internet e reúne a informação a ser preparado e apresentados. Lucene é especialmente importante para nós no portal acesso à informação.

Alguns dos pontos-chave do portal de acesso da informação são

- O texto completo e semelhança busca-XM-InformationMinder suporta-conceito base, fuzzy, pesquisa e semelhança e classifica os resultados por relevância. Em isso, a pesquisa é capaz de tolerar erros ortográficos. Para a busca interativa pelo informações de usuário, todos recuperados e páginas relevantes são então acessíveis através de métodos conceito de navegação.
- Reporting-A função de relatório é usado para transformar informação em declarações qualificados e estratégias. O aplicativo fornece um editor relatório onde o usuário pode inserir e comentar resultados de pesquisa. O

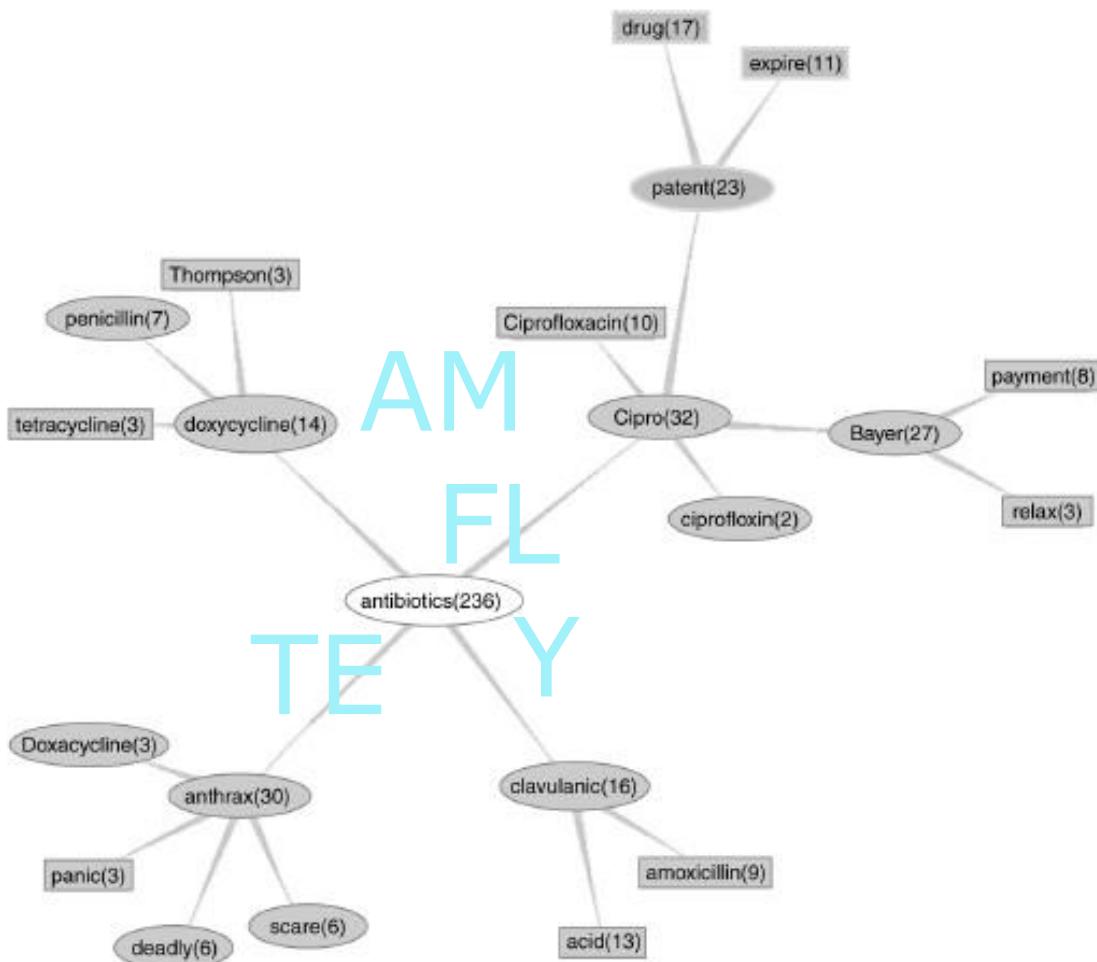


Figura Visualização 10,2 indicando relações entre antibióticos, o antraz, e os anti-antraz drogas Cipro, fabricado pela Bayer

documentos relatório resultante pode ser disponibilizado por e-mail ou através do busca do portal em si.

- Visualização e navegação Search- cordas e termos relacionados são relevantes calculado dinamicamente e são visualizados através de uma interface gráfica de usuário (Ver figura 10.2). Os usuários podem ampliar e controlar a pesquisa de forma interativa e navegar intuitivamente através do espaço de informação. Esta usuários auxiliares que podem não estar ciente de algumas existentes relevantes ligações cruzadas, as relações, ou importante novos tópicos dentro de um determinado domínio e permite ao usuário detectar e explorá-las. Sua visualização através de uma interface do usuário mostra diretamente o conexões entre os documentos dentro de seus contextos. Começando com uma

seqüência de pesquisa, os usuários vêem os documentos relacionados com a sua consulta na forma de um  
rede gráfica. A ferramenta de navegação visual induz a busca e auto-automaticamente se estende a consulta. As relações relevantes para os outros termos são sempre  
calculado em tempo de execução em relação ao termo de pesquisa original e os  
disponíveis

O subapplication segundo é agente um coletor de informações que se baseia  
uma tecnologia de agentes de software inteligentes que são capazes de distinguir  
essenciais a partir de informações não-essenciais. Através de métodos de aprendizado de  
máquina

linguagem e tecnologia, o sistema alcança altos níveis de precisão.

Para treinar o sistema, os usuários fornecem textos relevantes da amostra, que XM-Information-Análises Minder para propriedades comuns dentro de seu conteúdo. Informação XM-Minder, em seguida, usa esse conhecimento adquirido (o chamado modelo de relevância) por sua pesquisa e avaliação de novas informações. Como um segundo conjunto de pistas, os usuários indi-

cate para XM-InformationMinder onde a informação relevante é provável de ser encontrado ou quais as fontes (páginas web, sites concorrentes, portais, newsgroups, newsletters, e assim por diante) devem ser monitorados de forma sistemática. Neste processo, o agente remove

informações redundantes automaticamente através da aprendizagem de máquina mencionado e linguagem tecnologias.

#### 10.4.1 A arquitetura do sistema

XM-InformationMinder foi concebido e desenvolvido como um Java puro, J2EE baseado em aplicações empresariais que podem ser executados em qualquer aplicativo padrão plataforma de servidor.

A partição funcional apresentado anteriormente se reflete diretamente no sistema do design: existe primeiro um agente de software que coleta informações e um segundo portal de busca que fornece essas informações e ajuda na elaboração de relatórios sobre certos desenvolvimentos e tendências. As duas partes são frouxamente ligados via Java Messaging Service, e então isso torna possível distribuir as duas partes e seus tarefas em mais de uma máquina, ou seja, a parte do agente é executado em uma máquina, e parte do portal em outro.

O agente de software é um agente inteligente que usa um texto de aprendizagem de máquina classificador para distinguir entre informação relevante e irrelevante. É basicamente funciona como um web crawler flexível que é capaz de parar o rastreamento sempre que encontrar

a página web atualmente visitados não "interessante" ou adequado. As decisões do agente são baseadas em um modelo de relevância pré-definidos (como mencionado anteriormente), que tem

foram treinados de ambos os exemplos e terminologia da área particular de interesse, neste caso, a indústria de genéricos da indústria farmacêutica. O ponto de partida

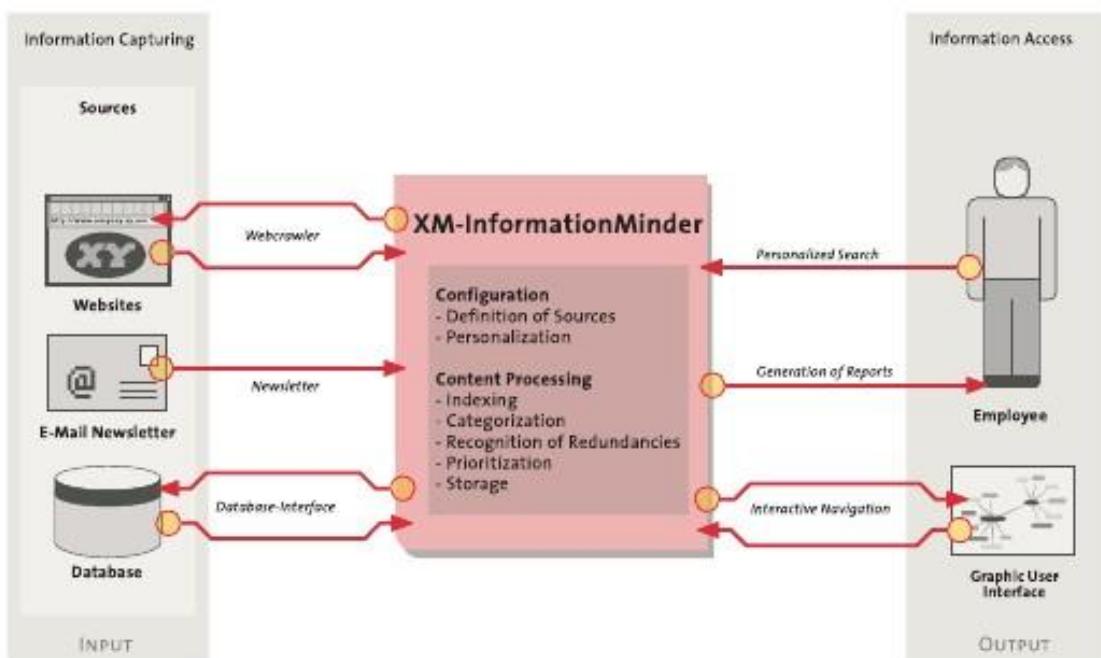


Figura 10.3 Descrição da concepção global e fluxo de trabalho do sistema

pontos para pesquisas do agente são principalmente boletins online a partir de várias notícias agências de notícias que recebe via e-mail serviços subscritos. Ele também reage a mudanças em (user-defined) páginas web importantes, tais como sites de concorrentes web. Além disso, também existem vários newsgroups juntamente com os artigos usuais na indústria (que se encontra, por exemplo, no diretório do Yahoo) que podem ser digitalizados para a entrada adequada para o agente. Veja a figura 10.3.

O atual núcleo de ambas as partes do sistema baseia-se as funcionalidades **PRO-**  
**DESDE** pelo Lucene, com cada empregando seu próprio índice que funciona diretamente. Além disso, cada parte emprega um banco de dados para manter infra-estrutura básica informação, tais como usuário e gerenciamento de configuração. O banco de dados também redundância-  
dantly mantém algumas das informações que podem ser encontrados no índice Lucene para duas razões específicas:

- Falha na recuperação-Se o índice de alguma forma torna-se corrompido (por exemplo, através da falha de disco), ele pode facilmente e rapidamente ser reconstruído a partir dos dados armazenadas no banco de dados sem qualquer perda de informação. Este é alavancado mais
- Envelhecidos pelo fato de que o banco de dados pode residir em uma máquina diferente, na caso em que o aplicativo precisa acessar um determinado documento por uma dada

identificador, o banco de dados pode devolvê-lo mais eficiente do que poderia Lucene. (O identificador é a chave primária de um documento no banco de dados). Se nós empregaria Lucene aqui, teria que pesquisar seu índice geral para o documento com o identificador armazenado em um dos campos do documento.

A interação do agente e as informações subapplications portal de acesso é apresentado na visão geral do fluxo de trabalho seguinte resumo:

- 1 O primeiro agente lê e-mails enviados a ele e inspeciona-las como bem como todas as fontes de dados para outros novos links que ele deve processar.
- 2 O agente executa o processo de rastreamento e busca todos relevantes web páginas, Word, PDF, RTF e outros documentos que podem ser convertidos em texto simples. (Isso geralmente acontece durante a noite por causa da tráfego da Internet reduzida.) Durante este procedimento, todos os dados pesquisados é o primeiro convertidos na mosca do formato original em texto simples e subsequentemente armazenados em um banco de dados para processamento posterior.
- 3 O processo do agente continua alimentando o indexador Lucene com o armazenados dados de conteúdo e colocar os resultados em seu índice correto.
- 4 Após a indexação, o agente envia uma mensagem para a aplicação portal. Este por sua vez, começa a fundir o índice de agente em seu próprio, insere os dados (Incluindo todos os metadados, tais como o tempo de rastreamento de um documento) de banco de dados do agente no banco de dados do portal, e transfere a web armazenados páginas e documentos de um lugar para o portal de acesso.

Um ponto adicional a ser observado é que a manipulação de transações é suportado: Se o-sys dez (agente ou o portal de acesso a informação) ou um usuário adiciona um novo doc-que este documento no índice, então o primeiro agente tenta gravar os dados no banco de dados e depois para o índice. Se um dos dois passos falhar, toda a transação é revertida; fazendo assim, a integridade dos dados é garantida. O mesmo se aplica para o DELE-ção de documentos a partir da aplicação.

Para a nossa aplicação, nós modificamos o motor Lucene de tal forma que ele pode métodos de suporte avançado de busca como o visual mapas de tópicos de pesquisa exploratória e para documentos semelhantes. Porque a informação recolhida pelo agente de software geralmente contém informações redundantes, nós também tivemos que estendeu Lucene do índice ing filtrando redundância automática. Embora Lucene certamente torna-se mais poderoso e flexível com os nossos próprios algoritmos e extensões, o excel-emprestou estruturas básicas de dados e persistência de Lucene são usados sem qualquer modifi-cação. Encontramos Lucene uma plataforma ideal para as nossas tecnologias e ainda se aplicam Funcionalidades do Lucene pesquisa básica de texto completo em nossa aplicação.

#### 10.4.2 Como Lucene tem nos ajudado

Não pode haver nenhuma discussão sobre este assunto: Lucene é uma ferramenta de enviado dos céus para qualquer

Java desenvolvedor que precisa de um open-source, extensível motor de busca de texto completo. Ele

superia caros sistemas disponíveis comercialmente, mas ele vem com um limpo e fácil de entender de design que está aberto a extensões. Negó-XtraMind de ness é, de uma forma para fornecer serviços e soluções em torno avançados natural-processamento de linguagem, e Lucene representa uma informação muito eficiente plataforma de recuperação nesse contexto.

É claro, nós fizemos algumas melhorias Lucene para atender às nossas necessidades.

Primeiro, nós

encontrada a expansão fuzzy (`FuzzyEnum`) Do Lucene para ser muito ineficiente em muito grandes coleções de texto e, portanto, que substituiu o original com um de nossos próprios métodos que reduz o número de palavras expandida. Segundo, nós estendemos o mecanismo resultado coleção com uma fila de prioridade que restringe os conjuntos de resultados para um

alguns hits melhores dúzia de certos demorado pesquisas. Houve quase nenhuma mudança ao núcleo do Lucene, porque todas as alterações que tínhamos de fazer para lan-

pré-processamento calibre poderia ser feito por herança e modificar variantes de classe, principalmente para o `Similaridade classe` e os `HitCollector`. Por exemplo, nós estendemos o `Similaridade classe` através de um método que coleta os termos reais de uma tal consulta que poderíamos fazer destacar e extração de texto com mais facilidade.

Nós desenvolvemos um invólucro para o núcleo do motor Lucene em um adicional de camada de conforto que executa funções adicionais, tais como ortografia automatizado correção e processamento de consultas avançadas, o que nos ajuda a integrar cross-lingual informações recursos de recuperação para o quadro existente.

Dito isto, não temos uma lista de desejos particularmente longo para Lucene. Mas um dos problemas que nós não conseguimos encontrar boa solução foi o rápido geração de extratos para um resultado de pesquisa dada hit. Nós atualmente o índice de documento de conteúdo e, em seguida, retokenize o conteúdo quando geramos texto curto extratos. O problema é que este método não é muito eficiente quando indexação documentos longos: o índice se torna muito grande, ea geração extrair leva até tempo demais. Por isso, gostaríamos de ver algum método introduzido que pode calcular rapidamente uma janela de palavras em torno de um termo de pesquisa para qualquer documento sem ter que aceder ao seu conteúdo.

## 10.5 Alias-i: a variação ortográfica com Lucene

---

Contribuição de Bob Carpenter de Alias-i, Inc.

Usuários que buscam informações em um documento grande coleção são tipicamente interessados em indivíduos, e não tokens. Esta meta é subvertida por dois fatos da linguagem humana: nome de sobrecarga e variação de nome. Nome sobrecarga envolve o uso da mesma nome para se referir a pessoas diferentes. Variação nome envolve o uso de nomes diferentes para se referir à mesma pessoa.

O foco desta discussão é a variação nome e como Lucene pode ser utilizado para melhorar a procura por algo com muitos apelidos. Especificamente, um modelo de termo como um saco de subseqüências de caracteres e armazenar cada saco de seqüências tais como um Documento Lucene. Consultas são igualmente analisado em sacos de caráter subseqüências de que cláusulas de consulta boolean são formadas. Os hits resultante de uma termo de consulta serão os documentos que representam os termos em ordem de fuzzy similaridade string para o termo da consulta. Adiantando um pouco, nós tokenize um termo como como Al Jazeera como subseqüências de comprimento 2-4:

```
"Al", "l", "J", ..., "ah",
:::
"Al' Ja", "l Ja", "Jaz", ..., "erah"
```

Usado como uma consulta, "Al Jazeera" retorna as respostas acima do limiar mostrado na tabela 10.5.

Tabela 10.5 Variação ortográfica usado em uma consulta para "Al Jazeera"

Partitura	Resultado
999	Al Jazeera
787	Al Jazeera
406	Jazeera
331	Al-Jazeera
304	al-Jazeera
259	Al-Jazeera Jazirah
253	al-Jazeera TV
252	Al Jazirah

continua na página seguinte

Tabela 10.5 Variação ortográfica usado em uma consulta para "Al Jazeera" (Continuação)

Partitura	Resultado
222	Árabes do canal Al-Jazeera
213	Al-Jazeera

### 10.5.1 Alias-i arquitetura de aplicação

Alias em-i, que se concentraram na construção de ferramentas e interfaces para ser usado por profissionais analistas de informação, concentrando-se em dois domínios de aplicação: analistas de inteligência do governo de rastreamento notícias do mundo, e biomédica pesquisadores acompanhando a genômica e proteômica literatura de pesquisa. Especializando-se em ing em subdomínios particular é muitas vezes necessário para importar o necessário conhecimento para resolver os problemas introduzidos pela sobrecarga nome e variação.

O fluxo de informações de alto nível e de armazenamento de dados arquitetura do Alias-i Sistema Tracker é ilustrado na figura 10.4.

Do mais alto nível, a arquitetura web padrão de três camadas é organizado formulário da esquerda para a direita na figura 10.4. As camadas consistem de uma interface externa, uma modelo interno (o chamado lógica de negócios), e um armazenamento de dados encapsulados. Doc- processamento que este documento dentro do modelo segue um pipeline com base em fila. Este arranjo foi escolhido principalmente por suas propriedades de escalabilidade. Mesmo dentro de uma única JVM, I / O balanceamento de carga pode ser realizada com a capacidade de resposta muito boa usando as filas lock-split de Doug Lea `util.concurrent` pacote. A fila-arquitetura baseada facilmente escalas para várias máquinas com transacional robusto- ness através da implementação das filas como um Java 2 Enterprise Edition (J2EE) Java Message Serviço (JMS).

Feeds documento reunir documentos de fontes externas e empurrá-los para a primeira fila no pipeline. Implementações atuais incluem um assinante em um padrão Publish / Subscribe, um downloader web-page com base em pesquisas através de API do Google, e um disco baseado em diretório andador. Documentos são transformados por seu manipulador de alimentos, com base na sua proveniência, em nosso XML padronizado formato para documentos de notícias e colocada na fila de documento de entrada. HTML é normalizada com NekoHTML Andy Clark e processado com SAX.

As duas primeiras etapas do pipeline armazenar os documentos e indexá-los. A indexação é realizada com o motor de busca Apache Lucene. O Lucene indexador próprio buffers documentos em um `RAMDirectory`, Usando um segmento separado para fundi-los periodicamente com um índice no disco.

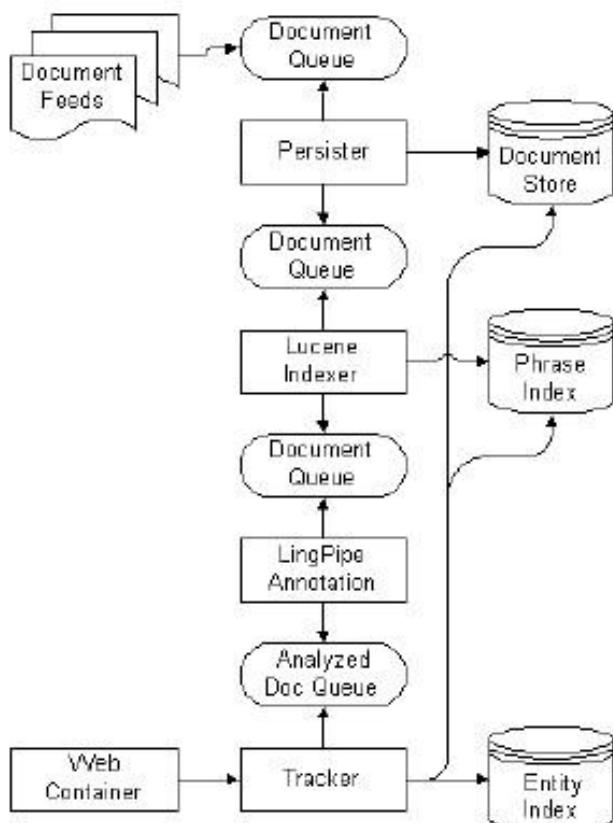


Figura 10.4 Alias-i Tracker arquitetura

As duas próximas etapas do processamento são linguística na natureza. LingPipe, Alias-i do open-source pacote de anotação linguística, é usado para anotar sentença limite-Aries, o extrato de nomes de pessoas relevantes para uma determinada aplicação, e determinar quando dois nomes no mesmo documento se referem ao mesmo indivíduo.

Finalmente, um documento linguisticamente anotada é processado pelo tracker mod-ULE de nomes de cluster que se referem ao mesmo indivíduo em documentos e armazenar as entidades em um banco de dados.

