



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Apache Solr Enterprise Search Server

Third Edition

Enhance your searches with faceted navigation, result highlighting, relevancy-ranked sorting, and much more with this comprehensive guide to Apache Solr 4

David Smiley
Kranti Parisa

Eric Pugh
Matt Mitchell

[PACKT] open source*
PUBLISHING community experience distilled

Apache Solr Enterprise Search Server

Third Edition

Enhance your searches with faceted navigation, result highlighting, relevancy-ranked sorting, and much more with this comprehensive guide to Apache Solr 4

David Smiley

Eric Pugh

Kranti Parisa

Matt Mitchell



BIRMINGHAM - MUMBAI

Apache Solr Enterprise Search Server

Third Edition

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: August 2009

Second edition: November 2011

Third edition: May 2015

Production reference: 1200515

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-136-3

www.packtpub.com

Cover image by Sylvia Smiley (sylvie.zajac@gmail.com)

Credits

Authors

David Smiley

Eric Pugh

Kranti Parisa

Matt Mitchell

Reviewers

Edd Grant

Aamir Hussain

Dmitry Kan

Acquisition Editors

Nikhil Karkal

Rebecca Youé

Content Development Editor

Shubhangi Dhamgaye

Technical Editor

Pankaj Kadam

Copy Editors

Puja Lalwani

Laxmi Subramanian

Project Coordinator

Harshal Ved

Proofreaders

Stephen Copestake

Safis Editing

Indexer

Tejal Soni

Graphics

Jason Monteiro

Production Coordinator

Manu Joseph

Cover Work

Manu Joseph

About the Authors

Born to code, **David Smiley** is a software engineer who's passionate about search, Lucene, spatial, and open source. He has a great deal of expertise with Lucene and Solr, which started in 2008 at MITRE. In 2009, as the lead author, along with the coauthor Eric Pugh, he wrote *Solr 1.4 Enterprise Search Server*, the first book on Solr, published by Packt Publishing. It was updated in 2011, *Apache Solr 3 Enterprise Search Server*, Packt Publishing, and again for this third edition.

After the first book, he developed 1- and 2-day Solr training courses, delivered half a dozen times within MITRE, and he has also delivered training on LucidWorks once. Most of his excitement and energy relating to Lucene is centered on Lucene's spatial module to include Spatial4j, which he is largely responsible for. He has presented his progress on this at Lucene Revolution and other conferences several times. He currently holds the status of committer & Project Management Committee (PMC) member with the Lucene/Solr open source project. Over the years, David has staked his career on search, working exclusively on such projects, formerly for MITRE and now as an independent consultant for various clients. You can reach him at dsmiley@apache.org and view his LinkedIn profile here: <http://www.linkedin.com/in/davidwsmiley>.

Writing a book is the biggest project I've ever worked on, where I've put time outside of employed working hours; it requires an inordinate amount of time. I have a great deal of respect for the other authors who undertake such projects.

I'm deeply appreciative of my wife, Sylvie, for enduring this time commitment and for taking care of our young daughters, Camille and Adeline. I also thank my coauthors, Eric, Kranti, and Matt, for their share of this herculean effort—it is too much for anyone of us. And finally, I am most appreciative of the compliments I've received from readers about the previous editions. It helps make the effort worthwhile.

Fascinated by the "craft" of software development, **Eric Pugh** has been involved in the open source world as a developer, committer, and user for the past decade. He is an emeritus member of the Apache Software Foundation.

In biotech, financial services, and defense IT, he has helped European and American companies develop coherent strategies to embrace open source software. As a speaker, he has advocated the advantages of Agile practices in search, discovery, and analytics projects.

Eric became involved in Solr when he submitted the patch SOLR-284 to parse rich document types, such as PDF and MS Office formats, that became the single-most popular patch, as measured by votes! The patch was subsequently cleaned up and enhanced by three other individuals, demonstrating the power of the free / open source models to build great code collaboratively. SOLR-284 was eventually refactored into Solr Cell.

He blogs at <http://www.opensourceconnections.com/blog/>.

Someone once told me the best business card you can have is a book, and I've discovered over the past three editions of this book that this is a very true thing. Of course, I've also learned that printing out 500 business cards is a much easier job than updating a book. Additionally, we wanted some fresh voices in this edition of the book. I'd like to thank Matt and Kranti for jumping feet first into this project. I'd like to thank David for putting so much passion into this book and being our fearless leader through all the twists and turns.

I also want to thank Erik Hatcher once again for his continuing support and mentorship over the past 10 years. Without his encouragement, I wouldn't have spoken at Euro Lucene or become involved in the Solr community.

I also want to thank all of my colleagues at OpenSource Connections. We've come a long way as a company, and I'm excited about SlopBucket, our very own conference this fall! Matt Overstreet, your creativity that you contributed to the content on SolrCloud was critical to my finishing Chapter 10.

My darling wife, Kate, who says, "It can't be that hard to finish, just do it!", I know life continues to be busy, but I couldn't be happier sharing my life with you, Morgan, and Asher. I love you.

Lastly, I want to thank all the adopters of Solr and Lucene! Without you, I wouldn't have this wonderful open source project to be so incredibly proud of! I look forward to meeting more of you at the next conference.

Kranti Parisa has more than a decade of software development expertise and a deep understanding of open source, enterprise software, and the execution required to build successful products.