Por exemplo, saída típica ficaria assim:

```

<Document>
  <P>
    <sent>
      <ENAMEX Id="393" type="PERSON"> John Smith </ ENAMEX>
      vive em <ENAMEX id="394" type="LOCATION"> Washington </ ENAMEX>.
    </ Enviadas>
    <sent>
      Mr. <ENAMEX id="393" type="PERSON"> Smith </ ENAMEX> trabalha para
  
```

```

<ENAMEX Id="395" type="ORGANIZATION">
    American Airlines
  </ ENAMEX>.
  </ Enviadas>
</ P>
<DOCUMENTO />

```

Temos aplicado LingPipe para aplicações em vários domínios e línguas, incluindo Hindi, notícias, Espanhol Holandês e Inglês, e genômica Inglês / proteômica. LingPipe é distribuído com um modelo para a notícia que os rótulos Inglês pessoas, lugares e organizações. Também é distribuído com um modelo para Inglês genômica que os rótulos proteínas, DNA, RNA e subcategorizing, eles com base em se eles são uma família, estrutura, molécula, infra-estrutura, ou sítio de ligação, como bem como a rotulagem células e organismos.

A arquitetura de interface de usuário segue o onipresente model-view-controle padrão. O rastreador desempenha o papel de modelo, aceitando o controle em forma de Além documento a partir do back-end da fila de processamento de documentos. Ele também aceita o controle da aplicação do usuário final. O modelo funciona como uma fachada para as bases de dados, fornecendo vistas dos dados à interface front-end. O con-Troller manusear o controle de fluxo aplicativo é implementado como parte da aplicação e executados dentro do container web.

### 10.5.2 variação Ortográfico

Há uma série de causas para a variação nome:

- Erros de ortografia-A termo de consulta, tais como Ferreiro pode ser soletrada accidentalmente como Smth ou Smiht. Motores de busca como o Google agora normalmente fornecem alternativas nativo para termos que aparecem muito pouco provável em comparação com alta variante verossimilhança.
- Ormas Algumas alternativas termos simplesmente ter mais de um convencionalmente ortografia aceitável, seja dentro ou entre dialetos. Considerar cor e cor em Inglês britânico e americano, ou, sobre o mesmo tema, as cores cinza e cinza. Mesmo os nomes próprios podem variar, como MacDonald e McDonald ou O'Leary e Oleary.
- Caráter reduzido sets-Frequentemente termos de idiomas com conjuntos de caracteres que são mais ricos do Inglês, como o alemão, são reduzidas a formas diferentes em Inglês, como Schütze sendo processado como Schuetze ou Schutze, ou o Palavra Inglês naïve alternando com a versão mais simples conjunto de caracteres
- Poème Alternativas e tokenization-Algumas termos variam com respeito a como são tokenized. Isto é particularmente comum com a terminologia biológica,

onde a mesma proteína pode ser referido como SLP-76 ou SLP76. Pessoa nomes sofrem o mesmo problema, com variações como Jun'ichi contra Junichi. Nomes de empresas sofrem o mesmo problema, com os usuários potencialmente writing FooBar Corp como Foo Bar Corp, ou nomes comuns, tais como cybercafe contra cyber café. Prefixos e sufixos mesmo simples às vezes são hifenizadas e às vezes não, como em coordenar vs coordenar, ou em siglas, com IBM alternando com I.B.M..

- Transliteração-Suplente transliteração de nomes estrangeiros leva a variações de ções, tais como traduções para Inglês do russo, como Tschaikowsky, Tchaikovsky, e Chaikovsky, ou de chineses, tais como Looi, Lui, e Loui. Estes podem afectar tokenization também, como na variação entre Abd al e Abdul ou entre Sa anúncio, Sa'ad, e Saad, transliterado do árabe.
- Nome especificidade variação-Há vários níveis de especificidade para nomes, como a proteína genéricos p53 versus suas variedades de bactérias e insetos p53b e p53i.
- Abreviada formulários Biológica terminologia é cheio de semi-padronizados abreviaturas dos termos, como linDNA para linearizado de DNA.
- Morfológicas variação-Há é também uma ampla gama de variações morfológicasção, a partir de sufixos simples como plural gene contra genes de prefixos, como vinculativo, cobinding, e co-ligação.

Um último problema que temos na área biomédica é que parar muitas comum palavras também são siglas ou nomes de genes, e são mesmo escritas em letras minúsculas, como em genes não, se, e para.

Os problemas tokenization e variante são parcialmente amenizada pela norma tokenization, decorrentes, e listas de parar. Por exemplo, p53 irá corresponder p53b se o latter é simbolizada por p, 53, e b. Derivações padrão lidar com alguns dos problemas com variação morfológica, enquanto ao mesmo tempo a introdução de problemas para busca de precisão igualando dois termos que devem ser mantidas separadas e removendo-ing termos relevantes, tais como os nomes gene não, se, e para. Listas de paragem pode lidar com alguns dos problemas com pontuação incompatíveis, se a pontuação é removida da pesquisa.

#### 10.5.3 O modelo de canal ruidoso de correção ortográfica

O modelo de correção de ortografia padrão é baseado em Claude Shannon noisy-modelo de canal (Manning e Schütze, 2003). O modelo de canal ruidoso assume que as mensagens estão sendo escolhidos de acordo com uma distribuição probabilística,  $P(\text{mes-sage})$  E processado como um sinal que é recebido por um ouvinte,  $P(\text{sinal} \mid \text{Mensagem})$ .

Trabalho do destinatário é para decodificar a mensagem pretendida a partir do sinal recuperado. Shannon mostrou que este processo é otimizado no caso em que a mensagem decodificados de um sinal dado é a mensagem que maximiza a probabilidade conjunta

$$P(\text{sinal de mensagem}) = P(\text{mensagem}) * P(\text{sinal} | \text{Mensagem}).$$

Para correção ortográfica, suponho que somos capazes de estimar a probabilidade

$P(\text{ortografia} | \text{palavra})$  de uma ortografia dada uma palavra. Por exemplo,  $P("Jones" | "Jones")$

será muito alto, mas a transposição caso erro de digitação  $P("Jnoes" | "Jones")$  eo exclusão caso erro de digitação  $P("Jone" | "Jones")$  será maior do que uma ortografia totalmente revertida

$P("senoJ" | "Jones")$  ou resultado completamente alheios  $P("Smith" | "Jones")$ . Similarmente  $P("confusible" | "confundido")$  será elevada em um bom modelo porque reduzido Eue um mesmo sor com muitas vezes são confundidas na ortografia. Probabilística modelos deste tipo são normalmente implementadas através de uma noção de distância de edição

(Gusfield 1997).

Haverá também uma probabilidade para as palavras subjacentes em questão. Para exemplo, IBM podem ter uma probabilidade muito maior (digamos, 1 / 1000) do que BM (Digamos,

10/01, 000.000) no texto. Então, quando um termo como BM é visto, medimos  $P("BM") * P("BM" | "BM")$  coi tra  $P("IBM") * P("BM" | "IBM")$ . Em essência, nós perguntar se o probabilidade de um BM vezes a probabilidade de a ortografia da palavra BM como BM é maior ou menor que a probabilidade de IBM vezes a probabilidade de erros de digitação IBM como BM. Além de a primeira hipótese, melhor, o topo N hipóteses também são fáceis de decodificar. Isto é como Microsoft Office é capaz de converter hte em o e para fornecer mais sugestões nos casos em que não é 100% confiante na sua escolha alternativa. Ele também é como o Google é capaz de fornecer sugestões de ortografia alternativa.

#### 10.5.4 O modelo de comparação vetor de variação ortográfica

A popular modelo estatístico (embora não probabilística) para a comparação de duas palavras de similaridade de ortografia envolve comparar subsequências seu caráter (Anglell et al 1983). Uma seqüência de ncaracteres é normalmente chamado de personagem n-grama. O personagem n-gramas para John são os gram-0 ""; o unigrams "J", "o", "h", "n", a bigramas "Jo", "oh", "hn"; trigramas "João" e "ohn", e os 4-gram "John". O termo Jon tem n-gramas "", "J", "o", "n", "Jo", "on" e "Jon".

Na verdade, tem sido demonstrada para pesquisa de texto que a substituição de um texto com todas as seu 4-gramas, e consultas analisar com TF / IDF ponderada vetores termo co-seno, fornece resultados de precisão e recall de busca muito semelhante ao uso de palavra inteira pesquisa (Cavnar 1994). O analisador analisador de consulta e apresentamos seguinte poderia ser usado para implementar um sistema de recuperação completa de informações com base em n-gramas.

### 10.5.5 A subpalavra analisador Lucene

É simples para extrair os n-gramas a partir de uma palavra com Lucene usando um especial analisador ized. A classe stream token mostrado na listagem 10,2 faz o trabalho.

#### Listagem 10,2 n-programação TokenStream

```
private static int MIN_NGRAM = 2;
private static int MAX_NGRAM = 4;

classe public static NGramTokenStream estende TokenStream {
    private int mLength = MIN_NGRAM;
    private int mStart = 0;
    mToken private String final;
    NGramTokenStream pública (String token) {
        mToken = token;
    }
    pública Token next () {
        int consertar = mStart + mLength;
        if (mLength > MAX_NGRAM || emendar mToken.length > ())
            return null;
        MToken.substring String s = (mStart, o MEND);
        Resultado = new Token Token (s, mStart, o MEND);
        if (emendar == mToken.length ()) {
            + + MLength;
            mStart = 0;
        Else {}
            + + MStart;
        }
        resultado de retorno;
    }
}
```

Supondo que temos um método `String readerToString (Reader)` que lê o conteúdo de um leitor em uma string sem jogar exceções, podemos converter o fluxo de token em uma classe de analisador diretamente: 6

```
public static class SubWordAnalyzer Analyzer estende {
    pública TokenStream tokenStream (String fieldName, Reader leitor) {
        Conteúdo String = readerToString (leitor);
        return new NGramTokenStream (conteúdo);
    }
}
```

---

<sup>6</sup> Nota dos autores: A `KeywordAnalyzer` na secção 4.4 converte um `Leitor` a um `Corda` e poderia ser adaptado para uso aqui.

Com o analisador eo conjunto de termos que está interessado, é muito simples para a construção de documentos correspondentes a um acordo com o seguinte método:

```
Índice Directory public static (String [] termos) {
    Directory indexDirectory = new RAMDirectory ();
    IndexWriter indexWriter
        = IndexWriter novo (indexDirectory, novos SubWordAnalyzer (), true);
    for (int i = 0; i < lines.length; + + i) {
        Documento doc = new Document ();
        doc.add (novo Field (NGRAM_FIELD, linhas [i], false, true, true));
        doc.add (
            novo campo (FULL_NAME_FIELD, linhas [i], true, false, false));
        indexWriter.addDocument (doc);
    }
    indexWriter.optimize ();
    indexWriter.close ();
    retorno indexDirectory;
}
```

Note-se que armazena o nome completo no seu próprio campo para exibir os resultados de recuperação. Nós empregam o mesmo n-gram extractor, convertendo os sinais n-gram em prazo cláusulas de consulta:

```
NGramQuery classe public static {estende BooleanQuery
NgramQuery pública (String queryTerm) throws IOException {
    TokenStream tokens = new NGramTokenStream (queryTerm);
    Token token;
    while ((token = tokens.next ())! = null) {
        Termo t = new Prazo (NGRAM_FIELD, token.termText ());
        add (nova TermQuery (t), false, false);
    }
}
```

Note que eles são adicionados à consulta boolean como termos de opcionais que não são nem exigidas, nem proibido de modo que eles contribuirão para a correspondência TF / IDF sup-necidas pelo Lucene. Nós simplesmente estender o `IndexSearcher` para construir no n-gram analisador de consulta:

```
NGramSearcher classe public static {estende IndexSearcher
    pública NGramSearcher (diretório Directory) {
        super (IndexReader.open (indexDirectory));
    }
    Hits pesquisa pública (prazo String) {
        Hits search = (novo NGramQuery (termo));
    }
}
```

A parte agradável sobre esta implementação é que Lucene faz todo o heavy-lift ing nos bastidores. Entre os serviços prestados são TF / IDF ponderação dos n-gram vetores, a indexação de termos de n-gramas, e cosseno de computação e resultado de ordenação.

Aqui está um exemplo de algumas das consultas atropelado 1307 newswire documentos selecionados de uma série de fontes norte-americanas e do Oriente Médio. Entre esses documentos, havia 14.411 pessoas únicas, organizações e locais extraídos por detector de LingPipe entidade nomeada. Estes nomes foram, então, entidade indexado usando 2-gramas, de 3 gramas e 4 gramas. Em seguida, cada um dos nomes foi usado como uma consulta. Tempo de processamento total foi de menos de dois minutos em um modesto pessoal

computador, incluindo o tempo para ler as cordas de um arquivo de índice, eles em memory, otimizar o índice, e em seguida, analisar e executar cada nome como uma consulta e escrever os resultados. Além do um na introdução, considerar o seguimiento resultado. O número de visitas indica o número total de nomes que compartilhados, pelo menos, um n-grama, e só atinge pontuação de 200 ou acima são retornados:

```
Query = Mohammed Saeed al-Sahaf
Número de hits = 7733
1000Mohammed Saeed al-Sahaf
819Muhammed Saeed al-Sahaf
769Mohammed Saeed al-Sahaf
503Mohammed Saeed
493Mohammed al-Sahaf
490Saeed al-Sahaf
448Mohammed Said el-Sahaf
442Muhammad Saeed al-Sahaf
426Mohammed Sa'id al-Sahaf
416Mohammed Sahaf
368Mohamed Said al-Sahaf
341Mohammad Said al-Sahaf
287Mohammad Saeed
270Mohammad Said al-Sahaf
267Muhammad Saeed al-Tantawi
254Mohammed Sadr
254Mohammed Said
252Mohammed Bakr al-Sadr
238Muhammad Said al-Sahaf
227Mohammed Sadeq al-Mallak
219Amer Mohammed al-Rasheed
```

Em cada um desses casos, transliteração do árabe apresenta variação ortográfica que vai bem além da capacidade de um derivado de manusear. Observe também que nem todos os resultados são corretos. Por outro lado, o trabalho de um derivado é handled ordenadamente, como exemplificado pelo

```
Query = Suécia
Número de hits = 2216
1000Sweden
736Swede
277Swedish
```

Em particular, quanto maior a sobreposição substring, o maior dos erros. Por exemplo, "Ministério da Defesa", além de correspondência a variação correta Ministério "da Defesa "no 354, o termo" Analista de Defesa "no 278 e" Bem-Estar Ministério "e "Ministério da Agricultura", ambos em 265. Alias em-i, que mistura nível de caractere modelos com o token de nível modelos para maior precisão.

#### 10.5.6 Precisão, eficiência, e outras aplicações

Precisão pode ser ajustado com precisão / recall trade-offs de várias maneiras. Para começar, termos podem ser escritos em minúscula. Alternativamente, duas variantes, maiúsculas e minúsculas de

n-gramas, com letras maiúsculas em si pode ser fornecido. Além disso, n-gramas podem ser ponderados com base em seu comprimento, que é facilmente suportado pelo Lucene. Com mais n-gramas sendo upweighted, as distribuições retornado será afiada, mas o tempo-token problema de sobreposição torna-se mais pronunciado.

As implementações anteriores são destinados a fins expositivos, e não um aplicação escalável. Para a eficiência, a construção de Símbolo objetos poderiam ser contornado no construtor de consulta. A HitCollector prioridade da fila-base, ou sim-uma face que se aplica um limite, deve reduzir significativamente alocação de objetos durante as consultas. Finalmente, um diretório de arquivos do sistema poderia ser aplicado para armazenar mais dados no disco.

#### 10.5.7 mistura em contexto

Além da variação prazo ortográfica, também consideramos o contexto em que um termo ocorre antes de decidir se dois termos se referem ao mesmo indivíduo. Se o palavras em uma janela em torno do termo em questão são tidos em conta, é bastante possível classificar as três dezenas de Smiths diferentes John aparecendo em dois anos de New York Times artigos com base na similaridade de seus contextos (Bagga e Calvowin 1998). Isso realiza em cerca de 80% de precisão e recall como medida mais as relações entre pares de indivíduos que são os mesmos; portanto, um verdadeiro-positivos é um par de menções que são relacionados, um falso positivo envolve relacionar dois homens-ções que não devem ser ligados, e um falso negativo envolve deixar de relacionar duas menciona que devem ser ligados. Juntos, a variação string eo contexto variação são mescladas em uma pontuação de similaridade geral, para a qual clustering pode ser aplicada para extrair as entidades (Jain e Dubes 1988).

### 10.5.8 Referências

- Alias-i. 2003. LingPipe 1.0. <http://www.aliasi.com/lingpipe>.
- Anglell, R., B. Freund, e P. Willett. 1983. Correção ortográfica automática usando uma medida de similaridade trígrama. *Information Processing & Management* 19 (4) :305-316.
- Bagga, Amit e Breck Baldwin. 1998. Entidade baseada em Cross-Dокументo Coreferencing usando o modelo de espaço vetorial. *Anais do 36 Reunião da Associação de Linguística Computacional*. 79-85.
- Cavnar, William B. 1994. Usando uma representação documento n-gram-base com um modelo de recuperação de vetor de processamento. Em *Proceedings of the Third Text Conferência de recuperação*. 269-277.
- Clark, Andy. 2003. CyberNeko Parser HTML 0.9.3. <http://www.apache.org/~Andyc/ neko / doc / html />.
- Gusfield, Dan. 1997. Algoritmos em Strings, árvores e seqüências: Computer Ciência e Biologia Computacional. Cambridge University Press.
- Jain, Anil K., e Richard C. Dubes. 1988. Algoritmos de agregação de dados. Prentice Hall.
- Lea, Doug. 2003. Visão geral do pacote Release util.concurrent 1.3.4. <http://Gee.cs.oswego.edu / dl / classes / EDU / oswego / cs / dl / util / concorrentes / intro.html>.
- Manning, Christopher D., e Hinrich Schütze. 2003. Fundações de Estatística-Processamento de Linguagem Natural estatística. MIT Press.
- Sun Microsystems. 2003. J2EE Java Message Service (JMS). <http://java.sun.com / products / jms />.

### 10,6 busca Artful em Michaels.com

---

Contribuição de Paredes Craig

Michaels.com é a presença online de Michaels Stores, Inc., uma artes e ofícios varejista com mais de 800 lojas nos Estados Unidos e Canadá. Usando este site web, Michaels alvos entusiastas crafting com artigos, idéias de projetos, e informações sobre o produto destinado a promover o passatempo crafting ea Michaels da marca. Além disso, Michaels.com também oferece uma seleção de mais de 20.000 cópias da arte para compra online.

Com uma oferta tão vasta de idéias e produtos, Michaels.com exige rápida e facilidade da busca robusta para ajudar seus clientes a localizar as informações que eles precisa para desfrutar de seu ofício.

Quando lançado, Michaels.com empregou uma abordagem ingênua para a pesquisa. Com todo o conteúdo do site é armazenado em um banco de dados relacional, eles usaram básicos SQL

consultas envolvendo `LIKE` cláusulas. Porque as tabelas de conteúdo eram muito grandes e contidos colunas longas, procurando desta forma foi muito lento. Mais-mais, pesquisas complexas envolvendo múltiplos critérios não eram possíveis.

Percebendo as limitações da pesquisa por SQL, Michaels.com virou-se para um-com solução de pesquisa comerciais. Embora esta ferramenta oferecida uma ferramenta de pesquisa melhorada

pesquisar sobre SQL, ele ainda não era ideal. Resultados da pesquisa foram muitas vezes inconsistentes,

omitindo os itens que deveriam ter acompanhado os critérios de pesquisa. Reconstrução do índice de pesquisa envolvidos, tendo o mecanismo de busca offline. E, para fazer matérias suporte pior, documentação e técnicas para o produto veio faltando.

Depois de muita frustração com o produto comercial, Michaels.com começou a procurar um substituto. Os seguintes critérios foram definidos para encontrar um substituto adequado:

- Qualquer desempenho busca, não importa o quanto complicado, deve retornar resultados rapidamente. Embora rapidamente nunca foi quantificado, entendeu-se que web surfistas são impacientes e que qualquer pesquisa que levou mais tempo do que alguns segundos iria durar mais do que a paciência do cliente.
- A escalabilidade ferramenta deve ser bem dimensionada, tanto em termos da quantidade de dados indexados, bem como com a carga do site durante o pico de tráfego.
- A robustez-índice deve ser reconstruído freqüentemente sem levá-lo a busca offline facilmente.

Após um período breve avaliação, Michaels.com escolheu Lucene para cumprir suas requisitos de pesquisa. O que se segue é uma descrição de como Lucene drives Michaels.com "s facilidade da busca.

#### 10.6.1 indexação de conteúdo

Michaels.com tem quatro tipos de conteúdo pesquisável: cópias da arte, artigos, na loja produtos de informação e projetos.

Todos os tipos de pesquisa estão indexados em Lucene com um documento que contenha, pelo menos, dois campos: um `ID` campo e um `palavras-chave` de campo. Embora Lucene é usada para pesquisar em Michaels.com, um banco de dados relacional contém o conteúdo real. Portanto, o `ID` campo em cada documento Lucene contém o valor da chave primária da

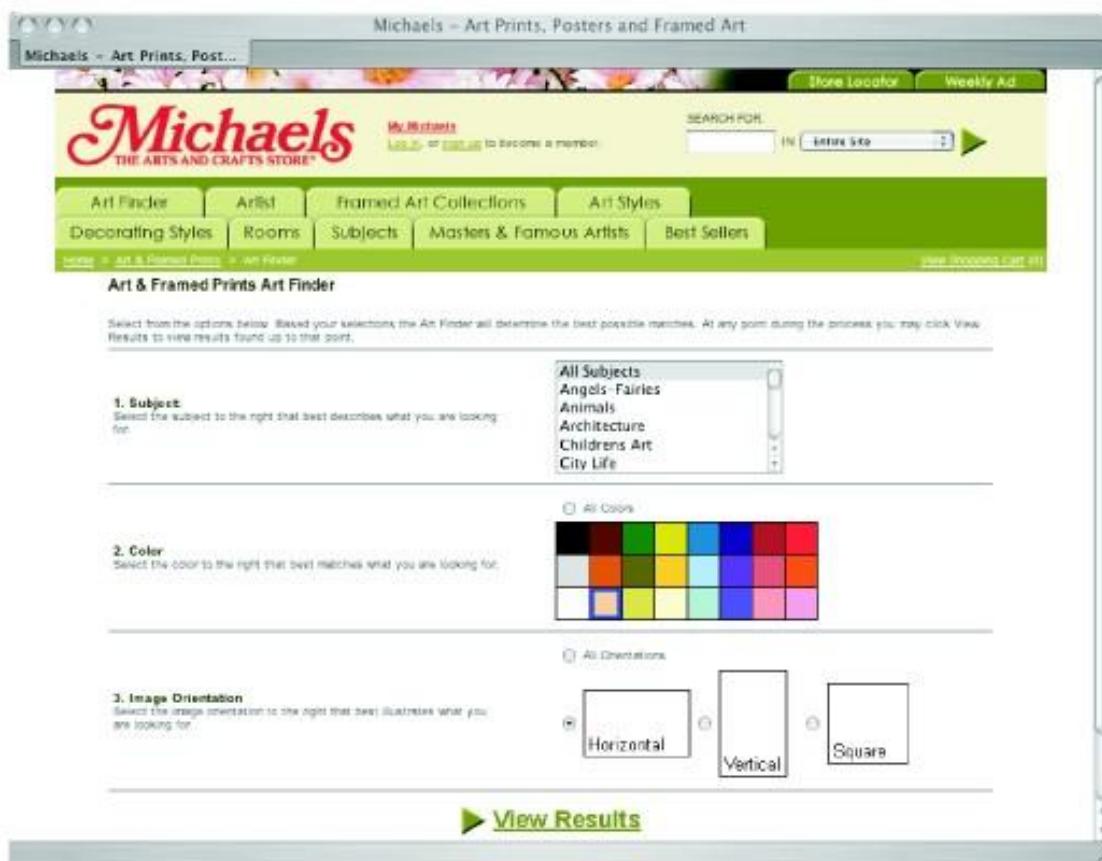


Figura 10.5 O Michaels.com Art ferramenta de busca Finder

conteúdo do banco de dados. O campo contém uma ou mais palavras que podem ser procurada.

Cópias da arte têm requisitos especiais para além de busca simples palavra-chave de busca ing. Michaels.com oferece uma ferramenta Finder Art (figura 10.5) que permite que uma cópia da arte

cliente para localizar uma impressão adequada com base em um ou mais de um print de orientação (paisagem, retrato, ou quadrado), assunto e cores dominantes. Como tal, um cópia da arte está indexada em Lucene com um documento que contém assunto, orientação, e campos de cor, além de os campos ID e palavras-chave.

#### Análise de texto palavra-chave

Uma das exigências colocadas sobre Michaels.com 's Motor de pesquisa foi a capacidade para combinar termos de pesquisa contra sinônimos e erros ortográficos comuns. Para exemplo, a linha de produtos Xyron crafting é muito popular entre scrapbookers e outros papel crafting entusiastas. Infelizmente, muitos visitantes

Michaels.com equivocadamente feitiço Xyron como parece: Zyon. Para permitir que esses usuários

encontrar a informação que eles estão procurando, pesquisa Michaels.com 's deve ser perdoando deste erro de ortografia.

Para acomodar isso, a equipe de desenvolvimento criou um costume Michaels.com Lucene analisador chamado `AliasAnalyzer`. Um `AliasAnalyzer` começa com uma `Alpha-numericTokenizer` (Uma subclasse de `org.apache.lucene.analysis.LetterTokenizer` que também aceita dígitos numéricos em um token) para quebrar a seqüência de palavras-chave em individuais. O fluxo de token é então passado através de uma cadeia de filtros, incluindo `org.apache.lucene.analysis.LowerCaseFilter`, `org.apache.lucene.analysis.StopFilter` e `org.apache.lucene.analysis.PorterStemFilter`. O último filtro aplicado ao fluxo de token é um costume `AliasFilter` (Listagem 10.3) que procura um aliases token a partir de um arquivo de propriedade e introduz o aliases (se houver) para o fluxo de token.

#### Listagem 10.3 AliasFilter introduz tokens sinônimo para o fluxo de token

```

classe AliasFilter estende TokenFilter {
    private final static MultiMap ALIAS_MAP = new MultiHashMap ();
    Stack privada currentTokenAliases = new Stack ();

    static {
        ResourceBundle aliasBundle = ResourceBundle.getBundle ("alias");
        Chaves enumeração = aliasBundle.getKeys ();

        while (keys.hasMoreElements ()) {
            Chave String = keys.nextElement (String) ();
            loadAlias (key, aliasBundle.getString (key));
        }
    }

    private static void loadAlias (String palavra, aliases String) {

        StringTokenizer tokenizer = new StringTokenizer (aliases);
        while (tokenizer.hasMoreTokens ()) {
            String token = tokenizer.nextToken ();
            ALIAS_MAP.put (word, token);
        }
    }
}

Alias lista de carga de
arquivo de propriedades

AliasFilter (TokenStream stream) {
    super (stream);
}

Permitir aliasing bidirecional

    ALIAS_MAP.put (token, palavra);
}

pública Token next () throws IOException {
    if (currentTokenAliases.size ()> 0) {
        retorno (Token) currentTokenAliases.pop ();
    }
}
}

Alias retorno próximo
como próximo token

```

```

Token nextToken = input.next();

if (nextToken == null) return null;

Aliases coleção =
(Coleção) ALIAS MAP.get (nextToken.termText ());

pushAliases (aliases);           ← Push aliases
                                na pilha
retorno nextToken;             ← Retorno
}                               próximo token

privada pushAliases void (Collection aliases) {

    if (aliases == null) return;           Alias lista de carga de
                                            arquivo de propriedades

    for (Iterator i = aliases.iterator (); i.hasNext ()) {
        String token = (String) i.next ();
        currentTokenAliases.push (novo Token (token, 0, token.length ()));
    }

}

```

Por exemplo, se o texto palavra-chave é "A máquina Zyon" e as seguintes propriedades arquivo é usado, o fluxo resultante token deverá conter os seguintes tokens: zyron, Xyron, dispositivo, e máquina:

```

zyron = Xyron
dispositivo da máquina =

```

#### Analisando as cores de impressão de arte

Inicialmente, a cor dominante de cada impressão foi a de ser escolhido manualmente pelo processo pessoal. No entanto, este plano foi falho em que a análise de cores por um ser humano é subjetivo e lento. Portanto, a equipe desenvolveu uma análise Michaels.com ferramenta para determinar as cores dominantes uma impressão automaticamente.

Para começar, uma paleta de cores finitos foi escolhida para combinar com cada impressão contra.

O tamanho da paleta foi mantido pequeno para evitar ambigüidade de cores semelhantes, mas ainda era suficientemente grande para acomodar as expectativas mais decoradores. Em última análise, palavras de 21 cores e três tons de cinza foram escolhidos (ver tabela 10.6).

Quadro 10.6 A paleta de cores Michaels.com para encontrar cópias da arte

# 000000	# CCCCCC	# FFFFFF
# 663300	# CC6600	# FFCC99

continua na página seguinte

Quadro 10.6 A paleta de cores Michaels.com para encontrar cópias da arte (Continuação)

# 006633	# 666600	# CCCC66
# CCCC00	# FFCC33	# FFFFCC
# 006699	# 99CCFF	# 99CCCC
# 330066	# 663399	# 6633CC
# 993333	# CC6666	# FF9999
# FF3333	# FF6600	# FF99CC

A ferramenta de análise de processos de imagens JPEG de cada impressão. Cada pixel da imagem é comparado com cada cor na paleta de cores em uma tentativa de encontrar a paleta cor que mais se aproxime da cor do pixel. Cada cor na paleta tem uma pontuação associada, que reflete o número de pixels na imagem que corresponde a essa cor.

Quando a correspondência de cores de um pixel para a paleta de cores, uma fórmula de distância de cor é aplicada. Considere os RGB (vermelho / verde / azul) componentes de uma cor a ser mapeados no espaço euclidiano. Encontrar a distância entre duas cores é simplesmente uma questão de determinar a distância entre dois pontos no espaço euclidiano com a fórmula mostrada na figura 10.6.7

Depois de cada pixel é avaliada contra a paleta de cores, as três cores com a maior pontuação são consideradas as cores dominantes para a cópia da arte. Mais-mais, se qualquer uma das contas de cores para menos de 25% dos pixels na impressão, então que a cor é considerado insignificante e é jogado fora.

Uma vez que as cores dominantes foram escolhidos, os seus triplos hexadecimal (tal como FFCC99) são armazenados no banco de dados relacional, juntamente com outros a impressão de

$$distance = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

Figura 10.6 Fórmula de distância de cor

<sup>7</sup> Na verdade, a fórmula empregada por Michaels.com é ligeiramente mais complicado do que isso. O ser humano olho é mais sensível a variações de algumas cores do que outros. Mudanças no componente verde são mais visível do que alterações no componente vermelho, que são mais visíveis do que as mudanças na componente azul. Portanto, a fórmula deve ser ajustado para dar conta do fator humano da cor. A fórmula atual usada pela Michaels.com é um derivado da fórmula explicada no <http://www.compuphase.com/cmetric.htm>.

da informação. A rotina de análise de cor é uma rotina de uma só vez aplicado quando um impressão é primeiro adicionada ao site e não é executado toda vez que uma impressão é indexados no Lucene.

#### Executando os indexadores

O índice de pesquisa é reconstruído a partir do zero uma vez por hora. A discussão de fundo desperta, cria um novo índice vazia, e então passa para adicionar dados de conteúdo para o índice. Isto é simplesmente uma questão de desenhar os dados de um item de conteúdo a partir da rela-  
banco de dados internacional, construindo um documento Lucene para conter esses dados, e depois  
adicioná-lo ao índice.

De modo que o mecanismo de busca permanece disponível durante a indexação, há dois índices: um índice ativo e um índice de trabalho. O índice de ativos está disponível para busca por clientes Michaels.com, enquanto o índice de trabalho é onde indexação ocorre. Uma vez que o indexador é completo, o directo-trabalho e ativa Ries são trocados de forma que o novo índice torna-se o índice de ativos e os velhos Índice de espera para ser reconstruída uma hora depois.

Para evitar vários arquivos de índice, Michaels.com recentemente começou a usar o novo formato composto índice disponível em Lucene 1.3.

#### 10.6.2 conteúdo Searching

Vários formulários HTML drive a busca de Michaels.com. No caso de uma simples palavras-chave de busca, o formulário contém um palavras-chave de campo. No caso de um Finder Art busca, o formulário contém um HTML <select> nomeado assunto, Um campo oculto (Preenchidos por meio de JavaScript) chamado cor, E um conjunto de botões de rádio chamado orientação.

Quando a pesquisa é apresentada, cada um desses campos é colocado em um java.util. Mapa (Onde o nome do parâmetro é a chave eo valor do parâmetro é o valor) e passado para o método construtor Lucene consulta mostrada na listagem 10.4.

#### Listagem 10.4 Construindo uma consulta Lucene partir de um mapa de campos

```
String private static final [] = IGNORE_WORDS
new String [] {"e", "ou"};
```

```
constructLuceneQuery String public static (campos Mapa) {
    StringBuffer queryBuffer = new StringBuffer ();

    for (Iterator iterator = chaves fields.keySet () () ; keys.hasNext ()) {
        String key = (String) keys.next (); Ciclo em cada
        Campo String = (String) fields.get (key); campo no Mapa
```

```

Nonalphanumeric Strip if (key.equals ("keywords")) {
    personagens de palavras-chavePalavras-
chave String =
    . removeNonAlphaNumericCharacters (campo) toLowerCase ();

StringTokenizer tokenizer = new StringTokenizer (palavras-chave);

while (tokenizer.hasMoreTokens ()) {Separar palavras-chave
    String nextToken = tokenizer.nextToken (); no espaço delimitador
    if (Arrays.binarySearch (IGNORE_WORDS, nextToken) > 0) {
        continuar; Se reservados
    } palavra, ignore
    if (! StringUtils.isEmpty (palavras-chave)) {
        queryBuffer.append ("+"). append (nextToken) append ("");
    } Adicionar palavra-chave
} para consulta

}
else {
    queryBuffer.append ("+"). append (chave). append (":");
    append (campo) append ("");
}
}
queryBuffer.toString return ();
}

```

Adicionar campo nonkeyword e valor para consulta

Quando se lida com palavras-chave, os cuidados devem ser tomados para garantir que nenhum caractere com significado especial para Lucene são passados para a consulta. Uma chamada para o `removeNonAlphaNumericCharacters ()` utilitário tira todos os caracteres que não são a Z, 0-9, ou espaços. O `palavras-chave` campo também é normalizada para minúsculas e despojado de qualquer palavra com um significado especial para Lucene (neste caso, e e ou).

Neste ponto, o `palavras-chave` string está limpo e pronto para ser adicionado à busca consulta. Se quiséssemos que a consulta seja uma consulta inclusiva (incluindo todos os documentos

combinando qualquer uma das palavras-chave), nós poderíamos apenas acrescentar a `palavras-chave` string para o consulta e ser feito. Em vez disso, cada palavra na string é prefixado com um plus sinal (+), Indicando que documentos correspondentes deve conter o word.8

Por exemplo, dada uma frase de pesquisa de "Mãe e criança", o que resulta Lucene consulta seria "Mãe-filho + +".

No caso dos campos de busca nonkeywords, nós simplesmente adicionar o nome e valor do campo na consulta, separados por dois pontos (:). Por exemplo, teve a

---

<sup>8</sup> Nota dos autores: Há interações bastante estranhas entre analisadores e `QueryParser` para nos acrescentar um aviso aqui. Construção de uma expressão de consulta no código para ser analisado por `QueryParser` pode ser peculiar. Um alternativa é construir um `BooleanQuery` com aninhadas `TermQuery`s diretamente.

cliente usou Art Finder para localizar uma cópia da arte horizontal em qualquer assunto com o escuro

marrom como cor dominante, a consulta seria "Orientação: horizontal color: 663300".

Com a consulta construída, agora estamos prontos para realizar a pesquisa.

### Submeter a consulta

O `findDocuments ()` método (listagem 10.5) é responsável pela consulta de um dado Lucene índice e retornar uma lista de documentos que correspondem a essa consulta.

#### 10.5 A listagem `findDocuments ()` método retorna uma lista de documentos correspondentes

```
privada findDocuments List (String queryString,
    String indexDirectory) {  
  
    IndexSearcher searcher = null;
    try {
        searcher = new IndexSearcher (indexDirectory);  
  
        Query = QueryParser.parse (
            queryString, "keywords", novo SearchAnalyzer ());
    }  
  
    Hits hits = searcher.search (query);
    DocumentList lista = new ArrayList ();
    for (int i = 0; i < hits.length () 9; i + +) {
        documentList.add (
            nova BaseDocument (hits.doc (i), hits.score (i)));
    }
    retorno DocumentList;  
  
}  
catch (Exception e) {
    throw new SystemException ("Um erro ocorreu pesquisa");
}
finally {
    LuceneUtils.close (pesquisador);
}  
}
```

Open IndexSearcher  
no diretório especificado

Analisar consulta

Fazer pesquisa

<sup>9</sup> Nota dos autores: Esteja ciente do número potencial de hits, o tamanho de seus documentos, e escalabilidade do necessidades do seu aplicativo quando você escolhe iterar sobre todos os hits, especialmente se você coletá-los usando `hits.doc (i)` como este. Como observado neste estudo de caso, o desempenho nesse cenário tem sido mais do que aceitável, mas os índices muito maiores e consultas arbitrárias mudar a paisagem de forma dramática.

O `BaseDocument` classe (listagem 10.6) é simplesmente um meio de amarrar uma Lucene Documento ao seu índice de relevância. Como iludido pelas `getId ()` método, a única coisa que nós nos importamos sobre dentro de um documento retornado é o seu ID. Nós vamos usar esse valor para procurar o peça completa dos dados do banco de dados relacional.

#### Listagem 10.6 BaseDocument associa um Documento e sua pontuação

```
BaseDocument public class {
    Documento protegido documento final;
    protegidos pontuação float final;

    BaseDocument (documento Documento, pontuação float) {
        = documento this.document;
        this.score = pontuação;
    }

    getId public int () {
        retorno Integer.parseInt (documento.get ("id"));
    }

    String getFieldValue (String fieldName) {
        retorno documento.get (fieldName);
    }
}
```

Com a lista de `BaseDocuments` retornou de `findDocuments ()`, Nós estamos prontos para pare para baixo os resultados em valor de uma página de dados:

```
DocumentList lista = findDocuments (consulta, indexPath);
SubList lista = documentList.subList (início,
    Math.min (start + contar, documentList.size ()));
```

O `começar` variável indica o primeiro documento para a página atual, enquanto o `contar` variável indica quantos itens estão na página atual.

Usando o ID de cada documento em `subList` como uma chave primária, o último passo é recuperar dados adicionais sobre cada documento do banco de dados relacional.

### 10.6.3 estatísticas Search

Na época isso foi escrito (março 2004), Michaels.com ostentava 23.090 arte imprime, 3.327 projetos, 385 na loja promoções de produtos, e 191 crafting artigos, tudo pesquisável através Lucene.

Durante o período de compras de 2003 feriado, normalmente um tempo de pico de tráfego para

Michaels.com, o mecanismo de busca foi contratado cerca de 60.000 vezes por dia. Sem falhar, Lucene retornou resultados em tempo subsecond para cada solicitação.

#### 10.6.4 Resumo

Michaels.com teve enorme sucesso em empregar Lucene para conduzir sua pesquisa facilidade, permitindo que os clientes para encontrar a arte e informação de artesanato e produtos que eles estão procurando. Usando sua API simples e intuitiva, fomos capazes de integrar Lucene em codebase do nosso site rapidamente. Ao contrário de seus antecessores, Lucene tem provou ser estável, robusto e muito rápido. Além disso, corre-se praticamente de mãos livre, não necessitando de qualquer intervenção desenvolvedor em mais de um ano e meio.

### 10,7 Eu amo Lucene: TheServerSide

---

Contribuição de Dion Almaer

"TheServerSide.com é uma comunidade online para os arquitetos corporativos e Java desenvolvedores, fornecendo notícias diárias, entrevistas falar tech com figuras-chave da indústria, padrões de projeto, fóruns de discussão, sátira, tutoriais, e muito mais. "

-Http: / www.theserverside.com /

TheServerSide historicamente tinha um motor de busca pobre. Graças ao Jakarta Lucene, poderíamos resolver o problema com uma solução de alta qualidade de fonte aberta. Neste caso estudo discute como TheServerSide implementado Lucene como sua base tecnologia de busca.

#### Capacidade de pesquisa 10.7.1 Building melhor

Há uma série de áreas em TheServerSide que gostaríamos de mudar. Confiar em nós. Desde que me juntei TheServerSide eu encolhi no nosso motor de busca implementação. Ele não fez um bom trabalho, e isso significava que nossos usuários não conseguiam chegar até informação que eles queriam. Análise de interface do usuário tem demonstrado que a funcionalidade de pesquisa é muito importante sobre a web (veja <http://www.useit.com/alertbox/20010513.html>), então nós realmente tivemos que limpar nosso ato aqui. Este estudo de caso discute como TheServer- Lado construída uma infra-estrutura que nos permite indexar e pesquisar o nosso conteúdo diferente usando Lucene. Vamos conversar sobre nossa infra-estrutura de alto nível, como índice de nós e pesquisa, bem como a forma como somos facilmente capaz de ajustar a configuração. Então, nós queríamos um bom motor de busca, mas quais são as opções? Estavamos usando ht: / Dig / e tê-lo rastrear nosso site, a criação do índice, uma vez que foi along.<sup>10</sup> Este

---

<sup>10</sup> Para saber mais sobre ht: / / Dig, visite <http://www.htdig.org/>.

processo não foi pegando todo o conteúdo e não nos dão uma API limpa agradável para nos para ajustar os resultados de busca. Ele fez um coisa assim, e que estava procurando através de nossas notícias. Este foi um efeito colateral de ter notícias na página inicial, que ajuda os rankings (mais cliques ht: / / Dig necessários para navegar a partir de casa página, menor o ranking).

Embora ht: / / Dig não ia um grande trabalho, nós poderíamos ter tentado ajudá-lo em seu caminho. Por exemplo, poderíamos ter criado um arquivo HTML especial que ligava a diversas áreas do site e usou isso como a página de raiz para que a engatinhar. Talvez nós poderia ter colocado um filtro de servlet que verificado para o ht: agente de usuário / Dig / e voltou o conteúdo de uma maneira diferente (limpeza do HTML e tal).

Olhamos para usar o Google para gerenciar nossa busca por nós. Quero dizer, eles são muito bons em busca, não são?! Embora eu tenho certeza que poderíamos ter tido uma boa pesquisa de usá-los, nós funcionamos em um par de questões:

- Não foi fácil para nós (a empresa de pequeno porte) para obter muita informação a partir de -los.
- Para o tipo de pesquisa que precisávamos, ele estava olhando muito caro.
- Temos ainda as questões de uma infra-estrutura baseada em rastreador.

Enquanto estávamos procurando no Google, eu também estava olhando para Lucene. Lucene tem

sempre me interessou, porque não é um projeto open-source típico. Na minha experiência, a maioria dos projetos open-source são estruturas que evoluíram. Levar alguma coisa como Struts. Antes de Struts, muitas pessoas estavam rolando suas próprias camadas MVC

em cima de Servlets / JSPs. Fazia sentido para não ter de reinventar a roda, de modo Struts veio ao redor.

Lucene é um animal diferente. Ele contém alguns realmente complicado de baixo nível trabalho, não apenas um quadro bem concebido. Fiquei realmente impressionado que alguma coisa desta qualidade foi só colocar lá fora!

No começo eu estava um pouco decepcionado com o Lucene, porque eu realmente não understand o que era O. Imediatamente eu estava olhando para a funcionalidade do rastreador que me permitisse criar um índice apenas como ht: / / Dig estava fazendo. Na época, LARM estava no Sandbox Lucene (e eu tenho desde que ouvi falar de vários outros subprojetos), mas eu achei estranho que este não seria construído na distribuição principal. Ele Levei um dia para perceber que Lucene não é um produto que você acabou de executar. É um-top

notch busca API que você pode usar para conectar ao seu sistema. Sim, você pode ter que escrever um código, mas você também terá grande poder e flexibilidade.

### 10.7.2 infra-estrutura de alto nível

Quando você olha para construir a sua solução de busca, você encontrará muitas vezes que o processo é dividido em duas tarefas principais: construção um índice, e pesquisar esse índice. Este é definidamente o caso do Lucene (e o único momento em que este não é o caso é se o seu pesquisa vai diretamente para o banco de dados).

Queríamos manter a interface de pesquisa bastante simples, de modo que o código que interatua com o sistema vê duas interfaces principais: `IndexBuilder` E `IndexSearch`.