He has fallen in love with enterprise search technologies, especially Lucene and Solr, after his initial implementations and customizations carried out in early 2008 to build a legal search engine for bankruptcy court documents, docket entries, and cases. He is an active contributor to the Apache Solr community. One of his recent contributions, along with Joel Bernstein, SOLR-4787, includes scalable and nested join implementations.

Kranti is currently working at Apple. Prior to that, he worked as a lead engineer and search architect at Comcast Labs, building and supporting a highly scalable search and discovery engine for the X1/X2 platform—the world's first entertainment operating system.

An entrepreneur by DNA, he is the cofounder and technical advisor of multiple start-ups focusing on cloud computing, SaaS, big data, and enterprise search based products and services. He holds a master's degree in computer integrated manufacturing from the National Institute of Technology, Warangal, India.

You can reach him on LinkedIn: <http://www.linkedin.com/in/krantiparisa>.

First and foremost, many thanks to Albert Einstein for his extraordinary innovations and one of his many inspirational quotes: *I have no special talent. I am only passionately curious.* I'd like to thank and acknowledge all the contributors to the Apache Lucene and Solr projects. You're totally awesome! Completing this work would have been all the more difficult were it not for the support and friendship provided by my coauthors—David, Eric, and Matt. I am indebted to them for their help.

Thanks to my family and friends for believing in me; I couldn't have done this without you. A very special thanks to my darling mother, Nagarani, and my beautiful wife, Pallavi, for their unconditional love, support, and patience as I spent countless weekends working on this book.

And, of course, I want to thank the team at Packt Publishing for their tremendous support in all ways, large and small. There are many more people I would like to thank, but time, space, and modesty compel me to stop here.

Matt Mitchell studied music synthesis and performance at Boston's Berklee College of Music, but his experiences with computers and programming in his younger years inspired him to pursue a career in software engineering. A passionate technologist, he has worked in many areas of software development, is active in several open source communities, and now has over 15 years of professional experience. He had his first experiences with Lucene and Solr in 2008 at the University of Virginia Library, where he became a core contributor to an open source search platform called Backlight. Matt is the author of many open source projects, including a Solr client library called RSolr, which has had over 1 million downloads from `rubygems.org`. He has been responsible for the design and implementation of search systems at several tech companies, and he is currently a senior member of the engineering team at LucidWorks, where he's working on a next generation search, discovery, and analytics platform.

You can contact Matt on LinkedIn at <https://www.linkedin.com/in/mattmitchell4>.

I'd like to thank my amazing wife, Jenny, and our kids, Henry and Dorothy, for their unbelievable patience and support during this journey. My parents, thank you for making my fantastic life possible. Eric, Kranti, and David, for all the blood, sweat, and tears you've put into this book, along with all the time you've spent helping me. My good friend, Anthony Fox, who never seems to stop encouraging and inspiring me. Erik Hatcher and Bess Sadler, for getting me started with all of this search stuff in the first place. The Lucene and Solr communities and committers, for all of their amazing work. Packt Publishing, for their endless patience and exceptional guidance. And, of course, the readers and reviewers of this book — thank you all!

About the Reviewers

Edd Grant is a freelance software engineer who has been building software professionally since 2003. He is passionate about designing first-class, maintainable systems by leveraging agile and TDD principles and has helped his clients adopt and excel at these practices.

Edd is an experienced implementor of cloud-scale web applications and services, continuous delivery, and infrastructure automation. As an open source advocate, he has helped many clients take advantage of a diverse range of such products.

Edd has a website, which he updates when he gets the time (<http://www.eddgrant.com>), and has a passion for mountain biking and tea.

Aamir Hussain is an experienced customer- and business-focused technology leader with rich hands-on engineering, business and management experience. He has over 6 years of experience in software engineering and complex systems design with focus on the Cloud software architecture and design, Software as a Service (SaaS), Platform as a Service (PaaS), monitoring and tools infrastructure, network design, and data center operations.

Starting the journey of his career from the world's most disturbed and heavily militarized zone, Aamir had also been honored and awarded multiple times in the application development programs conducted by Health2con and WHO in USA. He is currently working with one of India's largest e-commerce logistics company (Delhivery) as a senior architect.

Aamir had also managed to get his name on multiple books of Apache Solr and Python published by Packt Publishing.

Dmitry Kan leads the search technology development at AlphaSense, the one-stop financial search engine company. In parallel, he is the founder and CEO of the language intelligence company SemanticAnalyzer. Dmitry enjoys building and blogging about software, in particular, search (Solr/Lucene), machine learning (sentiment detection and machine translation), and tools that make a programmer's life easier. You can find his blogs at dmitrykan.blogspot.com and semanticanalyzer.info/blog. He developed his fully blown search engine back in 2003 as a university project. The main achievements were beating MySQL full-text search engine in speed by over 5 million records. This is when he introduced himself to the world of skip lists and balanced hash tables. In 2010, Dmitry learned about Lucene and Solr, and since then, he has been an active community member, occasionally taking part (and winning!) in the famous Stump the Chump sessions. Dmitry holds a PhD in CS from the Saint Petersburg State University (Russia) and a master's degree in CS from the University of Kuopio (Finland). In his free time, Dmitry enjoys answering questions on Stack Overflow, building models on kaggle, and cycling.

He is the maintainer and developer of *Lucene Luke*, which can be found at <https://github.com/dmitrykey/luke>. You can reach him on Twitter at twitter.com/dmitrykan.