#### IndexBuilder

Qualquer processo que precisa para construir um índice passa por o `IndexBuilder` (Figura 10.7). Este é um simples interface que fornece dois pontos de entrada para a indexação processo. Para fazer uma compilação incremental e controlar a freqüência para otimizar o índice Lucene como você adicionar registros, passe individual definições de configuração para a classe. Para controlar as configurações de um arquivo de configuração externa, use um nome do plano. Você também verá um `main (...)` método. Nós criamos este para permitir a um programa de linha de comando para chutar fora de um processo de construção.

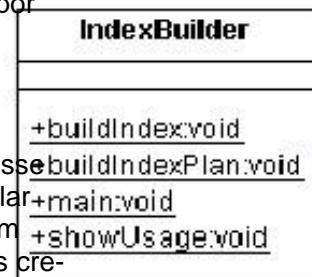


Figura 10.7  
IndexBuilder

#### IndexSources

O `IndexBuilder` abstrai os detalhes de Lucene, e os `IndexSources` que são usado para criar o índice em si. Como veremos na próxima seção, TheServerSide tem vários conteúdos que queria ser capaz de indexar, de forma um design simples é utilizado onde podemos plug 'n play fontes novo índice.

#### IndexSearch

A interface de pesquisa também é mantida muito simples (ver figura 10,8). A pesquisa é feita através de

```

IndexSearch11.search inputQuery String (, int
    resultsStart,
    int resultsCount);
  
```

Por exemplo, nós olhamos para os termos EJB e WebLogic, retornando até os 10 primeiros resultados:



Figura 10.8     IndexSearch

```

IndexSearch.search ("EJB E WebLogic", 0, 10);
  
```

<sup>11</sup> Nota dos autores: Tenha cuidado para não confundir TheServerSide de `IndexSearch` classe com Lucene `IndexSearcher` classe.

A consulta é construída através do Lucene `QueryParser` (Na verdade, uma subclasse que criamos, que você vai ver em detalhe mais tarde). Isto permite que nossos usuários a entrada típica Google-consultas esque. Mais uma vez, um `main ()` método existe para permitir a linha de comando em busca de índices.

### 10.7.3 Construção do índice

Vimos que a interface externa para a construção de nosso índice de pesquisa é a classe `IndexBuilder`. Agora vamos discutir o processo de construção do índice e do design escolhas que fizemos.

**Quais campos devem fazer o nosso índice?**

Queríamos criar um conjunto bastante genérica de campos que o nosso índice de conteria. Acabamos com os campos mostrados na tabela 10.7.

Tabela 10.7 TheServerSide estrutura de campo de índice

Campo	Tipo Lucene	Descrição
título	Field.Text	Um título curto do conteúdo.
sumário	Field.Text	Um parágrafo sumário introduzindo o conteúdo.
fullcontents	Field.UnStored	Todo o conteúdo para o índice, mas não loja.
proprietário	Field.Keyword	O proprietário do conteúdo (que escreveu o post? Que era o autor do artigo?).
categoria	Field.Keyword	O tipo de este conteúdo (é uma notícia? Um artigo?).
caminho	Field.Keyword	O único caminho que aponta para este recurso.
ModifiedDate	Field.Keyword	A data de modificação no formato Lucene. Usado para exibir o data exata do conteúdo para o usuário.
CreatedDate	Field.Keyword	A data de criação no formato Lucene. Usado para exibir o data exata do conteúdo para o usuário.
modifieddate_range	Field.Keyword	Data como um <code>Corda</code> com o formato YYYYMMDD. Usado para intervalo de datas-queries.
createddate_range	Field.Keyword	Data como um <code>Corda</code> com o formato YYYYMMDD. Usado para intervalo de datas-queries.

Nós criamos uma representação Java simples destes dados, `SearchContentHolder`, Que nossa API usa para passar esta informação ao redor. Ele contém o modificado e `cre-`datas ated como `java.util.Date`, E todo o conteúdo é armazenado como um `StringBuffer`

ao invés de um `Corda`. Este foi reformulado em nosso projeto, porque descobrimos que alguns `IndexSources` continham uma grande quantidade de dados, e nós não deseja adicionar ao `Cordas`.

Que tipos de indexação?

Como o conteúdo TSS que queríamos índice é bastante grande e um monte de não mudança, que queríamos ter o conceito de indexação incremental, bem como uma completa indexação a partir do zero. Para cuidar disso, temos um `incrementalDays` variável que está configurada para o processo de indexação. Se este valor for definido para 0 ou menos, então fazer um

índice cheio. Se este não for o caso, então o conteúdo que é mais recente (criada / modificada) que hoje - `incrementalDays` devem ser indexados. Neste caso, em vez de criar um novo índice, nós simplesmente excluir o registro (se ele já existir) e inserir mais recente dados nela.

Como você excluir um registro no Lucene novamente? Precisamos da `org.apache.lucene.index.IndexReader`. O trecho que faz o trabalho é mostrado na listagem 10.7.

Listagem de 10.7 Snippet IndexHolder que exclui a entrada do índice, se for  
já lá

```
IndexReader reader = null;
try {
    this.close(); // fecha o escritor índice subjacente

    reader = IndexReader.open (SearchConfig.getIndexLocation ());
    Termo termo Term = new ("caminho", theHolder.getPath ());
    reader.delete (prazo);
} Catch (IOException e) {
    ... lidar com exceção ...
Finally {}
    try {reader.Close ();} catch (IOException e) {/* suck it up */}
}

this.open (); // reabrir o escritor índice
```

Como você pode ver, primeiro fechar a `IndexWriter`, E então abrir o índice através do `IndexReader`. O campo caminho é o ID que corresponde a esta "a ser indexado" entrada. Se ele existir no índice, ele será apagado, e pouco depois vamos voltar a adicionar a informação do índice novo.

Qual o índice?

TheServerSide como tem crescido ao longo do tempo, temos o efeito colateral de possuir conteúdo que vive em fontes diferentes. Nossas discussões threaded estão no banco de dados,

mas nossos artigos vivemos em um sistema de arquivos. O Talks Tech Hard Core também sentar-se no arquivo sistema, mas de uma maneira diferente do que nossos artigos.

Queríamos ser capaz de conectar diferentes fontes para o índice, por isso criamos um simples `IndexSource` interface e um correspondente `Fábrica` classe que retorna todas as fontes de índice a ser indexado. O código a seguir mostra a simples `IndexSource` interface:

```
public interface indexSource {  
    pública addDocuments void (titular IndexHolder);  
}
```

Há apenas um método, `addDocuments()`, Que um `IndexSource` tem a implement. O `IndexBuilder` é acusado de chamar esse método em cada índice Fonte e passando em um `IndexHolder`. A responsabilidade da `IndexHolder` está na envolvendo em torno do índice de pesquisa Lucene específicas (via `Lucene.org.apache.lucene.index.IndexWriter`). O `IndexSource` é responsável por tomar este titular e adicionar registros a ele no processo de indexação.

Vejamos um exemplo de como um `IndexSource` faz isso olhando para o `ThreadIndexSource`.

## ThreadIndexSource

Esta fonte de índice passa pelo banco de dados de TSS e os vários índices tópicos de todos os nossos forums.<sup>12</sup> Se estamos fazendo uma compilação incremental, então o resultados são simplesmente limitada pela consulta SQL que emitimos para obter o conteúdo.

Quando chegamos de volta os dados do banco de dados, precisamos transformar isso em um instância de `SearchContentHolder`. Se não temos um resumo, então nós simplesmente cortar o corpo para um comprimento de resumo regidos pela configuração.

O principal campo de pesquisa é que nós `fullcontents`. Para se certificar de que um usuário do sistema encontra o que quer, nós fazemos neste campo não somente o corpo de uma thread mes-sábio, mas sim uma concatenação do título da mensagem, o proprietário do mensagem, e, finalmente, o conteúdo da mensagem em si. Você poderia tentar usar boolean consultas para se certificar de que a busca encontra um bom jogo, mas achamos muito mais simples de colocar em uma concatenação atrevida!<sup>13</sup>

Assim, este deve mostrar como é simples criar um `IndexSource`. Criamos fontes para artigos e palestras de tecnologia (e de fato um par de versões para lidar com uma

<sup>12</sup> Nota dos autores: Para esclarecer, a palavra fio aqui se refere a uma série de postagens do fórum com um tema comum.

<sup>13</sup> Nota dos autores: Para saber mais sobre consultando vários campos e esta técnica de concatenação, consulte a seção 5.3.

upgrade em instalações de gestão de conteúdo). Se alguém quer que a gente busca um novo fonte, criamos um novo adaptador, e estamos no negócio.

### Como ajustar o ranking de registros

Quando a mão do `IndexHolder` `umSearchContentHolder`, Ele faz o trabalho de adding-lo ao índice Lucene. Esta é uma tarefa bastante trivial de tomar os valores de o objeto e adicioná-los a um documento Lucene:

```
doc.add (Field.UnStored ("fullcontents", theHolder.getFullContents ()));
doc.add (Field.Keyword ("dono", theHolder.getOwner ()));
```

Há um pedaço de lógica que vai além munging os dados para um Lucene-friendly maneira. É nesta classe que calcula qualquer que impulsiona deseja colocar em campos ou do próprio documento. Acontece que vamos acabar com os impulsionadores mostrado na tabela 10.8.

Tabela 10.8 TheServerSide campo aumenta

Impulsionar	Descrição
Título	Um título deve ter mais peso do que algo no corpo de uma mensagem, assim colidir acima este reforço de campo.
Sumário	Um resumo também deve ter mais peso do que o corpo da mensagem (embora não tanto como um título), então fazer o mesmo aqui.
Categoria	Algumas categorias nascem mais importantes que outros. Por exemplo, nós o peso de primeira página tópicos e artigos mais elevados do que os fóruns de discussão.
Impulsiona a data	Novas informações é melhor, não é? Nós impulso um documento se ele é novo, eo impulso diminui à medida que o tempo passa.

O aumentar momento tem sido muito importante para nós. Temos dados que remonta a uma longa tempo e parecia estar retornando relatórios antigos com muita freqüência. O booster data-base truque ficou em torno deste, permitindo o mais novo conteúdo a borbulhar.

O resultado final é que agora temos um belo design simples que nos permite adicionar novas fontes para o nosso índice com o tempo de desenvolvimento mínimo!

#### 10.7.4 Pesquisando o índice

Agora, temos um índice. Ele é construído a partir de diversas fontes de informação que nós tem e está apenas esperando alguém para procurá-la.

Lucene fez esta muito simples para nós chicote. As entranhas de busca são escondido atrás da `IndexSearch` classe, conforme mencionado na visão geral de alto nível. O trabalho é tão simples que eu posso até colá-lo aqui:

```
pesquisa pública SearchResults estática (inputQuery String,
                                         int resultsStart,
                                         int resultsCount) throws SearchException {
try {
    Pesquisa Pesquisa = new
        IndexSearcher (SearchConfig.getIndexLocation ());
    String [] fields = {"title", "fullcontents"};
    ...
    Hits hits = searcher.search (
        CustomQueryParser.parse inputQuery (, campos,
                                         nova StandardAnalyzer ()));
    ...
    SearchResults sr = new SearchResults (hits, resultsStart,
                                         resultsCount);
    searcher.close ();
    retorno sr;
} {} Catch (...){
    throw new SearchException (e);
}
}
```

Este método simplesmente envolve o Lucene `IndexSearcher` e por sua vez envelopes os resultados como nossos próprios `SearchResults`.

O único item ligeiramente diferente a notar é que nós criamos as próprias simples Consulta-Analisador variante. O CustomQueryParser estende Lucene e é construído para permitir que um consulta de pesquisa padrão para pesquisar tanto os `título` e `fullcontents` campos. Ele também dispõe o útil, ainda caro, consultas curinga e fuzzy. A última coisa que queremos é para alguém fazer um monte de consultas, tais como '`A *`', Fazendo um monte de trabalho na Motor Lucene. Nosso analisador consulta personalizada é mostrada na listagem de 10.8.14

Analisador de listagem 10.8 TheServerSide de consulta

```
CustomQueryParser public class QueryParser
{
    / **
     * Método de análise estática que irá consultar o título eo
     * Os campos fullcontents através de um BooleanQuery
     * /
    analisar consulta public static (String query, String [] campos,
                                    Analisador analisador) throws ParseException {
        BooleanQuery bQuery BooleanQuery = new ();
        for (int i = 0; i < campos.length; i + +) {
            Parser QueryParser = CustomQueryParser novos campos ([i],
                        analisador);
```

<sup>14</sup> Autores note: Consulte a seção 6.3.2 para um analisador de consulta quase idêntico costume e ainda mais discussão sobre o uso de subclasse QueryParser.

```
Query q = parser.parse (query);
bQuery.add (q, false, false);
}

retorno bQuery;
}

CustomQueryParser pública (campo String, analisador Analyzer) {
    super (campo, analisador);
}

getWildcardQuery Query final protegidas (String campo, String termo)
    throws ParseException {
    throw new ParseException ("Query curinga não são permitidos.");
}

getFuzzyQuery Query final protegidas (String campo, String termo)
    throws ParseException {
    throw new ParseException ("Fuzzy Query não é permitido.");
}

}
```

Combine consultas,  
nem exigir, nem  
proibindo jogos

Isso é tudo, pessoal. Como você pode ver, é bastante trivial para pegar a bola rolando no lado busca da equação.

### 10.7.5 Configuração: um lugar para todos governar

Houve ajustes, tanto no processo de indexação e processo de pesquisa que clamavam de abstração. Onde devemos colocar o local do índice, o gatogoria listas, e os valores impulso, e registrar o índice de fontes? Nós não queremos ter isso no código, e desde que a configuração foi hierárquico, recorreu-se usando XML.

Agora, eu não sei sobre você, mas eu não sou um grande fã das APIs de baixo nível como o SAX e DOM (ou até mesmo JDOM, dom4j, e assim por diante). Em casos como este, nós não se preocupam com a análise a este nível. Eu realmente só quero minha configuração informa-

ção, e seria perfeito para ter esta informação dada a mim como um objeto modelo. Este é o lugar onde ferramentas como Castor-XML, JiBX, JAXB, e Jakarta Commons Digester vêm dentro

Optou-se pela Digester Jakarta neste caso. Nós criamos o modelo de objeto para manter a configuração que precisávamos, todos atrás da `SearchConfig` fachada. Esta fachada tem um `Singleton` objeto que segurava a configuração, como mostrado na listagem 10.9.

## Indexação de listagem 10,9 Resumos e configuração pesquisa

```

    /**
     * Enrole uma instância Singleton que detém uma ConfigHolder
     * @ Return
     */
    getConfig ConfigHolder public synchronized estático () {
        if (ourConfig == null) {
            try {
                String configname = "/ search-config.xml";
                Entrada de arquivo = new File (PortalConfig.getSearchConfig () +
                    configname);
                Regras de arquivo = new File (PortalConfig.getSearchConfig () +
                    "/ Digestor rules.xml");

                Digestor digestor = DigesterLoader.createDigester (
                    rules.toURL ());

                ourConfig = (ConfigHolder) digestor.parse (entrada);
            } Catch (...) {
                / / ...
            }
        }
        retorno ourConfig;
    }

```

Este método diz o conto de digestor. Leva o arquivo de configuração XML (pesquisa config.xml) e as regras para a construção do modelo de objeto (digestor rules.xml) e joga-los em uma panela junto, e você acabar com o modelo de objeto (ourConfig).

## Arquivo de configuração XML

O arquivo de configuração aciona o processo de indexação e ajuda o sistema de busca. Para registrar um fonte de índice específico, basta adicionar uma entrada sob o <index-source> elemento. Listagem 10,10 mostra um exemplo de nossa configuração.

## Listagem 10,10 Exemplo de arquivo de pesquisa-config.xml

```

<search-config>
    <! - O caminho para onde o índice de pesquisa é mantida ->
    <Índice de localização-windows = "/ temp / tss-SearchIndex"
      unix = "/ tss / SearchIndex" />

    <! - Ano de início do conteúdo que é indexado ->
    <beginning-year> 2000 </ a partir do ano->

    <! - Informações sobre resultados de pesquisa ->
    <search-results results-per-page="10" />

```

```
<! - Configuração Plano Index ->
<index-plan name="production-build">
    <optimize-frequency> 400 </ otimizar frequência->
</ Index plano->

<index-plan name="test-build">
    <optimize-frequency> 0 </ otimizar frequência->
</ Index plano->

<index-plan name="daily-incremental">
    <incremental-build> 1 </ incremental-build>
    <optimize-frequency> 0 </ otimizar frequência->
</ Index plano->

<! - Mapping Config Categoria ->
<categories>
    <category number="1" name="news" boost="1.3" />
    <category number="2" name="discussions" boost="0.6" />
    <category number="3" name="patterns" boost="1.1" />
    <category number="4" name="reviews" boost="1.08" />
    <category number="5" name="articles" boost="1.1" />
    <category number="6" name="talks" boost="1.0" />
</ Categories>

<! - Configuração Valor Boost ->
<Aumentar a quantidade de data-base = "1.0" data-boost-per-count =
"0,02"
    title = "2.0" summary = "1.4" />

<! - Lista de todas as fontes Index ->
<index-sources>
    <Thread-index-source sumário comprimento = "300"
        class-name = "com.portal.util.search.ThreadIndexSource">
        <excluded-forums>
            <forum> X </ forum>
        </ Excluidos-forums>
    </ Thread-index-source>

    <Article-index-source
        class-name = "com.portal.util.search.ArticleIndexSource"
        = diretório "web / tssdotcom / artigos"
        categoria-name = "artigos"
        caminho-prefix = "/ artigos / article.jsp? l ="
        padrão-criação-date = "hoje"
        padrão-modificado-date = "hoje" />

</ Index-fontes>
</ Pesquisa-config>
```

Se você examinar o arquivo, você verá que agora nós podemos ajustar a maneira que o índice é construída através de elementos como `<boost>` , O `<categories>` , Informações e em

<index-sources>. Essa flexibilidade permitiu-nos jogar com vários ajustes de boost até que parecia certo.

### Digestor arquivo Regras

Como é que o Digestor tomar a pesquisa-config.xml e know-how para construir o modelo de objeto para nós? Esta mágica é feito com um arquivo Regras Digestor. Aqui nós dizemos

O digestor o que fazer quando se depara com uma determinada tag.

Normalmente você vai dizer o motor para fazer algo como isto:

- 1 Criar um novo objeto IndexPlan quando você encontrar uma <index-plan>.
- 2 Tomar os valores de atributos e métodos definidos na chamada correspondente objeto (category.setNumber (...), category.setName (...)) E assim por diante).

Listagem 10,11 mostra um trecho das regras que empregamos.

Listagem 10,11 Um trecho do digestor rules.xml-

```

<? Xml version = "1.0"?>

<digestor-rules>
    <-! Object ConfigHolder Top Level ->
    <pattern value="search-config">
        <Objeto criar regras
            classname = "com.portal.util.search.config.ConfigHolder" />
            <set-properties-rule/>
        <Padrão />

        <! - Resultados da Pesquisa ->
        <pattern value="search-config/search-results"> <pattern
            <set-properties-rule>
                <Alias attr-name = "resultados por página"
                    prop-name = "resultsPerPage" />
                </ Set-propriedades da regra->
            <Padrão />

            <! - Plano Index ->
            <pattern value="search-config/index-plan">
                <Objeto criar regras
                    classname = "com.portal.util.search.config.IndexPlan" />
                    <Padrão de feijão-propriedade-setter regra = "incremental-build"
                        propname = "incrementalBuild" />
                    <Feijão-propriedade-setter regra-padrão = "otimizar-freqüência"
                        propname = "optimizeFrequency" />
                    <set-properties-rule/>
                    <set-next-rule methodname="addIndexPlan" />
                <Padrão />

                ... regras mais aqui ...
            </ Digestor as regras>

```

Todas as regras para o digestor estão fora do escopo deste estudo de caso, mas você pode provavelmente achar muito com este trecho. Para mais informações, visite <http://jakarta.apache.org/commons/digester.15>

Então, graças a uma ferramenta open-source, fomos capazes de criar um bem simples ainda poderoso conjunto de regras de configuração para as nossas necessidades de pesquisa específico. Nós não

tem que usar uma rota de configuração XML, mas permite-nos ser flexíveis. Se fôssemos realmente gente boa, teríamos o sistema reformulado para permitir-programa configuração automática. Para fazer isso bem seria bastante trivial. Teríamos uma interface de configuração e uso de injeção de dependência (IoC) para permitir que o código instalação de qualquer aplicação (sendo um deles o construtor de arquivo XML, o outro vem de codificação manual).

#### 10.7.6 Web tier: TheSeeeeeeeeeeeeerverSide?

Neste ponto, temos uma interface agradável, limpo para a construção de um índice e procurar em um. Pois precisamos que os usuários pesquisem o conteúdo através de uma interface web, o último item

na lista de desenvolvimento foi a de criar o gancho camada web para a interface de busca.

TheServerSide infra-estrutura de portal utiliza uma home-grown MVC web tier. É casa cresceu puramente porque foi desenvolvido antes de os gostos de Struts, WebWork, ou Tapestry. O nosso sistema tem a noção de ações (Ou, como chamamos, montadores), de modo a criar a cola web que tínhamos de

- Criar uma ação web: SearchAssembler.java
- Criar uma visão web: A página de pesquisa e os resultados

##### SearchAssembler ação web

A ação web tier é responsável por tomar a entrada do usuário, passando até `IndexSearch.search (...)` E embalagem dos resultados em um formato pronto para a exibição. Não há nada interessante neste código. Tomamos a busca entrada da consulta para o usuário e criar a consulta Lucene, pronto para a busca infra-estrutura. O que quero dizer com "construir a consulta"? Basta colocar, nós adicionamos todos

as informações de consulta feita pelo usuário em uma string de consulta Lucene.

Por exemplo, se o usuário digitou Lucene na caixa de pesquisa, selecionados de uma data "Depois de 01 de janeiro de 2003", e estreitou as categorias de busca para "notícias", gostaríamos de end edificação

```
E Lucene categoria: Noticias e modifieddate_range: [20040101 TO 20100101]
```

Então o nosso código contém pequenos trechos, como

---

<sup>15</sup> Nota dos autores: Digestor também é utilizado para indexação de documentos XML na seção 7.2.

```

if (dateRangeType.equals ("antes")) {
    querySB.append (
        "E modifieddate_range: [19900101 TO" + + daterange "]");
} Else if (dateRangeType.equals ("depois")) {
    querySB.append (
        "E modifieddate_range: [" + + daterange "TO 2010010116]");
}

```

### Pesquisar vista

A tecnologia de exibição que usamos é JSP (mais uma vez, por razões de legado). Nós usamos nossos

MVC para se certificar de que o código Java é mantido fora do JSPs si. Então, o que ver neste código é basicamente apenas HTML com um par de tags JSP aqui e ali.

A uma parte da lógica real é quando há vários resultados (ver figura 10.9). Aqui nós temos que fazer alguma matemática para mostrar as páginas de resultado, que página você está, e assim por diante. Isso deve parecer familiar para a paginação no Google e similares. O única diferença é que nós sempre mostrar a primeira página, porque descobrimos que na maioria das vezes, página 1 é realmente o que você quer. Este é o lugar onde poderíamos ter Google realmente copiado e colocado TheSeeeeeeeeerverside ao longo das páginas.

A camada web é limpo e mantido o mais fino possível. Temos de aproveitar o trabalho feito no `IndexBuild` e `IndexSearch` interfaces de alto nível para Lucene.

### 10.7.7 Resumo

Você já viu todas as partes e peças de TheServerSide subsistema de busca. Nós o poder de alavancagem Lucene, mas expor uma visão abstrata de busca. Se tivéssemos para apoiar um outro sistema de busca, então poderíamos plug que por trás dos bastidores, e os usuários dos pacotes de busca não seria afetada.

Dito isto, não vemos qualquer razão para afastar-se Lucene. Tem foi um prazer trabalhar com e é uma das melhores peças de software de código aberto que eu pessoalmente já trabalhei.

TheServerSide pesquisa utilizado ser um elo fraco no site. Agora é uma potência. Estou constantemente a usá-lo como editor, e agora eu conseguir encontrar exatamente o que eu quero.

Indexação nossos dados é tão rápido que nós nem sequer precisa executar o incremental construir plano que desenvolvemos. Em um ponto que teve um engano `Índice Writer.optimize ()` chamada cada vez que nós adicionamos um documento. Quando relaxado que para executar com menos freqüência, que derrubou o momento da indexação a uma questão de segundos. Ele costumava levar muito mais tempo, mesmo enquanto 45 minutes.<sup>16</sup>

---

<sup>16</sup> Nota dos autores: Oh grande, então temos um problema Y2010 em TSS. Dion provavelmente pensa que não vai trabalhar lá por então, e alguém vai ter o prazer de rastrear porque as pesquisas não funcionam em 02 de janeiro de 2010! O

Então, para recapitular: Ganhamos velocidade, relevância e poder com esta abordagem. Podemos ajustar a nossa forma de indexar e pesquisar o nosso conteúdo com pouco esforço. Obrigado assim muito para toda a equipe Lucene.

## 10.8 Conclusão

É nós, Otis e Erik, de volta novamente. Nós, pessoalmente, gostava de ler esses casos estudos. As técnicas, truques e experiências fornecidos por estes estudos de caso ter de volta tidos em conta nosso próprio conhecimento e, implicitamente, aparecem ao longo deste livro. Saímos, em sua maior parte, as contribuições do estudo de um caso original intacto como eles foram fornecidos para nós. Esta secção dá-nos uma oportunidade para adicionar a nossa perspectiva.

Nutch, co-desenvolvido pela própria Lucene do corte Doug criador, é um fenomenal arquitetura projetada para a escalabilidade de farm de servidores de grande porte. Lucene se beneficiou

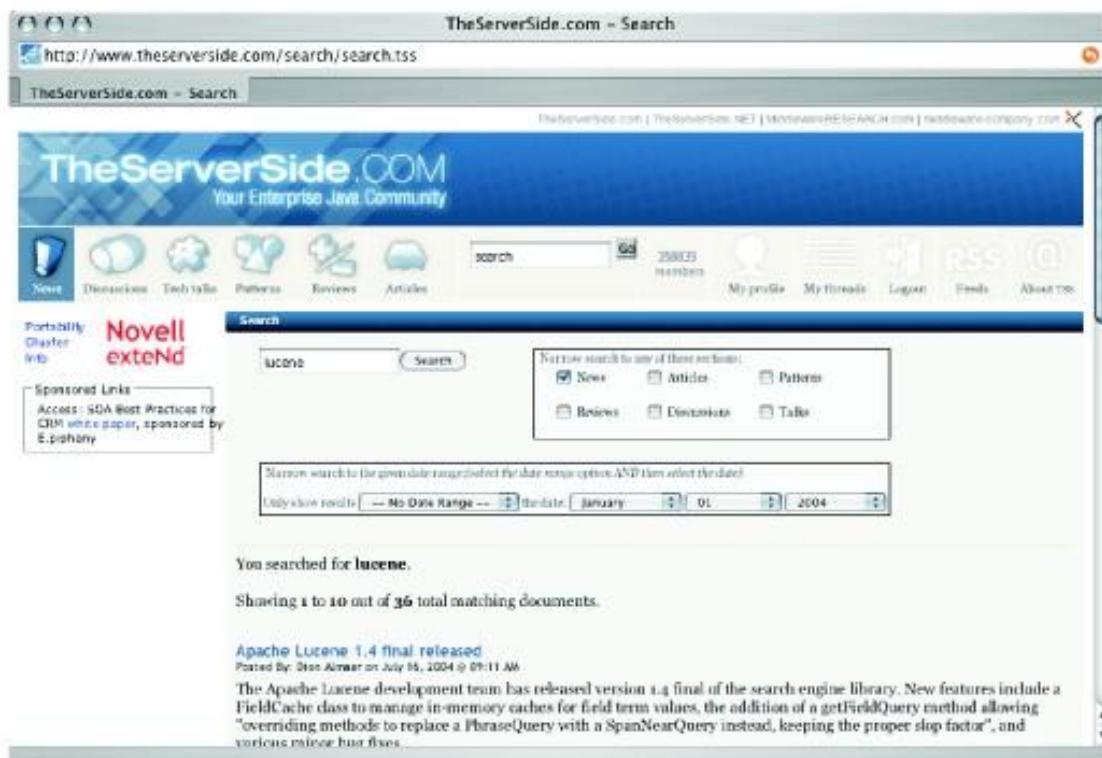


Figura 10.9 TheSeeeeeeeeeverSide

<sup>17</sup> Nota dos autores: otimização Index é coberto na seção 2.8.

dos esforços de Doug Nutch. O analisador Nutch é uma alternativa inteligente para evitar perda de precisão devido à palavra-stop remoção, mas velocidades de procura manter maximizada.

A busca jGuru site fornece resultados de alta qualidade procura de termos Java. Próprias vidas no FAQ do Lucene jGuru. Dar ao site uma tentativa próxima vez que você tem um em Java questão relacionada. É muitas vezes melhor do que as consultas Google por causa de seu domínio natureza específica.

SearchBlox dá algo Lucene lhe falta: a interface do usuário e manageabilidade. Lucene em si é uma API de baixo nível que deve ser incorporado em aplicações por desenvolvedores. Muitas vezes, as pessoas são enganadas pela descrição do Lucene e esperar que ela incluir os tipos de características SearchBlox oferece.

LingPipe e ortográficas variação-wow! Sentimos como se tivéssemos acabado de entrar no meio de um curso de análise de PhD nível linguístico. Bob Carpenter é um pernamenta endary neste espaço e um autor de renome.

Michaels.com e TheServerSide nos mostram que usando o Lucene não exige código complexo, e ser inteligente na forma como Lucene é incorporada rendimentos nifty efeitos. Indexação valores hexadecimal RGB e fornecendo indexação externo e configuração de busca são dois exemplos de simples e demonstrativas técnicas travelmente útil.

Gostaríamos novamente de agradecer aos colaboradores desses estudos de caso para a sua tempo e sua vontade de compartilhar o que tenho feito para seu benefício.

## Instalando Lucene

---



A versão Java do Lucene é apenas outro arquivo JAR. Usando a API Lucene em sua código requer apenas este único arquivo JAR no seu classpath de compilação e execução. Este apêndice fornece as especificações de onde obter Lucene, como trabalhar com o conteúdo, distribuição e como construir Lucene diretamente do seu código fonte. Se você está usando uma porta de Lucene em um idioma diferente do Java, consulte o capítulo 9 ea documentação fornecida com a porta. Este apêndice cobre o Java única versão.

## A.1 instalação Binary

---

Para obter a distribuição binária do Lucene, siga estes passos:

- 1 Baixar a versão mais recente Lucene binário da área de download da Jakarta site web: <http://jakarta.apache.org>. No momento da redação deste texto, o última versão é 1.4.2; as etapas subsequentes assume esta versão. Download nem o zip. ou. tar.gz, qualquer formato é mais conveniente para seu ambiente.
- 2 Extraia o arquivo binário para o diretório de sua escolha em seu sistema de arquivos. O arquivo contém um diretório de nível superior chamada lucene-1.4.2, portanto é seguro para extrair para c: \ no Windows ou seu diretório home no UNIX. No Windows, se você tem WinZip acessível, usá-lo para abrir o arquivo zip. e extraia seu conteúdo para c: \. Se você estiver no UNIX ou usando cygwin no Windows, descompacte e untar (alcatrão  
`zxvf lucene-1.4.2.tar.gz`) O arquivo. Tar.gz no seu diretório home.
- 3 Criado sob o lucene-1.4.2 diretório, você encontrará lucene-1.4.2.jar. Este é o único arquivo necessário para introduzir Lucene em seus aplicativos. Como você incorporar arquivo JAR do Lucene para a sua aplicação depende seu ambiente, há inúmeras opções. Recomendamos o uso de Ant para construir o código do seu aplicativo. Tenha certeza que seu código é compilado contra o JAR Lucene usando as opções do classpath `<javac>` tarefa.
- 4 Incluir o arquivo JAR do Lucene na distribuição da sua aplicação de forma adequada. Por exemplo, uma aplicação web usando o Lucene incluiria lucene-1.4.2.jar no diretório WEB-INF/lib. Para aplicações de linha de comando, seja Lucene certeza está no classpath ao lançar a JVM.

A distribuição binária inclui uma quantidade substancial de documentação, incluindo-Javadocs ing. A raiz da documentação é docs / index.html, que você pode aberta em um navegador da web. Distribuição Lucene também dois navios de demonstração aplicátioms. Pedimos desculpas antecipadamente para o estado bruto dessas demos-lhes falta

polonês quando se trata de facilidade de uso, mas a documentação (encontrado em docs / demo.html) descreve como usá-las passo a passo, mas que também abrange os princípios de executá-los aqui.

## A.2 Executando o demo de linha de comando

---

A demo Lucene de linha de comando consiste em duas de linha de comando programas: um que os índices de uma árvore de diretórios de arquivos e outra que fornece uma simples pesquisa interface. Para executar esse demo, definir o diretório de trabalho atual para o diretório onde a distribuição binária foi ampliado. Em seguida, execute o `IndexFiles` programa como este:

```
java-cp lucene-1.4.2.jar; lucene-demos-1.4.2.jar
⇒ org.apache.lucene.demo.IndexFiles docs

.

.

acrescentando docs / queryparsersyntax.html
acrescentando docs / resources.html
acrescentando docs / systemproperties.html
acrescentando docs / whoweare.html
9454 milissegundos total
```

Este comando índices de todo o `docs` árvore de diretórios (339 arquivos em nosso caso) em um índice armazenado no índice subdiretório do local onde você executou o comando.

**NOTA** Literalmente todos os arquivos no `docs` árvore de diretórios é indexada, inclusive. gif e arquivos. jpg. Nenhum dos arquivos são analisados, em vez disso, cada arquivo é indexado por streaming em seus bytes `StandardAnalyzer`.

Para pesquisar o índice acabou de criar, executar `SearchFiles` desta maneira:

```
java-cp lucene-1.4.2.jar; lucene-demos-1.4.2.jar
⇒ org.apache.lucene.demo.SearchFiles

Consulta: IndexSearcher E QueryParser
Procurando por: + + indexsearcher QueryParser
10 total documentos correspondentes
0. docs / api / index-all.html
1. docs / api / allclasses-frame.html
2. docs / api / allclasses-noframe.html
3. docs / api / org / apache / lucene busca / / classe de uso /
Query.html
4. docs / api / overview-summary.html
5. docs / api / overview-tree.html
6. docs/demo2.html
```

```
7. docs/demo4.html
8. docs / api / org / apache / lucene / search / package summary.html-
9. docs / api / org / apache / lucene / search / package tree.html-
```

SearchFiles prompts de forma interativa com Consulta::QueryParser é usado com Standard-Analisador para criar um Pergunta. Um máximo de 10 hits são mostrados em um momento, se mais, você pode folhear-los. Pressione Ctrl-C para sair do programa.

### A.3 Executando o demo aplicação web

---

A demo web é um pouco envolvidos para definir e executar corretamente. Você precisa de um web container, o nosso instruções são para o Tomcat 5. A documenta-docs / demo.htmlção fornece instruções detalhadas para configurar e executar o web aplicações ção, mas você também pode seguir os passos apresentados aqui.

O índice usado pelo aplicativo web é um pouco diferente que no command line-demo. Primeiro, ela se restringe a indexação só. Html,. Htm, e. Txt arquivos. Cada arquivo ele processa (incluindo arquivos. Txt) é analisado usando um costume rudi-parser HTML complementar. Para construir o índice, inicialmente, execute IndexHTML:

```
java -cp lucene-1.4.2.jar; lucene-demos-1.4.2.jar
      org.apache.lucene.demo.IndexHTML-create-indice WebIndex docs

.

.

acrescentando docs / resources.html
acrescentando docs / systemproperties.html
acrescentando docs / whoweare.html
Otimizando index ...
7220 milissegundos total
```

O índice WebIndex switch define o local do diretório do índice. Em um momento, você terá o caminho completo para este diretório para configurar a aplicação web. O final docs argumento para IndexHTML é a árvore de diretórios para o índice. O -Criar switch cria um índice a partir do zero. Remover essa opção para atualizar o índice com arquivos que foram adicionados ou alterados desde a última vez que o índice foi construído.

Em seguida, implantar luceneweb.war (a partir do diretório raiz do extraído distribuição) em CATALINA\_HOME / webapps. Inicie o Tomcat, aguarde o recipiente para completar a rotina de inicialização, em seguida, editar CATALINA\_HOME/webapps/lucene-web / configuration.jsp utilizando um editor de texto (Tomcat deveria ter se expandido a guerra. arquivo em um diretório luceneweb automaticamente). Alterar o valor de indexlocation apropriadamente, como neste exemplo, especificando o caminho absoluto para o índice que construído com IndexHTML:

```
Indexlocation String =
"/ Dev/LuceneInAction/install/lucene-1.4.2/webindex";
```

Agora você está pronto para tentar a aplicação web. Visite <http://localhost:8080/lucene-web> no seu navegador web, e você deve ver "Welcome to the Template Lucene aplicação ... "(você também pode alterar o cabeçalho e rodapé em configuration.jsp). Se tudo estiver bem com a sua configuração, em busca de Lucene específicas palavras como "E QueryParser Analyzer" deve listar resultados válidos com base na Documentação do Lucene.

Você pode tentar clicar em um dos resultados os links de pesquisa e receberá um erro. IndexHTML os índices de um url campo, que neste caso é um caminho relativo de docs / .... Para

fazer as ligações resultado funcionar corretamente, copie o docs diretório do Lucene distribuição para CATALINA\_HOME / webapps / luceneweb.

Sim, estes passos são um pouco mais manual do que deveriam. Tenha certeza de que melhorias para os aplicativos do Lucene exemplo, são em nossa lista de coisas a fazer, logo que estamos terminou de escrever este livro!

**TIP** Cool Hand Luke. Agora que você já construiu dois índices, um para o comando demo de linha e outro para o demo aplicação web, é um momento perfeito para tentar Luke. Consulte a seção 8.2 para detalhes sobre como usar Lucas. Apontá-lo para o índice, e surfar um pouco para ter uma idéia de Lucas e os conteúdos do índice.

## A.4 Gerando a partir da fonte

---

Lucene código-fonte é livre e facilmente disponíveis a partir do CVS Apache Jakarta é repositório. Os pré-requisitos para a obtenção e construir Lucene da fonte são CVS client, Java Developer Kit (JDK) e Apache Ant. Siga estes passos para construir Lucene:

- 1 Confira o código fonte do repositório do Apache CVS. Siga as instruções no web site Jakarta (<http://jakarta.apache.org>) para acessar o repositório usando o acesso somente leitura anônimos. Isto resume-se à execução de orçamentos  
ção os seguintes comandos (a partir de cygwin no Windows, ou um shell UNIX):

```
cvs-d: pserver: anoncvs@cvs.apache.org: / home / login cvspublic
password: anoncvs
```

```
cvs-d: pserver: anoncvs@cvs.apache.org: / home / cvspublic
checkout jakarta-lucene
```

- 2 Construir Lucene com Ant. No prompt de comando, defina o seu trabalho atual para o diretório onde você check-out do CVS Lucene volte a Tory (C:\apache\jakarta-lucene, por exemplo). Tipo formiga no comando

linha. JAR Lucene será compilado para o subdiretório construir. O JAR nome do arquivo é lucene-<versão>. jar, onde <versão> depende do cur- Estado aluguel do código que você obteve.

- 3 Executar os testes de unidade. Se a construção Ant bem-sucedida, próxima corrida `formiga teste` (JUnit é adicionar JAR para `ANT_HOME / lib` se não estiver lá) e garantir que todas as Lucene testes da unidade de passagem.

Lucene usa gramáticas JavaCC para `StandardTokenizer`, `QueryParser`, E os `demo HTMLParser`. O já compilado. Versão java dos arquivos. Jj existe no CVS o código-fonte, assim JavaCC não é necessária para a compilação. No entanto, se você deseja modificar as gramáticas parser, você precisa JavaCC, você também deve executar o `formiga javacc` alvo. Você pode encontrar mais detalhes no arquivo `build.txt` no diretório raiz do Repositório CVS do Lucene.

## A.5 Troubleshooting

---

Nós preferimos não tentar adivinhar que tipo de problemas que você pode correr para que você siga as etapas para instalar Lucene, construir Lucene, ou executar o demos. Verificar o FAQ, buscar nos arquivos do lucene usuário lista de e-mail, e usando o Lucene é questão- sistema de monitoramento são bons primeiros passos quando você tiver dúvidas ou problemas. Você vai encontrar detalhes no site do Lucene: <http://jakarta.apache.org/lucene>.

# **Lucene formato de índice**

Até agora, temos tratado o índice Lucene mais ou menos como uma caixa preta e ter concerned-nos apenas com a sua vista lógico. Embora você não precisa de substand detalhes estrutura do índice, a fim de usar Lucene, você pode estar curioso sobre o "Mágica". Lucene estrutura do índice é um estudo de caso em si de dados altamente eficiente estruturas e arranjo inteligente para maximizar o desempenho e minimizar uso de recursos. Você pode vê-lo como uma realização puramente técnica, ou você pode ver como um trabalho magistral de arte. Há algo intrinsecamente belo sobre repre-representando a estrutura rica da maneira mais eficiente possível. (Considere a infor-informações representadas por fórmulas fractal ou DNA como prova da natureza.)

Neste apêndice, veremos a visão lógica de um índice Lucene, onde nós documentos alimentado em Lucene e recuperados los durante as pesquisas. Então, nós vamos expor a estrutura interna do índice invertido do Lucene.

## B.1 exibição do índice Logical

Vamos primeiro dar um passo atrás e começar com uma rápida revisão do que você já sabe sobre o índice do Lucene. Considere a figura B.1. A partir da perspectiva de um software desenvolvedor usando Lucene API, um índice pode ser considerado uma caixa preta representada pelo abstrato `Diretório classe`. Quando a indexação, você cria instâncias do Lucene `Documento classe` e preenchê-lo com `Campos` que consistem de nome e valor

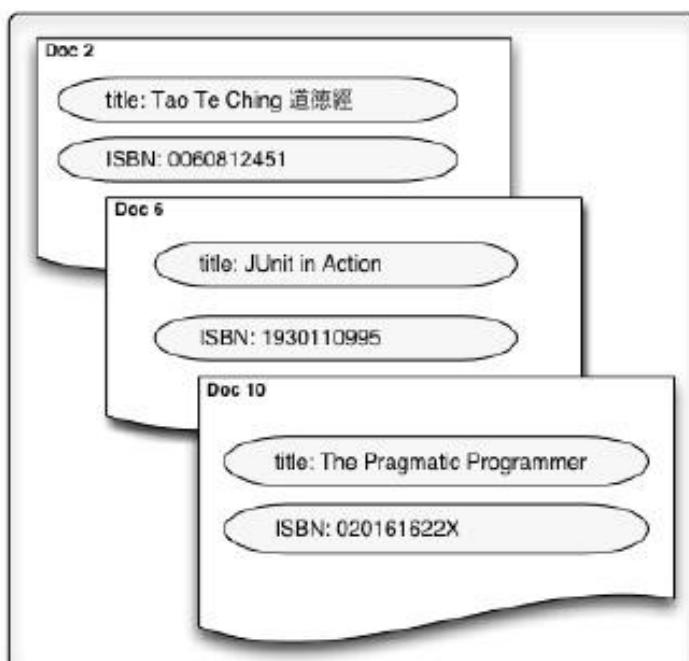


Figura B.1  
A lógica, visão de caixa-preta  
de um índice Lucene

pares. Tal Documento é, então, indexada passando-a para `IndexWriter.addDocument` (`Documento`). Ao pesquisar, você usar novamente o resumo `Diretório classe para repre-enviou o índice`. Você passa a `Diretório` ao `IndexSearcher classe` e em seguida, encontrar Documentos que correspondem a uma determinada consulta, passando termos de pesquisa encapsulado no Pergunta objeto para um dos `IndexSearcher'Procurar métodos s`. Os resultados estão combinando Documentos representado pelo `Hits` objeto.

## B.2 Sobre a estrutura de índice

---

Quando descrevemos Lucene `Diretório classe` na seção 1.5, destacamos que uma de suas subclasses concretas, `FSDirectory`, Armazena o índice em um arquivo de sistema di-Tory. Temos também utilizado `Indexador`, Um programa de indexação de arquivos de texto, mostrado na lista de ing 1.1. Lembre-se que vários argumentos especificado quando invocado `Indexador` na linha de comando e que um desses argumentos era o diretório no que nós queríamos `Indexador` para criar um índice Lucene. O que significa que o olhar de diretório como uma vez `Indexador` é feito a correr? O que ele contém? Nesta seção, nós vamos peek em um índice Lucene e explicar a sua estrutura.

Lucene suporta duas estruturas índice: índices de vários arquivos e compostos índices. O primeiro é o original, a estrutura mais velhos índice, o último foi introduzido em Lucene 1.3 e fez o padrão na versão 1.4. Vamos olhar para cada tipo de índice estrutura, começando com vários arquivos.