I am immensely grateful to my parents for giving me the support and hunger for knowledge. My wife, Tatiana, is my first kind listener to all the ideas I get around IT, apart from being a loving and supporting wife. She knows how hard it is to be around a programming geek like me. Big thanks to all Luke fans, you help me learn new things around Apache Lucene and search in general.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	xi
Chapter 1: Quick Starting Solr	1
An introduction to Solr	1
Lucene – the underlying engine	2
Solr – a Lucene-based search server	3
Comparison to database technology	4
A few differences between Solr 4 and Solr 5	6
Getting started	6
Solr's installation directory structure	7
Running Solr	10
A quick tour of Solr	10
Loading sample data	14
A simple query	15
Some statistics	18
The sample browse interface	19
Configuration files	20
What's next?	22
Schema design and indexing	22
Text analysis	23
Searching	24
Integration	25
Resources outside this book	25
Summary	26

Chapter 2: Schema Design	27
Is Solr schemaless?	28
MusicBrainz.org	29
One combined index or separate indices	30
One combined index	31
Problems with using a single combined index	32
Separate indices	33
Schema design	33
Step 1 – determine which searches are going to be powered by Solr	34
Step 2 – determine the entities returned from each search	35
Step 3 – denormalize related data	35
Denormalizing – one-to-one associated data	36
Denormalizing – one-to-many associated data	36
Step 4 – omit the inclusion of fields only used in search results (optional)	38
The schema.xml file	39
Field definitions	40
Dynamic field definitions	41
Advanced field options for indexed fields	41
The unique key	43
The default search field and query operator	43
Copying fields	44
Our MusicBrainz field definitions	44
Defining field types	47
Built-in field type classes	48
Numbers and dates	48
Some other field types	49
Summary	50
Chapter 3: Text Analysis	51
Configuring field types	52
Experimenting with text analysis	55
Character filters	57
Tokenization	58
Filtering	60
Stemming	62
Correcting and augmenting stemming	63
Processing synonyms	64
Synonym expansion at index time versus query time	65
Working with stop words	66
Phonetic analysis	67
Substring indexing and wildcards	69
ReversedWildcardFilter	70

N-gram analysis	70
N-gram costs	71
Sorting text	72
Miscellaneous token filters	73
The multilingual search	75
The multifield approach	75
The multicore approach	76
The single field approach	76
Summary	77
Chapter 4: Indexing Data	79
Communicating with Solr	80
Using direct HTTP or a convenient client API	80
Pushing data to Solr or have Solr pull it	81
Data formats	81
Solr's HTTP POST options	82
Remote streaming	84
Solr's Update-XML format	85
Deleting documents	86
Commit, optimize, and rollback the transaction log	86
Don't overlap commits	88
Index optimization	88
Rolling back an uncommitted change	89
The transaction log	89
Atomic updates and optimistic concurrency	90
Sending CSV-formatted data to Solr	91
Configuration options	93
The DataImportHandler framework	94
Configuring the DataImportHandler framework	95
The development console	96
Writing a DIH configuration file	97
Data sources	97
Entity processors	98
Fields and transformers	99
Example DIH configurations	101
Importing from databases	101
Importing XML from a file with XSLT	103
Importing multiple rich document files – crawling	104
Importing commands	105
Delta imports	106
Indexing documents with Solr Cell	107
Extracting text and metadata from files	107
Configuring Solr	108

Solr Cell parameters	109
Update request processors	111
Summary	115
Chapter 5: Searching	117
Your first search – a walk-through	118
A note on response format types	120
Solr's generic XML structured data representation	121
Solr's XML response format	121
Parsing the URL	122
Understanding request handlers	123
Query parameters	125
Search criteria related parameters	126
Result pagination related parameters	126
Output-related parameters	127
More about the fl parameter	127
Diagnostic parameters	129
Query parsers and local-params	130
Query syntax (the lucene query parser)	131
Matching all the documents	132
Mandatory, prohibited, and optional clauses	132
Boolean operators	133
Subqueries	134
Limitations of prohibited clauses in subqueries	135
Querying specific fields	136
Phrase queries and term proximity	136
Wildcard queries	137
Fuzzy queries	138
Regular expression queries	139
Range queries	139
Date math	140
Score boosting	141
Existence and nonexistence queries	141
Escaping special characters	142
The DisMax query parser – part 1	143
Searching multiple fields	144
Limited query syntax	145
Min-should-match	146
Basic rules	146
Multiple rules	147
What to choose	147
A default query	148
The uf parameter	148

Filtering	149
Sorting	150
Joining	151
The join query parser	151
Block-join query parsers	153
The block-join-children parser	154
The block-join-parent parser	154
Spatial search	155
Spatial in Solr 3 – LatLonType and friends	156
Configuration	157
Spatial in Solr 4 – SpatialRecursivePrefixTreeFieldType	157
Configuration – basic	159
Indexing points	160
Filtering by distance or rectangle	161
Sorting by distance	162
Returning the distance	163
Boosting by distance	163
Memory and performance of distance sorting and boosting	163
Advanced spatial	164
Summary	165
Chapter 6: Search Relevancy	167
Scoring	167
Alternative scoring models	169
Query-time and index-time boosting	170
Troubleshooting queries and scoring	170
Tools – Splainer and Quepid	172
The DisMax query parser – part 2	173
Lucene's DisjunctionMaxQuery	173
Boosting – automatic phrase boosting	174
Configuring automatic phrase boosting	174
Phrase slop configuration	175
Partial phrase boosting	175
Boosting – boost queries	176
Boosting – boost functions	177
Add or multiply boosts	178
Functions and function queries	179
Field references	180
Function references	181
Mathematical primitives	182
Other math	182
Boolean functions	183
Relevancy statistics functions	183
Ord and rord	184