### B.2.1 Compreender a estrutura do índice de vários

Se você olhar para o diretório índice criado pelo nosso `Indexador`, Você verá um número de os arquivos cujos nomes podem parecer aleatórias em primeiro lugar. Estes são arquivos de índice, e eles olham semelhantes aos mostrados aqui:

-Rw-rw-r -	1	otis	otis	4	Novem22o22:43	deletable
-Rw-rw-r -	1	otis	otis	1000000	Novem22o22:43	_lfyc.f1
-Rw-rw-r -	1	otis	otis	1000000	Novem22o22:43	_lfyc.f2
-Rw-rw-r -	1	otis	otis	31030502	Novem22o22:28	_lfyc.fdt
-Rw-rw-r -	1	otis	otis	8000000	Novem22o22:28	_lfyc.fdx
-Rw-rw-r -	1	otis	otis	16	Novem22o22:28	_lfyc.fnm
-Rw-rw-r -	1	otis	otis	1253701335	Novem22o22:43	_lfyc.frq
-Rw-rw-r -	1	otis	otis	1871279328	Novem22o22:43	_lfyc.prx
-Rw-rw-r -	1	otis	otis	14122	Novem22o22:43	_lfyc.tii
-Rw-rw-r -	1	otis	otis	1082950	Novem22o22:43	_lfyc.tis
-Rw-rw-r -	1	otis	otis	18	Novem22o22:43	segmentos

Observe que alguns arquivos compartilhar o mesmo prefixo. Neste índice exemplo, um número de arquivos começam com o prefixo `_lfyc`, seguido por várias extensões. Isso nos leva a a noção de segmentos.

### Segmentos de índice

Um índice Lucene consiste em um ou mais segmentos e cada segmento é formado de vários arquivos de índice. Arquivos de índice que pertencem ao mesmo segmento compartilham um com-

seg prefixo e diferem no sufixo. No índice de exemplo anterior, o índice de con-  
sistiu de um único segmento cujos arquivos iniciados com \_lfyc:

O exemplo a seguir mostra um índice com dois segmentos, \_lfyc e \_gabh:

-Rw-rw-r -	1	otis	otis	4	Novembro22:43	deletable
-Rw-rw-r -	1	otis	otis	1000000	Novembro22:43	_lfyc.f1
-Rw-rw-r -	1	otis	otis	1000000	Novembro22:43	_lfyc.f2
-Rw-rw-r -	1	otis	otis	31030502	Novembro22:28	_lfyc.fdt
-Rw-rw-r -	1	otis	otis	8000000	Novembro22:28	_lfyc.fdx
-Rw-rw-r -	1	otis	otis	16	Novembro22:28	_lfyc.fnm
-Rw-rw-r -	1	otis	otis	1253701335	Novembro22:43	_lfyc.frq
-Rw-rw-r -	1	otis	otis	1371279328	Novembro22:43	_lfyc.prx
-Rw-rw-r -	1	otis	otis	14122	Novembro22:43	_lfyc.tii
-Rw-rw-r -	1	otis	otis	1082950	Novembro22:43	_lfyc.tis
-Rw-rw-r -	1	otis	otis	1000000	Novembro22:43	_gabh.f1
-Rw-rw-r -	1	otis	otis	1000000	Novembro22:43	_gabh.f2
-Rw-rw-r -	1	otis	otis	31030502	Novembro22:28	_gabh.fdt
-Rw-rw-r -	1	otis	otis	8000000	Novembro22:28	_gabh.fdx
-Rw-rw-r -	1	otis	otis	16	Novembro22:28	_gabh.fnm
-Rw-rw-r -	1	otis	otis	1253701335	Novembro22:43	_gabh.frq
-Rw-rw-r -	1	otis	otis	1371279328	Novembro22:43	_gabh.prx
-Rw-rw-r -	1	otis	otis	14122	Novembro22:43	_gabh.tii
-Rw-rw-r -	1	otis	otis	1082950	Novembro22:43	_gabh.tis
-Rw-rw-r -	1	otis	otis	18	Novembro22:43	segmentos

Você pode pensar em um segmento como um subíndice, apesar de cada segmento não é totalmente índice independente.

Como você pode ver na figura B.2, cada segmento contém um ou mais Documentos, os mesmos que adicionamos ao índice com a addDocument (Documento) método na IndexWriter classe. Até agora você deve estar se perguntando qual a função segmentos servir em um índice Lucene, o que se segue é a resposta para essa pergunta.

### Indexação incremental

Usando segmentos permite adicionar rapidamente novas Documentos para o índice, adicionando-a recém-criada segmentos de índice e só periodicamente fundi-las com outros, segmentos existentes. Este processo faz com adições eficiente porque minimiza modificações físicas índice. Figura B.2 mostra um índice que detém 34 Documentos. Esta figura mostra um unoptimized índice que contém vários segmentos. Se isso índice fosse otimizado usando os parâmetros padrão do Lucene indexação, todos os 34 dos seus documentos seriam fundidos em um único segmento.

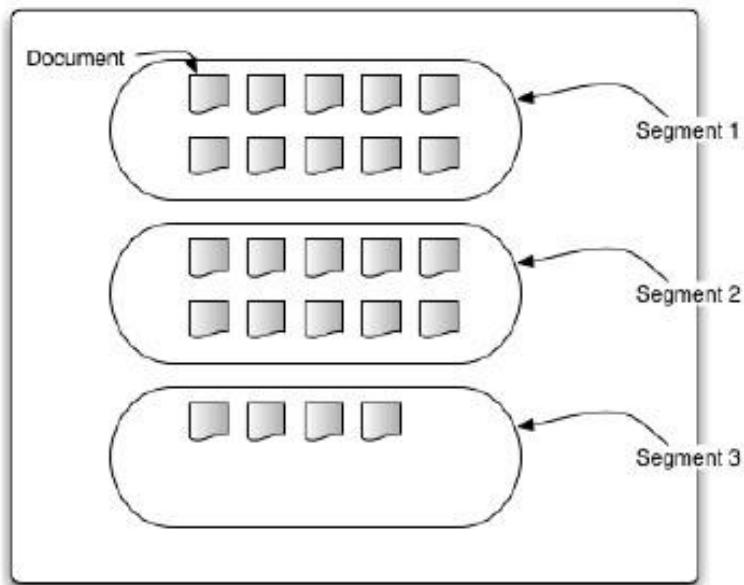


Figura B.2  
Índice unoptimized com  
3 segmentos, segurando  
34 documentos

Um dos pontos fortes do Lucene é que suporta indexação incremental, o que não é algo cada biblioteca é capaz de IR. Considerando que algumas bibliotecas IR necessidade de rein-

dex todo o corpo, quando novos dados são adicionados ao seu índice, Lucene não.

Depois que um documento foi adicionado a um índice, o seu conteúdo é imediatamente pesquisáveis. Em IR terminologia, este recurso importante é chamado indexação incremental. O fato de que Lucene suporta indexação incremental faz Lucene adequado para ambientes que lidam com grandes massas de informações em que completa reindexing seria complicado.

Porque novos segmentos são criados como novos Documentos são indexados, o número de segmentos e, portanto, os arquivos de índice, varia durante a indexação está em andamento. Uma vez que um

índice é totalmente construída, o número de arquivos de índice e segmentos permanece estável.

#### Um olhar mais atento arquivos de índice

Cada arquivo de índice traz um certo tipo de informação essencial para Lucene. Se houver arquivo de índice é modificado ou removido por qualquer outra coisa do que Lucene em si, o índice

corrompido, ea única opção é uma reindexação completa do original de dados. Por outro lado, você pode adicionar arquivos aleatórios para um diretório índice Lucene sem corromper o índice. Por exemplo, se adicionar um arquivo chamado random-document.txt para o diretório do índice, como mostrado aqui, Lucene ignora esse arquivo, e os índice não se corrompido:

```
-Rw-rw-r -      1 otis      otis      4      22 de novembro 22:43 deletable
-Rw-rw-r -      1 otis      otis      1000000   22 de novembro 22:43 lfyc.fl
```

## Lucene formato de índice

```

-Rw-rw-r - 1 otis          otis      1000000  Novembro22:43 _lfyc.f2
-Rw-rw-r - 1 otis          otis      31030502  Novembro22:28 _lfyc.fdt
-Rw-rw-r - 1 otis          otis      8000000  Novembro22:28 _lfyc.fdx
-Rw-rw-r - 1 otis          otis      16       Novembro22:28 _lfyc.fnm
-Rw-rw-r - 1 otis          otis      1253701335 Novembro22:43 _lfyc.frq
-Rw-rw-r - 1 otis          otis      1871279328 Novembro22:43 _lfyc.prx
-Rw-rw-r - 1 otis          otis      14122    Novembro22:43 _lfyc.tii
-Rw-rw-r - 1 otis          otis      1082950   Novembro22:43 _lfyc.tis
-Rw-rw-r - 1 otis          otis      128     Novembro22:34

⇒ random-document.txt
-Rw-rw-r - 1 otis

```

otis 18 22 de novembro segmentos 22:43

O segredo para isso é o arquivo de segmentos. Como você deve ter adivinhado a partir do seu nome,

o arquivo segmentos armazena os nomes de todos os segmentos índice existente. Antes de acesso

ing todos os arquivos no diretório do índice, Lucene consulta este arquivo para descobrir qual arquivos de índice para abrir e ler. Nossa índice de exemplo, tem um único segmento, \_lfyc, cujo nome é armazenada neste arquivo segmentos, assim Lucene sabe a olhar apenas para os arquivos

com o \_lfyc prefixo. Lucene também limita-se a arquivos com extensões conhecidas, tais como. fdt., FDX, e outras extensões mostrado em nosso exemplo, por isso mesmo guardar um ficheiro

com um prefixo segmento, tais como \_lfyc.txt, não vai jogar fora Lucene. É claro, polimentação um diretório com índice de não-Lucene arquivos é fortemente desencorajada.

O número exato de arquivos que constituem um índice Lucene e cada segmento varia de índice para índice e depende do número de campos do índice de contém. No entanto, cada índice contém um arquivo de segmentos único e uma única deletable arquivo. O último arquivo contém informações sobre documentos que foram marcadas para exclusão. Se você olhar para trás, o exemplo anterior, você verá dois arquivos de índice com uma extensão de fN., onde Né um número. Esses arquivos correspondem às indexados campos presentes na indexados Documentos. Lembre-se que Indexador a partir da listagem 1,1 cre-

Lucene ated Documentos com dois campos: um texto conteúdo campo e uma palavra-chave file-

nome de campo. Porque este índice contém dois campos indexados, nosso índice contém dois arquivos com a extensão fN.. Se este índice teve três campos indexados, um arquivo chamado \_lfyc.f3 também estar presentes no diretório índice. Ao olhar para arquivos de índice com esta extensão, você pode facilmente dizer quantos campos indexados tem um índice. Outra coisa interessante a se notar sobre esses. FN arquivos é o seu tamanho, o que reflete o número de Documentos com esse campo. Agora que você sabe disso, você pode dizer que o índice anterior tem 1.000.000 documentos apenas por olhar para os arquivos no diretório índice.

Criação de um índice de vários

Até agora você deve ter uma boa compreensão da estrutura do índice de vários arquivos, mas como

você usar a API para instruir Lucene para criar um índice de vários arquivos e não o padrão

composto arquivo index? Vamos olhar para trás em nossos fiéis Indexador a partir da listagem 1.1. Em

que lista, você mancha o seguinte:

```
Escritor IndexWriter IndexWriter = new (indexDir,
    nova StandardAnalyzer (), true);
writer.setUseCompoundFile (false);
```

Porque a estrutura do índice composto de arquivo é o padrão, vamos desativá-lo e mudar para um índice de vários chamando `setUseCompoundFile (false)` em um índice Escritor exemplo.

### B.2.2 Compreender a estrutura do índice composto

Quando descrevemos índices de vários arquivos, dissemos que o número de arquivos de índice depende do número de campos indexados presente no índice. Também os homens-mencionou que novos segmentos são criados como documentos são adicionados a um índice, já que

um segmento é composto por um conjunto de arquivos de índice, isso resulta em uma variável e, possivelmente, grande número de arquivos em um diretório do índice. Embora a estrutura do índice de vários é simples e funciona para a maioria dos cenários, não é adequado para ambientes com grande número de índices, índices com um grande número de campos, e outros ambiente onde usando Lucene resultados em um grande número de arquivos de índice.

A maioria, senão todos, os sistemas contemporâneos limitar o número de arquivos em o sistema que podem ser abertos ao mesmo tempo. Lembre-se que Lucene cria seg-novo mentos como os novos documentos são adicionados, e de vez em quando ele mescla-los para reduzir

o número de arquivos de índice. No entanto, enquanto o processo de mesclagem é a execução, o número de arquivos de índice dobra. Se Lucene é usado em um ambiente com muita índices que estão sendo pesquisados ou indexados ao mesmo tempo, é possível chegar a o limite de arquivos abertos definido pelo sistema operacional. Isso também pode acontecer com um

índice Lucene único se o índice não é otimizado, ou se outras aplicações estão running simultaneamente e manter muitos arquivos abertos. Lucene uso de arquivo aberto han-spindles depende da estrutura e do estado de um índice. Mais tarde, no apêndice, fórmulas presentes para o cálculo do número de arquivos abertos que Lucene exigirá para manipulação de seus índices.

#### Índice de arquivos compostos

A única diferença visível entre os índices compostos de vários arquivos e é o conteúdo de um diretório índice. Aqui está um exemplo de um índice composto:

-Rw-rw-r -	1	otis	otis	418	12	de outubro	2.cfs	22:13
-Rw-rw-r -	1	otis	otis	4	12	de outubro	22:13	deletable
-Rw-rw-r -	1	otis	otis	15	de outubro	12	segmentos	22:13

Em vez de ter que abrir e ler arquivos de 10 do índice, como no de vários índice, Lucene deve abrir somente dois arquivos ao acessar este índice composto, consumindo menos sistema recursos.<sup>1</sup> O índice composto reduz a número de arquivos de índice, mas o conceito de segmentos, documentos, campos e termos ainda se aplica. A diferença é que um índice composto contém um único arquivo. Cfs por segmento, enquanto que cada segmento em um índice de vários contém consiste em sete arquivos diferentes. A estrutura composta encapsula os arquivos de índice individual em um arquivo único cfs..

### Criação de um índice composto

Porque a estrutura do índice composto é o padrão, você não tem que fazer qualquer coisa a especificá-lo. No entanto, se você gosta de código explícito, você pode chamar o `setUseCompoundFile(boolean)` método, passando-o um verdadeiro valor:

```
Escritor IndexWriter writer = new (indexDir,
    nova StandardAnalyzer (), true);
writer.setUseCompoundFile (true);
```

Agradavelmente, você não está bloqueado para o formato de vários arquivos ou compostos. Depois de índice  
ing, você ainda pode converter de um formato para outro.

### B.2.3 Conversão de uma estrutura de índice para o outro

É importante notar que você pode alternar entre as duas descritas índice de estruturas, em qualquer ponto durante a indexação. Tudo que você tem a fazer é chamar o do `IndexWriter setUseCompoundFiles(boolean)` método, a qualquer momento durante a indexação, da próxima vez

Lucene funde segmentos de índice, que irá converter o índice para qualquer estrutura que você especificou.

Da mesma forma, você pode converter a estrutura de um índice existente sem adicionar documentos mais do que isso. Por exemplo, você pode ter um índice de vários arquivos que você quer para converter a um composto de um, para reduzir o número de arquivos abertos usado por Lucene.

Para fazer isso, abra o seu índice com `IndexWriter`, Especificar a estrutura composta, otimizar o índice e feche-a:

```
Escritor IndexWriter writer = new (indexDir,
    nova StandardAnalyzer (), false);
writer.setUseCompoundFile (true);
writer.optimize ();
writer.Close ();
```

---

<sup>1</sup> Não contamos o arquivo deletable porque não tem de ser lido durante a indexação ou pesquisa.

Note que o terceiro `IndexWriter` parâmetro é falso para assegurar que a actual índice não é destruído. Discutimos otimizando índices na seção 2.8. Otimizing forças Lucene para fundir segmentos de índice, dando-lhe a chance de escrever -los em um novo formato especificado através do `setUseCompoundFile` (boolean) método.

## B.3 Escolhendo a estrutura do índice

---

Embora a mudança entre as duas estruturas índice é simples, você pode querer saber de antemão quantos arquivos abertos Lucene recursos serão usados ao acessar seu índice. Se você está projetando um sistema com múltiplas simultaneamente indexados e procurou índices, mas você vai definitivamente querer ter uma caneta e um pedaço de papel e fazer algumas contas simples, com nós agora.

### B.3.1 Cálculo do número de arquivos abertos

Vamos considerar um índice de vários arquivos primeiro. Um índice de vários arquivos de índice contém sete para cada segmento, um arquivo adicional para cada campo indexado por segmento, e um deletable único e um arquivo de segmentos único para todo o índice. Imagine um sistema de que contém 100 índices Lucene, cada uma com 10 campos indexados. Também assumir que a estes índices não são otimizados e que cada um tem nove segmentos que não foram fundidos em um único segmento, no entanto, como é frequentemente o caso durante a indexação. Se todos os 100 índices estão abertas para a busca, ao mesmo tempo, isso irá resultar em 15.300 aberto arquivos. Aqui está como nós temos esse número:

```
100 índices * (9 * segmentos por índice
                (7 arquivos por segmento + 10 arquivos para campos indexados))
= 100 * 9 * 17
= 15300 arquivos abertos
```

Embora os computadores de hoje normalmente pode lidar com isso muitos arquivos abertos, a maioria vem com um limite pré-configurados que é muito menor. No ponto 2.7.1, discutimos como verificar e alterar isso em alguns sistemas operacionais.

Em seguida, vamos considerar os mesmos 100 índices, mas desta vez usando o composto estrutura. Apenas um único arquivo com uma extensão de cfs. Seja criado por segmento, em adição a um deletable único e um arquivo de segmentos único para todo o índice. Portanto, se usarmos o índice composto de vários arquivos em vez do um, o número de aberto arquivos é reduzido a 900:

```
100 índices * (9 segmentos por índice * (1 arquivo por segmento))
= 100 * 9 * 1
= 900 arquivos abertos
```

A lição aqui é que se você precisa desenvolver Lucene software baseado em que executado em ambientes com um grande número de índices Lucene com um número de indexados campos, você deve considerar o uso de um índice composto. Claro, você pode usar um índice composto, mesmo se você estiver escrevendo um aplicativo simples que lida com um índice Lucene único.

### B.3.2 Comparando o desempenho

O desempenho é outro fator que você deve considerar ao escolher o índice de estrutura. Algumas pessoas têm relatado que criar um índice com um composto estrutura é 5-10% mais lento do que criar um índice de vários equivalentes; nosso índice ing teste de desempenho, mostrado na Listagem B.1, confirma isso. Neste teste, criamos dois índices paralelo com 25.000 documentos artificialmente criados cada um. No testTiming () método, tempo que o tempo que o processo de indexação leva para cada tipo de índice e afirmam que a criação do índice composto leva mais tempo do que criação de seu primo com diversos.

#### Comparação Listagem B.1 do composto de vários arquivos e performance index

```
public class CompoundVersusMultiFileIndexTest extends TestCase {

    Cdir Directory privado;
    mdir Directory privado;
    Coleção privada docs = loadDocuments (5000, 10);

    setUp protected void () throws IOException {
        String = indexDir
        System.getProperty ("java.io.tmpdir", "tmp") +
        System.getProperty ("File.separator") + "index-dir";

        String = cIndexDir indexDir + "composto";
        String = mIndexDir indexDir + "-multi";
        . (Novo arquivo (cIndexDir)) delete ();
        . (Novo arquivo (mIndexDir)) delete ();

        Cdir = FSDirectory.getDirectory (cIndexDir, true);
        mdir = FSDirectory.getDirectory (mIndexDir, true);
    }

    testTiming public void () throws IOException {
        longo cTiming = timeIndexWriter (Cdir, true);
        longo mTiming = timeIndexWriter (mdir, false);

        assertTrue (cTiming > mTiming);           b Timing composto maior
                                                    do timing de vários
        System.out.println ("Tempo Composto:" + (cTiming) + "ms");
        System.out.println ("Tempo de arquivo Multi-:" + (mTiming) + "ms");
    }
}
```

```
timeIndexWriter longo privado (dir Directory, boolean isCompound)
throws IOException {
    longo start = System.currentTimeMillis ();
    addDocuments (dir, isCompound);
    longa stop = System.currentTimeMillis ();
    retorno (stop - start);
}

privada addDocuments void (dir Directory, boolean isCompound)
throws IOException {
    Escriptor IndexWriter IndexWriter = new (dir, novos SimpleAnalyzer (),
        true);
    writer.setUseCompoundFile (isCompound);

    / / Altere para ajustar o desempenho da indexação com FSDirectory
    writer.mergeFactor = writer.mergeFactor;
    writer.maxMergeDocs = writer.maxMergeDocs;
    writer.minMergeDocs = writer.minMergeDocs;

    for (Iterator iter = docs.iterator (); iter.hasNext ()) {
        Documento doc = new Document ();
        String palavra = (String) iter.next ();
        doc.add (Field.Keyword ("keyword", palavra));
        doc.add (Field.UnIndexed ("não indexados", palavra));
        doc.add (Field.UnStored ("UnStored", palavra));
        doc.add (Field.Text ("text", palavra));
        writer.addDocument (doc);
    }
    writer.optimize ();
    writer.Close ();

}

privada loadDocuments Collection (int numDocs, int wordsPerDoc) {
    Colecção docs = new ArrayList (numDocs);
    for (int i = 0; i < numDocs; i + +) {
        Doc = new StringBuffer StringBuffer (wordsPerDoc);
        for (int j = 0; j < wordsPerDoc; j + +) {
            doc.append ("Bibamus");
        }
        docs.add (doc.toString ());
    }
    retorno docs;
}

}
```

- b Este teste confirma que a criação de um índice com a estrutura de alguns compostos é o mais lento do que a construção de um índice de vários arquivos. Exatamente quanto mais lenta varia e depende do número de campos, a sua extensão, os parâmetros de indexação

usado, e assim por diante. Por exemplo, você pode ser capaz de obter a estrutura composta Índice de superar o índice de vários ajustando alguns dos indexação parâmetros descritos na seção 2.7.

Aqui está o nosso conselho: Se você precisa espremer cada bit de desempenho da indexação fora do Lucene, use a estrutura do índice de vários, mas tente primeiro composto tuning indexação estrutura através da manipulação dos parâmetros de indexação coberto de seção 2.7. Essa diferença de desempenho ea diferença na quantidade de sistemas de temperatura recursos as duas estruturas de índice são as suas usar apenas diferenças notáveis. Todos

Características do Lucene funcionam igualmente bem com qualquer tipo de índice.