Table of Contents

Miscellaneous functions	184
External field values	185
Function query boosting	186
Formula – logarithm	186
Formula – inverse reciprocal	188
Formula – reciprocal	189
Formula – linear	190
How to boost based on an increasing numeric field	190
Step by step...	191
How to boost based on recent dates	192
Step by step...	193
Summary	194
Chapter 7: Faceting	195
A quick example – faceting release types	196
Field requirements	197
Types of faceting	198
Faceting field values	198
Alphabetic range bucketing	200
Faceting numeric and date ranges	202
Range facet parameters	204
Facet queries	206
Building a filter query from a facet	207
Field value filter queries	208
Facet range filter queries	208
Pivot faceting	209
Hierarchical faceting	211
Excluding filters – multiselect faceting	212
Summary	215
Chapter 8: Search Components	217
About components	218
The highlight component	220
A highlighting example	220
Choose the Standard, FastVector, or Postings highlighter	222
The Standard (default) highlighter	222
The FastVector highlighter	223
The Postings highlighter	223
Highlighting configuration	224
The SpellCheck component	225
The schema configuration	227
Configuration in solrconfig.xml	228
Configuring spellcheckers – dictionaries	229
Processing the q parameter	233

Processing the spellcheck.q parameter	234
Building index- and file-based spellcheckers	234
Issuing spellcheck requests	235
Example usage for a misspelled query	238
Query complete/suggest	240
Instant-search via edge n-grams	242
Query term completion via facet.prefix	243
Query term completion via the Suggester	245
Query term completion via the Terms component	248
Field-value completion via the Suggester	248
The QueryElevation component	250
Configuration	251
The MoreLikeThis component	252
Configuration parameters	253
Parameters specific to the MLT search component	254
Parameters specific to the MLT request handler	254
Common MLT parameters	255
The MLT results example	257
The Stats component	259
Configuring the stats component	260
Statistics on track durations	260
The Clustering component	261
Collapsing and expanding	262
The Collapse query parser	262
The Expand component	263
An example	263
Compared to Result grouping	265
The TermVector component	267
Summary	267
Chapter 9: Integrating Solr	269
Working with the included examples	270
Inventory of examples	270
Solritas – the integrated search UI	271
The pros and cons of Solritas	273
SolrJ – Solr's Java client API	274
The sample code – BrainzSolrClient	275
Dependencies and Maven	275
Declaring logging dependencies	276
The SolrServer class	277
Using javabin instead of XML for efficiency	278
Searching with SolrJ	278

Indexing with SolrJ	279
Deleting documents	280
Annotating your JavaBean – an alternative	280
Embedding Solr	281
When should you use embedded Solr? Tests!	282
Using JavaScript/AJAX with Solr	283
Wait, what about security?	285
Building a Solr-powered artists autocomplete widget with jQuery and JSONP	285
AJAX Solr	289
Using XSLT to transform XML search results	291
Accessing Solr from PHP applications	292
solr-php-client	293
Drupal options	295
The Apache Solr Search integration module	295
Hosted Solr by Acquia	296
Ruby on Rails integrations	297
Solr's Ruby response writer	297
The sunspot_rails gem	298
Setting up the myFaves project	298
Populating the myFaves relational database from Solr	299
Building Solr indexes from a relational database	301
Completing the myFaves website	303
Which Rails/Ruby library should I use?	305
Nutch for crawling web pages	306
Solr and Hadoop	308
HDFS	308
Indexing via MapReduce	309
Morphlines	310
Running a Solr build using Hadoop	310
Looking at the storage	310
The data ingestion process	313
ManifoldCF – a connector framework	315
Connectors	316
Putting ManifoldCF to use	316
Document-level security	318
Summary	319
Chapter 10: Scaling Solr	321
Tuning complex systems is hard	322
Use SolrMeter to test Solr performance	323
Optimizing a single Solr server – scale up	325
Configuring JVM settings to improve memory usage	326
Using MMapDirectoryFactory to leverage additional virtual memory	327

Enabling downstream HTTP caching to reduce load	327
Solr caching	329
Tuning caches	331
Indexing performance	332
Designing the schema	332
Sending data to Solr in bulk	333
Disabling unique key checking	333
Index optimization and mergeFactor settings	334
Enhancing faceting performance	335
Using term vectors	335
Improving phrase search performance	336
Configuring Solr for near real-time search	338
Use SolrCloud to go big – scale wide	339
SolrCloud glossary	341
Launching Solr in SolrCloud mode	342
Managing collections and configurations	343
Stand up SolrCloud for our MusicBrainz artists index	344
Choosing the replication factor and number of shards	346
Creating and deleting collections	348
Replicas and leaders	349
Document routing	349
Shard splitting	350
Dealing with long running collection tasks	351
Adding nodes	352
Summary	352
Chapter 11: Deployment	353
Deployment methodology for Solr	353
Questions to ask	354
Installing Solr into a Servlet container	355
Differences between Servlet containers	355
Defining the solr.home property	356
Configuring logging	357
HTTP server request access logs	358
Solr application logging	359
Configuring logging output	360
Jetty startup integration	361
Managing log levels at runtime	361
A RequestHandler per search interface	362
Leveraging Solr cores	363
Configuring solr.xml	364
Property substitution	366
Include fragments of XML with XInclude	366
Managing cores	367
Some uses of multiple cores	368