## B.4 índice invertido

---

Lucene usa uma estrutura de índice bem conhecido chamado de índice invertido. Muito simplesmente,

e, provavelmente, sem surpresa, um índice invertido é um arranjo de dentro para fora da tais documentos que os termos no centro das atenções. Cada termo refere-se aos documentos que as contenham. Vamos dissecar a nossa amostra livro índice de dados para obter um vislumbre mais profundo nos arquivos em um índice Diretório.

Independentemente de saber se você está trabalhando com um RAMDirectory, Uma FSDirectory,

ou qualquer outro Diretório implementação, a estrutura interna é um grupo de arquivos. Em um RAMDirectory, Os arquivos são virtuais e vivem inteiramente dentro de RAM. FSDirectory

literalmente representa um índice como um diretório do sistema de arquivos, como descrito anteriormente neste apêndice.

O modo de arquivo composto (adicionado no Lucene 1.3) acrescenta um toque adicional sobre os arquivos em um Diretório. Quando um IndexWriter está definido para arquivo composto

modo, o "arquivos" são gravadas em um arquivo único cfs., que alivia a comum problema de ficar sem identificadores de arquivo. Consulte a seção "Arquivos compostos índice"

### B.4.1 Dentro do índice

este apêndice para mais informações sobre o modo de arquivo composto. O formato de índice Lucene é detalhado em todos os seus detalhes sangrentos no site Lucene em <http://jakarta.apache.org/lucene/docs/fileformats.html>. Seria doloroso para nós, e tedioso para você, se conseguirmos repetir este informações detalhadas aqui. Pelo contrário, nós escolheram para resumir a estrutura do arquivo geral usando os nossos dados da amostra livro como um exemplo concreto.

Nosso resumo glosas sobre a maioria das complexidades de compressão de dados utilizado no

o real representações de dados. Esta extrapolação é útil para dar-lhe uma sensação para a estrutura em vez de ter sido apanhado nas minúcias (que, mais uma vez, são detalhadas sobre o site web Lucene).

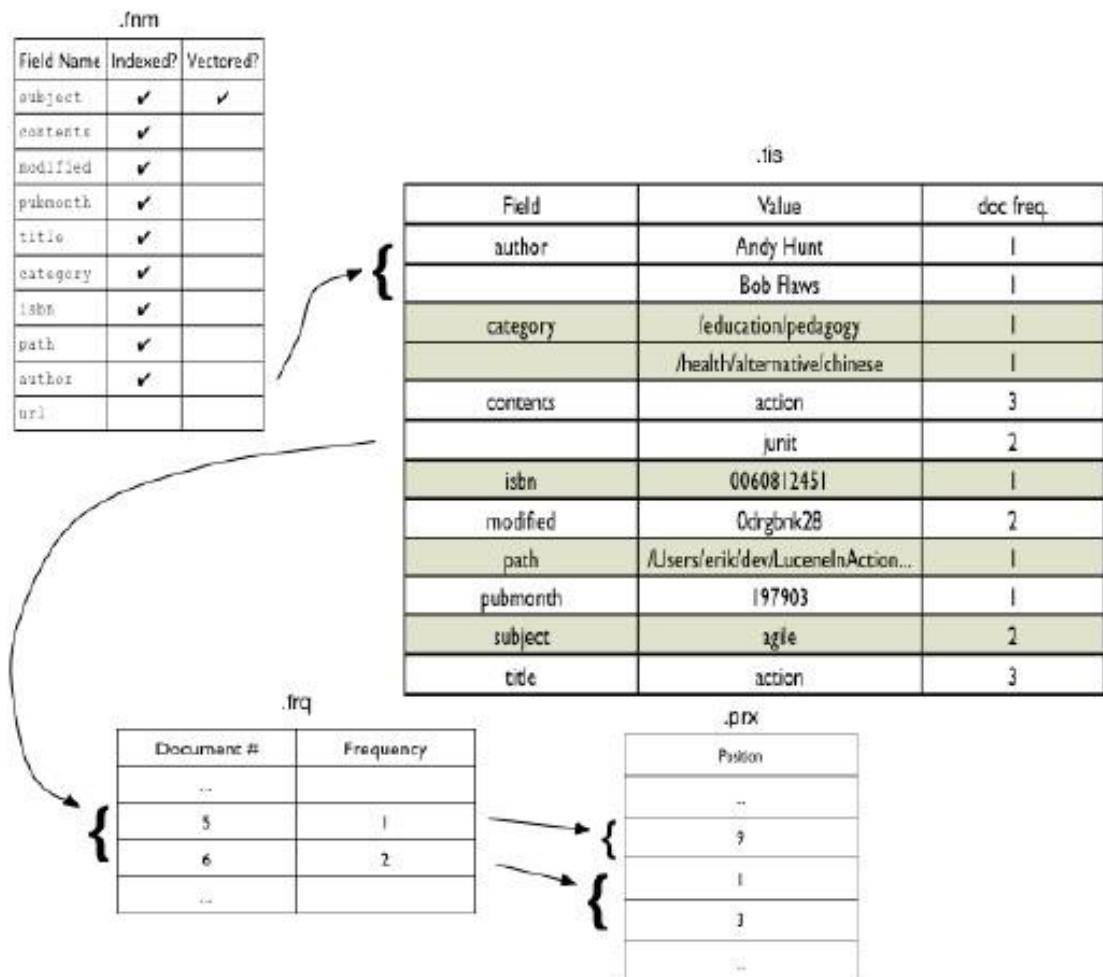


Figura B.3 Olhar detalhado dentro do formato de índice Lucene

Figura B.3 representa uma fatia de nosso índice livro da amostra. A fatia é de um único segmento (neste caso, tínhamos um índice otimizado com apenas um único segmento). O segmento é dado um prefixo de nome de ficheiro único (\_c neste caso).

As seções seguintes descrevem cada um dos arquivos mostrados na figura B.3 em mais detalhe.

#### Nomes de campos (. Fnm)

O arquivo. Fnm contém todos os nomes de campo usados pelos documentos na associados segmento. Cada campo é marcada para indicar se é indexada ou vectored. O ordem dos nomes de campo no arquivo. fnm é determinado durante a indexação e não é necessariamente em ordem alfabética. A posição de um campo no arquivo. Fnm é usada para associar

com os arquivos de normalização (arquivos com sufixo. f [0-9] \*). Nós não mergulhar no normalização arquivos aqui, consulte o site web Lucene para mais detalhes.

Em nosso índice de amostra, apenas o `assunto` campo é vectored. O `url` campo foi adicionado

como um `Field.UnIndexed` campo, que não é nem indexado nem vectored. O arquivo. Fnm mostrado na figura B.4 é uma visão completa do arquivo real.

#### Dicionário prazo (. Tis)

Todos os termos (tuplas de nome do campo e valor) em um segmento são armazenados no arquivo. Tis.

Termos são ordenados alfabeticamente pelo primeiro nome do campo e depois por valor dentro de uma

de campo. Cada entrada contém seu prazo freqüência documento: o número de documentos que contêm esse termo dentro do segmento.

Figura B.4 mostra apenas uma amostra dos termos em nosso índice, um ou mais de cada campo. Note que o `url` campo está em falta porque foi adicionado como um `UnIndexed` campo, que é armazenado e não apenas disponível como termos. Não é mostrado o arquivo. Tii, que é uma secção transversal do arquivo. tis projetado para ser mantido na memória física para de acesso aleatório para o arquivo. tis. Para cada termo no arquivo. Tis, o arquivo. Frq contém entradas para cada documento que contém o termo.

Em nosso índice de amostra, dois livros têm o valor "junit" no `conteúdo` campo: JUnit em Ação (ID documento 6), e Java Development com Ant (ID documento 5).

#### Freqüências prazo

Freqüências termo em cada documento estão listados no arquivo frq.. Em nossa amostra índice, Java Development com Ant (ID documento 5) tem o valor "junit" de vez em o conteúdo de campo. JUnit em Ação tem o valor "junit" duas vezes, desde que uma vez por o título e uma vez pelo sujeito. Nossa `conteúdo` campo é uma agregação de título, , assunto e autor. A freqüência de um termo em um documento fatores para a pontuação cálculo (ver secção 3.3) e, normalmente, aumenta a relevância de um documento para freqüências mais altas.

Para cada documento listados no arquivo frq., As posições de arquivo (. Prx) contém entradas para cada ocorrência do termo em um documento.

#### Posições prazo

O arquivo. Prx lista a posição de cada termo dentro de um documento. A posição informação é usada quando as consultas exigir-lo, como consultas frase e span consultas. Informações sobre a posição dos campos tokenized vem diretamente do token incrementos posição designada durante a análise.

Figura B.4 mostra três posições, para cada ocorrência do termo junit. O primeira ocorrência é documento em 5 (Java Desenvolvimento com Ant) na posição 9. No

caso de documento 5, o valor do campo (após análise) é "java desenvolvimento de formigas apache jakarta ant ferramenta de construção junit java desenvolvimento erik nascedouro steve lough-correu ". Utilizou-se o StandardAnalyzer; Assim parar com as palavras (com no Java Development com Ant, por exemplo) são removidos e não são contabilizados no posicional informação (ver secção 4.7.3 para mais informações sobre remoção de stop palavra e informação posicional). Documento 6, JUnit em Ação, tem um conteúdo campo contendo o valor "Junit" duas vezes, uma na posição 1 e novamente na posição 3: "A ação junit junit unidade testes mock objetos vincent Massol ted Husted ".<sup>2</sup>

## B.5 Resumo

---

A justificativa para a estrutura do índice é duplo: máximo desempenho e utilização de recursos mínimos. Por exemplo, se um campo não é indexado é muito operação rápida para dispensá-la inteiramente a partir de consultas com base na bandeira indexada de o arquivo. fnm. O arquivo. Tii, em cache na memória RAM, permite acesso aleatório rápido para o dicionário prazo arquivo. tis. Consultas span frase e não precisa procurar posicional informações se o termo em si não está presente. Racionalização das informações mais muitas vezes necessário, e minimizando o número de acessos a arquivos durante as pesquisas é de preocupação crítica. Estes são apenas alguns exemplos de como bem pensado o índice projeto da estrutura foi. Se esse tipo de baixo nível de otimização é de interesse, por favor referem-se ao índice Lucene detalhes de formato de arquivo no site do Lucene, onde detalhes temos encoberto aqui pode ser encontrado.

---

<sup>2</sup> Estamos em débito com Luke, o inspetor índice fantástico, por nos permitir facilmente reunir alguns dos dados fornecidas sobre a estrutura do índice.

# Recursos

---

Motores de busca da web são seus amigos. Tipo lucene em seu mecanismo de busca favorito, e você vai encontrar muitos interessantes Lucene projetos relacionados. Outro bom lugar para olhar é SourceForge, uma busca por lucene no SourceForge exibe uma série de projetos open-source escrito em cima do Lucene.

## C.1 Internacionalização

---

- Bray, Tim, "Personagens vs Bytes", <http://www.tbray.org/ongoing/When/200x/2003/04/26=UTF>
- Verde, Dale, "Trail: Internacionalização", <http://java.sun.com/docs/books/tutorial/i18n/index.html>
- Intertwingly ", Unicode e Weblogs," <http://www.intertwingly.net/blog/1763.html>
- Peterson, Erik, "Chinese Character Versão Dicionário Unicode", [http://www.mandarintools.com/chardict\\_u8.html](http://www.mandarintools.com/chardict_u8.html)
- Spolsky, Joel: "O Mínimo Absoluto Todo Software Developer Absolutamente, Positivamente Precisa Saber Sobre Unicode e Conjuntos de Caracteres (Sem Desculpas!)"  
<http://www.joelonsoftware.com/articles/Unicode.html>

## C.2 Detecção de idioma

---

- Apache patch Banco de Dados Bug: linguagem adivinhador contribuição, [http://issues.apache.org/bugzilla/show\\_bug.cgi?id=26763](http://issues.apache.org/bugzilla/show_bug.cgi?id=26763)
- JTextCat 0.1, <http://www.jedi.be/JTextCat/index.html>
- NGramJ, <http://ngramj.sourceforge.net/>

## C.3 vetores Term

---

- "Como LSI Works," [http://javelina.cet.middlebury.edu/lsa/out/lsa\\_explanation.htm](http://javelina.cet.middlebury.edu/lsa/out/lsa_explanation.htm)
- "Indexação semântica latente (LSI)," <http://www.cs.utk.edu/~lsi/>
- Stata, Raymie, Krishna Bharat, e Maghoul Farzin, "O termo vetor Banco de dados: Acesso Rápido aos Termos de indexação de páginas da Web ", <http://www9.org/w9cdrom/159/159.html>

## C.4 portas Lucene

---

- CLucene, <http://www.sourceforge.net/projects/clucene/>
- dotLucene, <http://sourceforge.net/projects/dotlucene/>
- Lupy <http://www.divmod.org/Home/Projects/Lupy/>,
- Plucene, <http://search.cpan.org/dist/Plucene/>
- PyLucene, <http://pylucene.osafoundation.org/>

## C.5 Os estudos de caso

---

- Alias-i <http://www.alias-i.com/>,
- jGuru, <http://www.jguru.com/>
- Michaels <http://www.michaels.com/>,
- Nutch, <http://www.nutch.org/>
- SearchBlox Software, <http://www.searchblox.com/>
- [Http://www.theserverside.com/](http://www.theserverside.com/) TheServerSide.com,
- XtraMind Technologies, <http://www.xtramind.com/>

## C.6 parsers Documento

---

- CyberNeko Ferramentas para XNI, [http://www.apache.org/~andyc/ neko / doc /](http://www.apache.org/~andyc/neko/doc/)
- Digestor, <http://jakarta.apache.org/commons/digester/>
- JTidy, <http://sourceforge.net/projects/jtidy>
- PDFBox, <http://www.pdfbox.org/>
- TextMining.org, <http://www.textmining.org/>
- Xerces2, <http://xml.apache.org/xerces2-j/>

## C.7 Diversos

---

- Calishain, Tara, e Rael Dornfest, Google Hacks (O'Reilly, 2003)
- Gilleland, Michael, "Distância Levenshtein, em três sabores:" <http://www.merriampark.com/ld.htm>
- GNU Compiler para o (GCJ) Java, <http://gcc.gnu.org/java/>
- Resultados de busca do Google para Lucene, <http://www.google.com/search?q=lucene>

- Jakarta Lucene, <http://jakarta.apache.org/lucene>
- Lucene Sandbox, <http://jakarta.apache.org/lucene/docs/lucene-sandbox/>
- Resultados da pesquisa SourceForge para Lucene, [http://sourceforge.net/search?type\\_of\\_search=soft&palavras=lucene](http://sourceforge.net/search?type_of_search=soft&palavras=lucene)
- Árvores de sufixos, <http://sequence.rutgers.edu/st/>
- SWIG, <http://www.swig.org/>

## C.8 software IR

---

- resultados dmoz para Recuperação de Informação, [http://dmoz.org/Computers/Software/Information\\_Retrieval/](http://dmoz.org/Computers/Software/Information_Retrieval/)
- Egothor, <http://www.egothor.org/>
- Google Directory para resultados Recuperação de Informação, [http://directory.google.com.br/Top/Computadores/Software/Information\\_Retrieval/](http://directory.google.com.br/Top/Computadores/Software/Information_Retrieval/)
- <Http://www.sourceforge.net/projects/harvest/> colheita,
- Harvest-NG, <http://webharvest.sourceforge.net/ng/>
- ht: // Dig, <http://www.htdig.org/>
- Gigabytes gestão para Java (MG4J), <http://mg4j.dsi.unimi.it/>
- Namazu, <http://www.namazu.org/>
- Ferramentas de busca para sites da Web e Intranets, <http://www.searchtools.com/>
- SWISH + +, <http://homepage.mac.com/pauljlucas/software/swish/>
- SWISH-E, <http://swish-e.org/>
- Verity, <http://www.verity.com/>
- WebGlimpse, <http://webglimpse.net>
- Xapian, <http://www.xapian.org/>

## Publicações C.9 corte de Doug

---

Lista oficial de Doug online de publicações, de onde foi derivada, está disponível em <http://lucene.sourceforge.net/publications.html>.

### C.9.1 papéis Conferência

- "Um intérprete de regras fonológicas," co-autoria com J. Harrington, Anais do Instituto de Acústica Conferência de Outono, nov 1986

- "Teatro de Informação versus Refinaria da Informação", co-autoria com J. Pedersen, P.-K. Halvorsen, e M. Withgott, AAAI Spring Symposium on Baseado em texto Sistemas Inteligentes, março 1990
- "Otimizações para Manutenção Índice Dynamic invertido", em co-autoria com J. Pedersen, Proceedings of SIGIR 90, setembro 1990
- "Uma Arquitetura Orientada a Objetos para Recuperação de texto", em co-autoria com JO Pedersen e P.-K. Halvorsen, Proceedings of PRAI 91, abril de 1991
- "Search Snippet: uma Abordagem única frase para acessar o texto," co-autoria com JO Pedersen e Tukey JW, Proceedings of the 1991 Conjunto estatística-cal Reuniões, agosto 1991
- "A Tagger parte de fala-Prático", co-autoria com J. de Kupiec, J. Pedersen, e P. Sibun, Anais da Terceira Conferência sobre Aplicada Natural Lan-medidor de Processamento, abril 1992
- "Scatter / Gather: uma abordagem baseada em Cluster de Navegação de Documento Grande Coleções ", co-autoria com D. Karger, J. Pedersen, e Tukey J., Proceedings de SIGIR 92, Junho de 1992
- "Constant Interaction-Time Scatter / Gather Navegação do Very Large Documentos ambiente ", co-autoria com D. Karger e Pedersen J., Proceedings de SIGIR 93, Junho de 1993
- "Portando um Tagger Part-de-Speech para sueco," Nordic Datalingvistik Dagen 1993, de Estocolmo, Junho de 1993
- "Otimizações Espaço para Ranking Total", co-autoria com J. Pedersen, Anais do PRAI 97, Montreal, Quebec, junho de 1997

### C.9.2 U. S. Patentes

- 5278980: "técnica iterativa para a formação de consulta e uma frase-informação sistema de recuperação ção empregando mesmo ", com J. Pedersen, P.-K. Halvorsen, J. Tukey, E. Bier, e D. Bobrow, arquivado agosto 1991
- 5442778: "Scatter recolher: um método baseado em cluster e aparelhos para testas-ing grandes coleções de documentos ", com J. Pedersen, Karger D. e J. Tukey, arquivado novembro 1991
- 5390259: "Métodos e aparelhos para selecionar semanticamente significativa imagens em uma imagem de documento sem decodificar o conteúdo da imagem ", com M. Withgott, S. Bagley, D. Bloomberg, D. Huttenlocher, R. Kaplan, T. Cass, P.-K. Halvorsen, e R. Rao, arquivado novembro 1991

- 5625554 "transdução de estado finito de formas de palavras relacionadas para indexação de texto e recuperação ", com P.-K. Halvorsen, R.M. Kaplan, L. Karttunen, M. Kay, e J. Pedersen, arquivado julho 1992
- 5.483.650 Método "de Clustering interação em tempo-Constant Aplicada à Documento Navegação ", com J. Pedersen e Karger D., arquivado novembro 1992
- 5.384.703 Method "e aparelhos para resumir os documentos de acordo ao tema ", com M. Withgott, arquivado julho 1993
- 5838323 "Documento síntese computador interface de usuário do sistema", com D. Rose, J Bornstein, e J. Hatton, arquivado setembro 1995
- 5867164 "sumarização interativo do documento", com D. Rosa, J. Born - stein, e J. Hatton, arquivado setembro 1995
- 5.870.740 System "e método para melhorar o ranking de informação resultados de recuperação para consultas curtas ", com D. Rose, arquivado setembro 1996



# índice

---

## A

---

abreviatura, manipulação de 355  
exatidão 360  
Ackley, Ryan 250  
Adobe Systems 235  
agente, distribuídos 349  
AliasAnalyzer 364  
Alias-i 361  
Almaer, Dion 371  
grafias alternativas 354  
análise de 103  
    durante a indexação 105  
    campo específico 108  
    línguas estrangeiras 140  
    em Nutch 145  
    posição lacunas 136  
    posicional questões gap 138  
    versus análise de 107  
    com QueryParser 106  
Analisadores de 19  
    282 adicionais  
    282 brasileiros  
    buffering 130  
    blocos de construção 110  
    built-in 104, 119  
    Chinese 282  
    escolher 103  
    CJK 282  
    Holandês 282  
    tipos de campo 105  
    para destacar 300  
    Francês 282  
    injetando sinônimos 129, 296  
    SimpleAnalyzer 108  
    Snowball 283

## B

---

StandardAnalyzer 120  
StopAnalyzer 119  
subpalavra 357  
usando WordNet 296  
visualizando 112  
WhitespaceAnalyzer 104  
    com 72 QueryParser  
Formiga  
    construção de 391 Lucene  
    construção de 310 Sandbox  
    indexação de um conjunto de arquivos 284  
Antiword 264  
ANTLR 100, 336  
Apache Jakarta 7, 9  
Apache Software Foundation 9  
Apache Software License 7  
Arábica 359  
arquitetura  
    design campo 374  
    TheServerSide  
        configuração de 379  
ASCII 142  
Análise de 142 línguas asiáticas

TooManyClauses  
    exceção 215  
usado com PhraseQuery 158  
aumentar 79  
    377 documentos  
documentos e campos de 38-39  
BrazilianAnalyzer 282

## C

---

C ++ 10  
CachingWrappingFilter 171, 177  
    caching DateFilter 173  
Cafarella, Michael 326  
Carpenter, Bob 351  
telefone celular, T9 WordNet  
    297 interface  
ChainedFilter 177, 304  
Chandler 307, 322  
charadas 125  
Análise chinês 142-143, 282  
CJK (Chinês Japonês  
    Coreano) 142  
CJKAnalyzer 143, 145, 282  
Clark, Andy 245  
Clark, Mike 214  
CLucene 314, 317  
    314 plataformas suportadas  
    Unicode suporte 316  
cor  
    distância fórmula 366  
    indexação 365  
interface de linha de comando 269  
índice composto  
    a criação de 400

índice composto (Continuação)  
 formato 341  
 conversão de arquivos nativos  
 ASCII 142  
 prazo de coordenação, consulta 79  
 Cozens, Simon 318  
 CPAN 318  
 rastreador 372  
 em SearchBlox 342  
 com XM-  
 InformationMinder 347  
 alternativas rastejando 330  
 CSS em destaque 301  
 Corte, Doug 9  
 trabalho relevante 9  
 CVS  
 obtenção de fonte do Lucene  
 código 391  
 Sandbox 268  
 CyberNeko. Ver NekoHTML  
 CzechAnalyzer 282