Setting up ZooKeeper for SolrCloud	369
Installing ZooKeeper	370
Administering Data in ZooKeeper	371
Monitoring Solr performance	372
Stats Admin interface	372
Monitoring Solr via JMX	374
Starting Solr with JMX	375
Securing Solr from prying eyes	376
Limiting server access	376
Put Solr behind a Proxy	379
Securing public searches	379
Controlling JMX access	380
Securing index data	380
Controlling document access	380
Other things to look at	381
Summary	382
Appendix: Quick Reference	383
Core search	383
Diagnostic	384
The Lucene query parser	384
The DisMax query parser	384
The Lucene query syntax	385
Faceting	385
Highlighting	386
Spell checking	386
Miscellaneous nonsearch	386
Index	387

Preface

If you are a developer building an application today, then you know how important a good search experience is. Apache Solr, built on Apache Lucene, is a wildly popular open source enterprise search server that easily delivers the powerful search and faceted navigation features that are elusive with databases. Solr supports complex search criteria, faceting, result highlighting, query-completion, query spellcheck, relevancy tuning, and more.

Apache Solr Enterprise Search Server, Third Edition is a comprehensive resource to almost everything Solr has to offer. It serves the reader right from initiation to development to deployment. It also comes with complete running examples to demonstrate its use and show how to integrate Solr with other languages and frameworks—even Hadoop.

By using a large set of metadata, including artists, releases, and tracks, courtesy of the MusicBrainz.org project, you will have a testing ground for Solr and will learn how to import this data in various ways. You will then learn how to search this data in different ways, including Solr's rich query syntax and boosting match scores based on record data. Finally, we'll cover various deployment considerations to include indexing strategies and performance-oriented configuration that will enable you to scale Solr to meet the needs of a high-volume site.



Solr 4 or Solr 5?

See the *What you need for this book* section further below.

What this book covers

Chapter 1, Quick Starting Solr, introduces Solr to you so that you understand its unique role in your application stack. You'll get started quickly by indexing example data and searching it with Solr's sample / browse UI. This chapter is oriented to Solr 5, but the majority of content applies to Solr 4 too.

Chapter 2, Schema Design, guides you through an approach to modeling your data within Solr into one or more Solr indices and schemas. It covers the schema thoroughly and explores some of Solr's field types.

Chapter 3, Text Analysis, covers how to customize text tokenization, stemming, synonyms, and related matters to have fine control over keyword search matching. It also covers multilingual strategies.

Chapter 4, Indexing Data, explores all of the options Solr offers for importing data, such as XML, CSV, databases (SQL), and text extraction from common documents. This includes important information on commits, atomic updates, and real-time search.

Chapter 5, Searching, covers the query syntax, from the basics to Boolean options to more advanced wildcard and fuzzy searches, join queries, and geospatial search.

Chapter 6, Search Relevancy, explains how Solr scores documents for relevancy ranking. We'll review different options to influence the score, called boosting, and apply it to common examples such as boosting recent documents and boosting by a user vote.

Chapter 7, Faceting, shows you how to use Solr's killer feature — faceting. You'll learn about the different types of facets and how to build filter queries for a faceted navigation interface.

Chapter 8, Search Components, explores how to use a variety of valuable search features implemented as Solr search components. This includes result highlighting, query spellcheck, query suggest / complete, result grouping / collapsing, and more.

Chapter 9, Integrating Solr, explores some external integration options to interface with Solr. This includes some language-specific frameworks for Java, JavaScript, Ruby, and PHP, as well as a web crawler, Hadoop, a quick prototyping option, and more.

Chapter 10, Scaling Solr, covers how to tune Solr to get the most out of it. Then we'll introduce how to scale beyond one instance with SolrCloud.

Chapter 11, Deployment, guides you through deployment considerations to include deploying Solr to Apache Tomcat, to logging, and to security, and setting up Apache ZooKeeper.

Appendix, Quick Reference, serves as a small parameter quick-reference guide you can print to have within reach when you need it.

What you need for this book

The *Getting started* section in *Chapter 1, Quick Starting Solr*, explains what you need in detail. In summary, you should obtain:

- Java 8, a JDK release. Java 7 is fine too. Support for Java 6 was last available in Solr 4.7. More information on this is in *Chapter 1, Quick Starting Solr*.
- Apache Solr 4.8.1 is officially the version of Solr this book was written for. Nonetheless, some of the features are discussed or referenced in the later versions of Solr as far as 5.0. In fact, *Chapter 1, Quick Starting Solr*, orients you to Solr 5, which has a different first-impression experience than its predecessor. Once you get Solr running, you should be able to follow along easily with Solr 5. In *Chapter 10, Scaling Solr*, there are some SolrCloud startup commands that are a little different, and we've pointed out how they change. The only substantial topic not covered in this book that evolved through the Solr 4 point releases is data-driven schemaless mode, and HTTP API calls to make schema changes.
- The code supplement to the book. It's not essential, but you'll want it to try some of the examples or to experiment with a sizable amount of real data. See the *Downloading the example code* section.