D

banco de dados de 8  
 indexação 362  
 chave primária 362  
 pesquisar 362  
 índice de armazenar dentro Berkeley DB 307  
 indexação data, 216  
 DateField 39  
 alternativas 218  
 questão 216  
 min e constantes max 173  
 consultas faixa 96  
 usado com DateFilter 173  
 DateFilter 171-173  
 cache 177  
 open-ended varia 172  
 com o cache 177  
 dentro ChainedFilter 306  
 DbDirectory 308  
 depuração, as consultas 94  
 DefaultSimilarity 79  
 documentos apagar 375  
 Digestor  
 configuração de 379  
 Directory 19  
 FSDirectory 19  
 RAMDirectory 19

diretório em Berkeley DB 308  
 DMOZ 27  
 DNA 354  
 Docco 265  
 DocSearcher 264  
 Documento de 20, 71  
 copy / paste de Lucas 274  
 edição com Lucas 275  
 campos heterogêneos 33  
 documento impulsionando 377  
 freqüência de documento visto com Lucas 273  
 manipulador de documento custo n zing para Ant 286  
 indexação com Ant 285  
 manipulação de tipo de documento em SearchBlox 342  
 documentação 398  
 dotLucene 317-318  
 download Lucene 388  
 Holandês 354  
 DutchAnalyzer 282

E

Egothor 24  
 codificação  
 ISO-8859-1 142  
 UTF-8 140  
 Étimo PJ 264  
 Explicação 80

F

Campo 20-22  
 anexando a 33  
 análise de palavra-chave, 121  
 armazenamento de vetores de termos 185  
 identificador de arquivo questão 340  
 Filtro de 76  
 cache 177  
 ChainedFilter 304  
 custom 209  
 usando HitCollector 203  
 dentro de um 212 Query  
 FilteredQuery 178, 212  
 filtragem  
 busca espaço 171-178  
 token. Ver TokenFilter  
 análise de língua estrangeira 140

Formatador 300  
 Fragmenter 300  
 FrenchAnalyzer 282  
 fuzzy string semelhança 351  
 FuzzyEnum 350  
 FuzzyQuery 92  
 a partir de 93 QueryParser 350 questões  
 problema de desempenho 213  
 proibindo 204

G

GCJ 308  
 Alemão análise 141  
 Giustina, Fabrizio 242  
 Glimpse 26  
 GNOME 318  
 Google 6, 27  
 palavra alternativa 128 sugestões  
 análise de 103  
 API 352  
 definições 292  
 custa 372  
 destacando prazo 300  
 de inteligência do governo, uso do Lucene 352

H

Colheita 26  
 Harvest NG-26  
 Harwood, Mark 300  
 destacando, termos de consulta 300-303, 343  
 Hindi 354  
 HitCollector 76, 201-203  
 personalizando 350  
 prioridade de fila de idéia 360  
 Filtros usados por 203  
 Hits 24, 70-71, 76  
 destacando 303  
 ht://Dig 26  
 TheServerSide uso 371  
 HTML 8  
 cookie 77  
 destacando 301  
 <meta> tag 140  
 análise de 107, 329, 352  
 HTMLParser 264

HTTP  
rastreador. Ver Nutch  
sessão 77  
Solicitação HTTP  
tipo de conteúdo-140

## Eu

I18N. Ver internacionalização  
índice de otimização 56-59  
Os requisitos de espaço em disco 56  
efeito de desempenho 56  
quando fazê-lo 58  
por que fazer isso 57  
estrutura do índice  
convertendo 400-401  
comparação de desempenho 402  
IndexFiles 389  
IndexHTML 390  
indexação  
documentos acrescentando 31-33  
análise, durante 105  
Ant tarefa 285  
em TheServerSide 373  
navegação ferramenta 271  
buffering 42  
365 cores  
formato composto 341  
índice composto 399-400  
regras de concorrência 59-60  
criação de 12  
estruturas de dados 11  
datas 39-40, 216  
depuração 66  
estrutura de diretório 395  
desativar o bloqueio 66  
404 formato de arquivo  
Ver arquivo com Lucas 277  
. Fnm arquivo 405  
para a classificação 41  
formato 393  
quadro 225-226, 254-263  
HTML 241, 248  
incremental 396  
arquivos de índice 397  
jGuru design 332  
limitando o tamanho do campo 54-55  
bloqueio 62-66  
visão lógica 394  
MaxFieldLength 54-55  
maxMergeDocs 42-47

mergeFactor 42-47  
índices de fusão 52  
Documentos do Microsoft Word  
248-251  
minMergeDocs 42, 47  
multifile estrutura do índice 395  
níumeros 40-41  
arquivos abertos 47-48  
parallelização 52-54  
PDF 235-241  
performance 42-47  
texto sem documentos  
253-254  
documentos remoção 33-36  
rich-text documentos 224  
RTF documentos 252-253  
agendamento de 367  
segmentos 396-397  
status com LIMO 279  
passos 29-31  
armazenamento em Berkeley DB 307  
termo do dicionário 406  
prazo de freqüência 406  
posições prazo 406  
thread-safety 60-62  
ferramentas 269  
documentos undeleting 36  
documentos atualização 36  
lotes 37  
usando RAMDirectory 48-52  
XML 226-235  
IndexReader 199  
documentos apagar 375  
recuperação de vetores de termos 186  
IndexSearcher 23, 70 e 78  
n-grama de extensão 358  
paginação através de resultados 77  
utilizando 75  
IndexWriter 19  
106 addDocument  
analisador de 123  
sobrecarga de informação 6  
Information Retrieval (IR) 7  
bibliotecas 24-26  
Instalando Lucene 387-392  
agente inteligente 6  
internacionalização 141  
freqüência inversa do documento 79  
404 índice invertido  
IR. Ver Information Retrieval (IR)  
ISO-8859-1 142

J  
Jakarta Commons Digester  
230-235  
Jakarta POI 249-250  
142 análise japonês  
Java Messaging Service 352  
em XM-  
InformationMinder 347  
Java, palavra-chave 331  
JavaCC 100  
construção de 392 Lucene  
JavaScript  
caráter escapando 292  
construção de consulta 291  
validação de consulta 291  
JDOM 264  
jGuru 341  
jGuruMultiSearcher 339  
Jones, Tim 150  
JPedal 264  
jSearch 7  
JTidy 242-245  
HTML com indexação  
Ant 285  
JUnitPerf 213  
JWordNet 297

K  
analisador de palavra-chave 124  
Konrad, Karsten 344  
Coreano análise 142

L  
linguagem  
manipulação de 354  
apoio 343  
LARM 7, 372  
Distância Levenshtein  
algoritmo 92  
definição de léxico, 331  
LIMO 279  
LingPipe 353  
lingüística 353  
Litchfield, Ben 236  
Lookout 6, 318  
Lucene  
construção da fonte 391  
10 comunidade

Lucene (Continuação)  
 aplicativos de demonstração 389-391  
 10 desenvolvedores  
 documentação 388  
 downloading 388  
 história, de 9 de índice 11  
 integração de 8  
 10 portas  
 aplicativo de amostra 11  
 Sandbox 268  
 compreensão seis usuários de 10  
 o que é de 7  
 Lucene portas 312-324  
 Resumo 313  
 Lucene Wiki 7  
 Lucene.Net 6  
 lucli 269  
 Lucas 271, 391  
 plug-ins-278  
 Lupy 308, 320-322

**M**

Gigabytes gestão 26  
 Matalon, Dror 269  
 Metaphone 125  
 MG4J 26  
 Michaels.com 361-371  
 Microsoft 6, 318  
 Microsoft Index Server 26  
 Microsoft Outlook 6, 318  
 Microsoft Windows 14  
 Microsoft Word 8  
 análise de 107  
 Miller, George 292  
 WordNet e 292  
 354 erros ortográficos  
 correspondência 363  
 mock objeto 131, 211  
 Moffat, Alistair 26  
 variação morfológica 355  
 Movable Type 320  
 MSN 6  
 MultiFieldQueryParser 160  
 Índice de vários arquivos, a criação de 398  
 vários índices 331  
 MultiSearcher 178-185  
 alternativas 339

em busca de vários segmentos.  
 Ver ParallelMultiSearcher  
 264 multivalentes

**N**

Namazu 26  
 native2ascii 142  
 linguagem natural com XM-InformationMinder 345  
 NekoHTML 245-248, 329, 352  
 .NET 10  
 n-gram TokenStream 357  
 NGramQuery 358  
 NGramSearcher 358  
 Nioche, Julien 279  
 noisy-channel modelo 355  
 normalização  
 comprimento do campo 79  
 consulta 79  
 numérico  
 padding 206  
 consultas gama 205  
 Nutch 7, 9, 329  
 Explicação 81

**O**

OLE 2 Compound Document formato 249  
 arquivos abertos fórmula 401  
 OpenOffice SDK 264  
 otimizar 340  
 variação ortográfica 354  
 Overture 6

**P**

paginação  
 em jGuru 336  
 TheServerSide pesquisa 383 resultados  
 através Hits 77  
 ParallelMultiSearcher 180  
 Parr, Terence 329  
 ParseException 204, 379  
 análise de 73  
 expressões de consulta.  
 Ver QueryParser  
 Método QueryParser 73  
 plurais stripping 334

versus análise de 107  
 índices de particionamento 180  
 PDF 8  
 Veja também PDF indexação  
 PDF Text 264 Stream  
 PDFBox 236-241  
 built-in suporte Lucene 239  
 PerFieldAnalyzerWrapper para 123 campos de palavra-chave execução

problemas com WildcardQuery 91  
 Hits iteração aviso 369  
 de teste de carga 217  
 de classificação 157  
 SearchBlox estudo de caso 341  
 estatísticas 370  
 213 testes, 220  
 Perl 10  
 farmacêutica, a utilização de Lucene 347  
 PhrasePrefixQuery 157-159  
 manipulação de sinônimos alternativas 134  
 PhraseQuery 87  
 em comparação com PhrasePrefixQuery 158  
 208 fim forçando prazo a partir de 90 QueryParser em contraste com SpanNearQuery 166  
 vários termos 89  
 emitir posição incremento 138  
 marcando 90  
 factor 139 slop com sinônimos 132  
 Piccolo 264  
 Plucene 318-320  
 POI 264

Porter algoritmo decorrentes 136  
 Porter, Dr. Martin 25, 136, 283  
 incremento de posição, deslocamento em SpanQuery 161  
 de precisão, 11 de 360  
 PrefixQuery 84  
 a partir de 85 QueryParser otimizado WildcardQuery 92  
 Propriedades do arquivo, codificação de 142  
 PyLucene 308, 322-323  
 Python 10

**Q**

Consulta 23, 70, 72  
criar programaticamente 81  
pré-processamento em jGuru 335  
começa com 84  
estatísticas 337  
`toString` 94  
Veja também `QueryParser`  
expressão de consulta, de análise.  
    Ver `QueryParser`  
`QueryFilter` 171, 173, 209  
alternativas utilizando  
    `BooleanQuery` 176  
como filtro de segurança 174  
dentro `ChainedFilter` 305  
`QueryHandler` 328  
consulta 70  
`QueryParser` 70, 72-74, 93  
análise de 106  
questões de análise 134  
analizador de escolha 107  
e `SpanQuery` 170  
consultas aumentar 99  
combinando com outro  
    Consulta 82  
combinando com programática  
    100 consultas  
criar `BooleanQuery` 87  
criar `FuzzyQuery` 93, 99  
criar `PhraseQuery` 90, 98  
criar `PrefixQuery` 85, 99  
criar `RangeQuery` 84  
criar `SpanNearQuery` 208  
criar `TermQuery` 83  
criar `WildcardQuery`  
    91, 99  
parsing de datas personalizado 218  
locale data de análise 97  
intervalos de datas 96  
operador por defeito 94  
caracteres de escape 93  
sintaxe da expressão 74  
estendendo 203-209  
seleção de campo 95  
expressões de agrupamento 95  
manipulação de intervalos numéricos 209  
questões 100, 107  
Campos de palavra-chave 122  
lowercasing curinga e  
    consultas prefixo 99

primordial para o sinônimo  
    injeção de 134  
`Questão PhraseQuery` 138  
proibindo caro  
    204 consultas  
consultas faixa 96  
`TheServerSide` personalizado  
    implementação 378  
Rápido, Andy 242

**R**

Raggett, Dave 242  
RAM, carregando índices em 77  
Arquivo, o carregamento `RAMDirectory`  
    índice para 77  
`RangeQuery` 83  
    a partir de 84 `QueryParser`  
manipulação de dados numéricos 205  
abrangendo múltiplas  
    índices 179  
escore bruto 78  
recall 11, 360  
expressões regulares.  
    Ver `WildcardQuery`  
banco de dados relacional. Ver banco de entre os documentos.  
dados  
relevância 76  
remoto pesquisar 180  
180 `RemoteSearchable`  
RGB indexação 366  
RMI, buscando através de 180  
Ruby 10  
Análise russa 141

**S**

`Sandbox` 268  
analisadores de 284  
componentes de construção 309  
`ChainedFilter` 177  
Highlighter 300  
`SAX` 352  
escalabilidade com `SearchBlox` 341  
205  
tuação 70, 77-78  
    normalização 78  
`ScoreDocComparator` 198  
Marcador 300  
marcando 78  
    afetados por `HitCollector` 203  
    fórmula 78  
rolagem. Ver paginação

pesquisa 68  
26 produtos  
recursos 27  
`Search Engine` 7  
    Ver `Nutch`; `SearchBlox`  
`SearchBlox` 7, 265-344  
`SearchClient` 182  
`SearchFiles` 389  
pesquisar 10  
    API 70  
resultados de filtragem 171-178  
para documentos similares 186  
índices em 180 paralela  
vários índices 178  
em vários campos 159  
`TheServerSide` 373  
usando `HitCollector` 201  
com Lucas 275  
`SearchServer` 180  
205  
`Searchtools` 27  
filtragem de segurança 174  
Selvaraj, Robert 341  
Resumindo, Allen 320  
termo de consulta similar.  
    Ver `FuzzyQuery`  
similaridade 80

entre os documentos.  
    Ver vetores prazo  
personalizando 350  
com XM-  
    `InformationMinder` 345  
`SimpleAnalyzer` 108, 119  
    exemplo 104  
`SimpleHTMLFormatter` 301  
`Simpy` 265  
poça  
    com `PhrasePrefixQuery` 159  
    com `SpanNearQuery` 166  
`Snowball` 25  
`SnowballAnalyzer` 282  
`SortComparatorSource`  
    195, 198  
`SortField` 200-201  
classificação  
    acessando valor personalizado 200  
    alfabeticamente 154  
    por um campo de 154  
    pela distância geográfica 195  
    pelo índice de ordem 153  
    por vários campos 155  
    152 por relevância

classificação (Continuação)  
 personalizado método 195-201  
 exemplo 150  
 tipo de campo 156  
 desempenho 157  
 revertendo 154  
 resultados da pesquisa 150-157  
 especificando locale 157  
**Soundex.** Ver *Metaphone*  
 código-fonte, *Sandbox* 268, 309  
**SpanFirstQuery** 162, 165  
**Espanhol** 354  
**SpanNearQuery** 99, 162, 166, 203, 208  
**SpanNotQuery** 162, 168  
**SpanOrQuery** 162, 169  
**SpanQuery** 161-170  
 agregando 169  
 e *QueryParser* 170  
 Utilitário de visualização de 164  
**SpanTermQuery** 162-165  
 correção ortográfica 354  
**Spencer**, Dave 293  
 alternativas spidering 330  
**SQL** 362  
 semelhanças com  
   *QueryParser* 72  
**StandardAnalyzer** 119-120  
 exemplo 104-105  
 com línguas asiáticas 143  
 com caracteres CJK 142, 145  
**estatística**  
 em *jGuru* 337  
 Michaels.com 370  
**Steinbach**, Ralf 344  
 decorrentes alternativas 359  
 decorrentes analisador 283  
**Stenzhorn**, Holger 344  
 parar 20 palavras, 103  
 em *jGuru* 335  
**StopAnalyzer** 119  
 exemplo 104  
**StringTemplate** 330  
**SubWordAnalyzer** 357  
**SWIG** 308  
**SWISH** 26  
**SWISH +** 26  
**SWISH-E** 26  
**SynonymEngine** 131  
 mock 132

**sinônimos**  
 analisador de injecão de 129  
 indexação 363  
 injetar com  
   *PhrasePrefixQuery* 159  
 com *PhraseQuery* 133  
 Veja também *WordNet*

**T**

**T9**, células interface de telefone 297  
**Tan**, Kelvin 291, 304  
**Prazo** 23  
**prazo**  
 definição de 103  
 navegação com Lucas 273  
 freqüência prazo de 79 anos, 331  
 ponderação 359  
**vetores prazo** 185-193  
 agregando 191  
 navegação com Lucas 275  
 ângulos de computação 192  
 computação arquétipo  
 documento 189  
**TermEnum** 198  
**TermFreqVector** 186  
**TermQuery** 24, 71, 82  
 contrastado com  
   *SpanTermQuery* 161  
 a partir de 83 *QueryParser*  
 com sinônimos 132  
**TextMining.org** 250-251  
**TheServerSide** 385  
**Arrumado.** Ver *JTidy*  
**Token** 108  
**TokenFilter** 109  
 282 adicionais  
 ordenação 116  
**tokenization**  
 definição de 103  
**tokenization.** Ver análise  
**Tokenizer** 109  
 282 adicionais  
 n-gram 357  
**tokens**  
 meta-dados de 109  
 offsets 116  
 incremento posição 109  
 incremento de posição no  
   *Nutch* 146  
**Tipo** 116, 127

posições visualizando 134  
**TokenStream** 107  
 arquitetura 110  
 para destacar 300  
**Tomcat**  
 demo aplicação 390  
**ferramenta**  
 de linha de comando  
   interface 269  
**Lucene Índice de Monitor** 279  
 Lucas 271  
**TopDocs** 200  
**TopFieldDocs** 200  
 transliteração 355, 359  
 solução de problemas 392

**U**

**UbiCrawler** 26  
**Unicode** 140  
**UNIX** 17  
 interface do usuário 6  
**UTF-8** 140

**V**

**Vajda**, Andi 308, 322  
**van Klinken**, Ben 314  
**vector.** Ver vetores prazo  
**Verity** 26  
**visualização**  
 com XM-  
   InformationMinder 346

**W**

**Paredes**, Craig 361  
**aplicação web**  
 CSS destacando 301  
 demo 390  
 JavaScript 290  
**LIMO** 279  
 Michaels.com 367  
**TheServerSide exemplo** 383  
**web crawler** 7  
 alternativas 330  
 Veja também rastreador  
**WebGlimpse** 26  
**WebStart**, índice Lucene  
 Caixa de ferramentas 272  
 ponderação, n-360 gramas

WhitespaceAnalyzer 119  
exemplo 104  
WildcardQuery 90  
a partir de 91 QueryParser  
problema de desempenho 213  
proibindo 204  
Witten, Ian H. 26  
WordNet 292-300  
WordNetSynonymEngine 297

---

**X**

Xapian 25

Omega 25  
xargs 17  
227-230 Xerces  
Xerces Native Interface  
(XNI) 245  
XM-InformationMinder  
344-350  
XML  
configuração de 380  
codificação de 140  
análise de 107  
resultados da pesquisa 343  
Xpdf 264

XSL  
transformando pesquisa  
343 resultados

---

**Y**

Yahoo! 6

---

**Z**

Zilverline 7

# Lucene EM

ACÃO  
Otis Gospodnetic • Erik Hatcher

PREFÁCIO Doug Cutting

**L**ucene é uma jóia no mundo open-source-a altamente escalável, rápido mecanismo de busca. Ele oferece desempenho e é surpreendentemente fácil de uso. Lucene in Action é o guia autorizado para Lucene. Ele descreve como índice de seus dados, incluindo tipos que você definitivamente precisa saber como o MS Word, PDF, HTML e XML. Introduz-lhe pesquisa, classificação, filtragem e destacando os resultados da pesquisa.

Lucene pesquisa poderes em lugares surpreendentes em grupos de discussão na Empresas da Fortune 100, em rastreadores de problemas comerciais, em busca de e-mail

da Microsoft, no motor de pesquisa Nutch web (que pode chegar a bilhões de páginas). Ele é usado por diversas empresas, incluindo Akamai, Overture, Technorati, HotJobs, Epiphany, FedEx, Mayo Clinic, MIT, Revista New Scientist, e muitos outros.

Adicionando pesquisa para a sua aplicação pode ser fácil. Com muitos reutilizáveis exemplos e bons conselhos sobre as melhores práticas, Lucene in Action mostra como fazer isso.

## O que há dentro

- Eu Como integrar o Lucene em suas aplicações
- Eu Ready-to-use estrutura para manipulação de documentos ricos
- Eu Estudos de caso, incluindo Nutch, TheServerSide, jGuru, etc
- Eu Lucene portas para Perl, Python, C #/.Net e C++
- Eu Classificação, filtragem, vetores prazo, múltiplas, e índice de controle remoto em busca
- Eu A família SpanQuery novo analisador de consulta estendendo, hit coleta
- Eu Testes de desempenho e tuning
- Eu Lucene add-ons (destacando hit, sinônimo de pesquisa, e outros)
- Eu

Um membro do Ant, Lucene, e open-source Tapestry projetos, Erik Hatcher é co-autor de Manning é premiado Java Development com Ant. Otis Gospodnetic é um committer Lucene, um membro da Apache Jakarta Project Management Committee, e mantenedor do de jGuru Lucene FAQ. Ambos os autores têm publicado inúmeros artigos técnicos incluindo vários em Lucene.

"... Cheia de exemplos e conselhos sobre como efetivamente utilizar esta incrivelmente poderosa ferramenta."

-Brian Goetz  
Consultor Principal,  
Quiotix Corporation

"... É desbloqueado para mim o incrível poder de Lucene."

-Reece Wilton, Engenheiro de Pessoal,  
Walt Disney Internet Group

"... Os exemplos de código são úteis e reutilizáveis."

-Scott Ganya  
Jakarta Lucene committer

"... Amostras de código como teste JUnit casos são incrivelmente útil".

-Norman Richards, co-autor  
XDoclet in Action

AUTOR  
ONLINE  
Pergunte ao Autor

Ebook edição

[www.manning.com / nascedouro](http://www.manning.com/nascedouro)

, 7IB9D2-  
djecid:! P; o;  
O; t; P



MANNING

\$ 44,95 EUA / Canadá \$ 60,95