Who this book is for

This book is primarily for developers who want to learn how to use Apache Solr in their applications. Only basic programming skills are assumed, although the vast majority of content should be useful to those with a solid technical foundation that have not yet programmed.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Typing `java -version` at a command line will tell you exactly which version of Java you are using, if any."

A block of code is set as follows:

```
"responseHeader": {  
    "status": 0,
```



```
    "QTime": 1,
    "params": {
      "q": "lcd",
      "indent": "true",
      "wt": "json"
    }
  }
  ...
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:


```
{
  "id": "9885A004",
  "name": "Canon PowerShot SD500",
  "manu": "Canon Inc.",
  "manu_id_s": "canon",
  "cat": [
    "electronics",
    "camera"
  ],
  "features": [
    "3x zoop, 7.1 megapixel Digital ELPH",
    "movie clips up to 640x480 @30 fps",
    "2.0\" TFT LCD, 118,000 pixels",
    "built in flash, red-eye reduction"
  ],
  "includes": "32MB SD card, USB cable, AV cable, battery",
  "weight": 6.4,
  "price": 329.95,
  "price_c": "329.95,USD",
  "popularity": 7,
  "inStock": true,
  "manufacturedate_dt": "2006-02-13T15:26:37Z",
  "store": "45.19614,-93.90341",
  "_version_": 1500358264225792000
},
...
```


Any command-line input or output is written as follows:

```
>> cd example/exampledocs
>> java -Dc=techproducts -jar post.jar *.xml
SimplePostTool version 5.0.0
Posting files to [base] url
http://localhost:8983/solr/techproducts/update using
content-type application/xml...
POSTing file gb18030-example.xml
```

```
POSTing file hd.xml
etc.
14 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/techproducts/
update...
```

New terms and important words are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Click on the **Core Selector** drop-down menu and select the **techproducts** link."

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

A copy of the code bundle and possibly other information will also be available at <http://www.solrenterprisesearchserver.com>.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Quick Starting Solr

Welcome to Solr! You've made an excellent choice to power your search needs. In this chapter, we're going to cover the following topics:

- An overview of what Solr and Lucene are all about
- What makes Solr different from databases?
- How to get Solr, what's included, and what is where?
- Running Solr and importing sample data
- A quick tour of the admin interface and key configuration files
- A brief guide on how to get started quickly

An introduction to Solr

Solr is an open source enterprise search server. It is a mature product powering search for public sites such as CNET, Yelp, Zappos, and Netflix, as well as countless other government and corporate intranet sites. It is written in Java, and that language is used to further extend and modify Solr through various extension points.

However, being a server that communicates using standards such as HTTP, XML, and JSON, knowledge of Java is useful but not a requirement. In addition to the standard ability to return a list of search results based on a full text search, Solr has numerous other features such as result highlighting, faceted navigation (as seen on most e-commerce sites), query spellcheck, query completion, and a "more-like-this" feature for finding similar documents.



You will see many references in this book to the term **faceting**, also known as **faceted navigation**. It's a killer feature of Solr that most people have experienced at major e-commerce sites without realizing it. Faceting enhances search results with aggregated information over all of the documents found in the search. Faceting information is typically used as dynamic navigational filters, such as a product category, date and price groupings, and so on. Faceting can also be used to power analytics. *Chapter 7, Faceting*, is dedicated to this technology.

Lucene – the underlying engine

Before describing Solr, it is best to start with Apache Lucene, the core technology underlying it. Lucene is an open source, high-performance text search engine library. Lucene was developed and open sourced by Doug Cutting in 2000 and has evolved and matured since then with a strong online community. It is the most widely deployed search technology today. Being just a code library, Lucene is not a server and certainly isn't a web crawler either. This is an important fact. There aren't even any configuration files.

In order to use Lucene, you write your own search code using its API, starting with indexing documents that you supply to it. A **document** in Lucene is merely a collection of **fields**, which are name-value pairs containing text or numbers. You configure Lucene with a text **analyzer** that will **tokenize** a field's text from a single string into a series of **tokens** (words) and further transform them by reducing them to their stems, called **stemming**, substitute synonyms, and/or perform other processing. The final indexed tokens are said to be the **terms**. The aforementioned process starting with the analyzer is referred to as **text analysis**. Lucene *indexes* each document into its **index** stored on a disk. The index is an **inverted index**, which means it stores a mapping of a field's terms to associated documents, along with the ordinal word position from the original text. Finally, you search for documents with a user-provided query string that Lucene parses according to its syntax. Lucene assigns a numeric relevancy **score** to each matching document and only the top scoring documents are returned.



This brief description of Lucene internals is what makes Solr work at its core. You will see these important vocabulary words throughout this book—they will be explained further at appropriate times.

Lucene's major features are:

- An inverted index for efficient retrieval of documents by indexed terms. The same technology supports numeric data with range- and time-based queries too.
- A rich set of chainable text analysis components, such as tokenizers and language-specific stemmers that transform a text string into a series of terms (words).
- A query syntax with a parser and a variety of query types, from a simple term lookup to exotic fuzzy matching.
- A good scoring algorithm based on sound **Information Retrieval (IR)** principles to produce the best matches first, with flexible means to affect the scoring.
- Search enhancing features. There are many, but here are some notable ones:
 - A highlighter feature to show matching query terms found in context.
 - A query spellchecker based on indexed content or a supplied dictionary.
 - Multiple suggesters for completing query strings.
 - Analysis components for various languages, faceting, spatial-search, and grouping and joining queries too.



To learn more about Lucene, read *Lucene In Action, Second Edition*, Michael McCandless, Erik Hatcher, and Otis Gospodneti, Manning Publications.

Solr – a Lucene-based search server

Apache Solr is an enterprise search server that is based on Lucene. Lucene is such a big part of what defines Solr that you'll see many references to Lucene directly throughout this book. Developing a high-performance, feature-rich application that uses Lucene directly is difficult and it's limited to Java applications. Solr solves this by exposing the wealth of power in Lucene via configuration files and HTTP parameters, while adding some features of its own. Some of Solr's most notable features beyond Lucene are as follows:

- A server that communicates over HTTP via multiple formats, including XML and JSON
- Configuration files, most notably for the index's schema, which defines the fields and configuration of their text analysis

- Several types of caches for faster search responses
- A web-based administrative interface, including the following:
 - Runtime search and cache performance statistics
 - A schema browser with index statistics on each field
 - A diagnostic tool for debugging text analysis
 - Support for dynamic core (indices) administration
- Faceting of search results (note: distinct from Lucene's faceting)
- A query parser called **eDisMax** that is more usable for parsing end user queries than Lucene's native query parser
- Distributed search support, index replication, and fail-over for scaling Solr
- Cluster configuration and coordination using ZooKeeper
- Solritas — a sample generic web search UI for prototyping and demonstrating many of Solr's search features

Also, there are two `contrib` modules that ship with Solr that really stand out, which are as follows:

- **DataImportHandler (DIH)**: A database, e-mail, and file crawling data import capability. It includes a debugger tool.
- **Solr Cell**: An adapter to the Apache Tika open source project, which can extract text from numerous file types.

As of the 3.1 release, there is a tight relationship between the Solr and Lucene projects. The source code repository, committers, and developer mailing list are the same, and they are released together using the same version number. Since Solr is always based on the latest version of Lucene, most improvements in Lucene are available in Solr immediately.

Comparison to database technology

There's a good chance that you are unfamiliar with Lucene or Solr and you might be wondering what the fundamental differences are between it and a database. You might also wonder if you use Solr, do you need a database.

The most important comparison to make is with respect to the data model — the organizational structure of the data. The most popular category of databases is relational databases — **RDBMS**. A defining characteristic of relational databases is a data model, based on multiple tables with lookup keys between them and a *join* capability for querying across them. That approach has proven to be versatile, being able to satisfy nearly any information-retrieval task in one query.

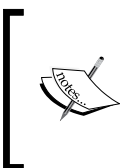
However, it is hard and expensive to scale them to meet the requirements of a typical search application consisting of many millions of documents and low-latency response. Instead, Lucene has a much more limiting *document-oriented* data model, which is analogous to a single table. Document-oriented databases such as MongoDB are similar in this respect, but their documents can be nested, similar to XML or JSON. Lucene's document structure is flat like a table, but it does support multivalued fields—a field with an array of values. It can also be very sparse such that the actual fields used from one document to the next vary; there is no space or penalty for a document to not use a field.



Lucene and Solr have limited support for *join* queries, but they are used sparingly as it significantly reduces the scalability characteristics of Lucene and Solr.

Taking a look at the Solr feature list naturally reveals plenty of search-oriented technology that databases generally either don't have, or don't do well. The notable features are relevancy score ordering, result highlighting, query spellcheck, and query-completion. These features are what drew you to Solr, no doubt. And let's not forget faceting. This is possible with a database, but it's hard to figure out how, and it's difficult to scale. Solr, on the other hand, makes it incredibly easy, and it does scale.

Can Solr be a substitute for your database? You can add data to it and get it back out efficiently with indexes; so on the surface, it seems plausible. The answer is that *you are almost always better off using Solr in addition to a database*. Databases, particularly RDBMSes, generally excel at *ACID* transactions, insert/update efficiency, in-place schema changes, multiuser access control, bulk data retrieval, and they have second-to-none integration with application software stacks and reporting tools. And let's not forget that they have a versatile data model. Solr falls short in these areas.



For more on this subject, see our article, *Text Search, your Database or Solr*, at <http://bit.ly/uwF1ps>, which although it's slightly outdated now, is a clear and useful explanation of the issues. If you want to use Solr as a document-oriented or key-value NoSQL database, *Chapter 4, Indexing Data*, will tell you how and when it's appropriate.

A few differences between Solr 4 and Solr 5

The biggest change that users will see in Solr 5 from Solr 4 is that Solr is now deployed as its own server process. It is no longer a WAR file that is deployed into an existing Servlet container such as Tomcat or Jetty. The argument for this boiled down to "you don't deploy your MySQL database in a Servlet container; neither should you deploy your Search engine". By owning the network stack and deployment model, Solr can evolve faster; for example, there are patches for adding HTTP/2 support and pluggable authentication mechanisms being worked on. While internally Solr is still using Jetty, that should be considered an implementation detail. That said, if you really want a WAR file version, and you're familiar with Java and previous Solr releases, you can probably figure out how to build one.

As part of Solr 5 being its own server process, it includes a set of scripts for starting, stopping, and managing Solr collections, as well as running as a service on Linux.

The next most obvious difference is that the distribution directory structure is different, particularly related to the old `example` and new `server` directory.



The rest of this chapter refers to Solr 5, however the remainder of the book was updated for Solr 4, and applies to Solr 5.

Getting started


We will get started by downloading Solr, examining its directory structure, and then finally run it.

This will set you up for the next section, which tours a running Solr 5 server.

1. **Get Solr:** You can download Solr from its website <http://lucene.apache.org/solr/>. This book assumes that you downloaded one of the binary releases (not the src (source) based distribution). In general, we recommend using the latest release since Solr and Lucene's code are extensively tested. For downloadable example source code, and book errata describing how future Solr releases affect the book content, visit our website <http://www.solrenterprise.com/>.

2. **Get Java:** The only prerequisite software needed to run Solr is Java 7 (that is, Java Version 1.7). But the latest version is Java 8, and you should use that. Typing `java -version` at a command line will tell you exactly which version of Java you are using, if any.

Java is available on all major platforms, including Windows, Solaris, Linux, and Mac OS X. Visit <http://www.java.com> to download the distribution for your platform. Java always comes with the **Java Runtime Environment (JRE)** and that's all Solr requires. The **Java Development Kit (JDK)** includes the JRE plus the Java compiler and various diagnostic utility programs. One such useful program is JConsole, which we'll discuss in *Chapter 11, Deployment*, and *Chapter 10, Scaling Solr* and so the JDK distribution is recommended.

 Solr is a Java-based web application, but you don't need to be particularly familiar with Java in order to use it. This book assumes no such knowledge on your part.


3. **Get the book supplement:** This book includes a code supplement available at our website <http://www.solrenterprisesearchserver.com/>; you can also find it on Packt Publishing's website at <http://www.packtpub.com/books/content/support>. The software includes a Solr installation configured for data from MusicBrainz.org, a script to download, and indexes that data into Solr—about 8 million documents in total, and of course various sample code and material organized by chapter. This supplement is not required to follow any of the material in the book. It will be useful if you want to experiment with searches using the same data used for the book's searches or if you want to see the code referenced in a chapter. The majority of the code is for *Chapter 9, Integrating Solr*.

Solr's installation directory structure

When you unzip Solr after downloading it, you should find a relatively straightforward directory structure (differences between Solr 4 and 5 are briefly explained here):

- `contrib`: The Solr `contrib` modules are extensions to Solr:
 - `analysis-extras`: This directory includes a few **text analysis** components that have large dependencies. There are some **International Components for Unicode (ICU)** unicode classes for multilingual support—a Chinese stemmer and a Polish stemmer. You'll learn more about text analysis in the next chapter.

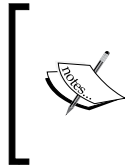
- `clustering`: This directory will have an engine for clustering search results. There is a one-page overview in *Chapter 8, Search Components*.
 - `dataimporthandler`: The `DataImportHandler` (DIH) is a very popular `contrib` module that imports data into Solr from a database and some other sources. See *Chapter 4, Indexing Data*.
 - `extraction`: Integration with Apache Tika—a framework for extracting text from common file formats. This module is also called `SolrCell` and Tika is also used by the DIH's `TikaEntityProcessor`—both are discussed in *Chapter 4, Indexing Data*.
 - `langid`: This directory contains a `contrib` module that provides the ability to detect the language of a document before it's indexed. More information can be found on the *Solr's Language Detection* wiki page at <http://wiki.apache.org/solr/LanguageDetection>.
 - `map-reduce`: This directory has utilities for working with Solr from Hadoop Map-Reduce. This is discussed in *Chapter 9, Integrating Solr*.
 - `morphlines-core`: This directory contains Kite Morphlines, a document ingestion framework that has support for Solr. The `morphlines-cell` directory has components related to text extraction. Morphlines is mentioned in *Chapter 9, Integrating Solr*.
 - `uima`: This directory contains library for Integration with Apache UIMA—a framework for extracting metadata out of text. There are modules that identify proper names in text and identify the language, for example. To learn more, see *Solr's UIMA integration* wiki at <http://wiki.apache.org/solr/SolrUIMA>.
 - `velocity`: This directory will have a simple search UI framework based on the Velocity templating language. See *Chapter 9, Integrating Solr*.
- `dist`: In this directory, you will see Solr's core and `contrib` JAR files. In previous Solr versions, the WAR file was found here as well. The core JAR file is what you would use if you're embedding Solr within an application. The Solr test framework JAR and `/test-framework` directory contain the libraries needed in testing Solr extensions. The `SolrJ` JAR and `/solrj-lib` are what you need to build Java based clients for Solr.
 - `docs`: This directory contains documentation and "Javadocs" for the related assets for the public Solr website, a quick tutorial, and of course Solr's API.

 If you are looking for documentation outside of this book, you are best served by the Solr Reference Guide. The docs directory isn't very useful.

- **example:** Pre Solr 5, this was the complete Solr server, meant to be an example layout for deployment. It included the Jetty servlet engine (a Java web server), Solr, some sample data and sample Solr configurations. With the introduction of Solr 5, only the `example-DIH` and `exampledocs` are kept, the rest was moved to a new `server` directory.
 - `example/example-DIH`: These are `DataImportHandler` configuration files for the example Solr setup. If you plan on importing with DIH, some of these files may serve as good starting points.
 - `example/exampledocs`: These are sample documents to be indexed into the default Solr configuration, along with the `post.jar` program for sending the documents to Solr.
- **server:** The files required to run Solr as a server process are located here. The interesting child directories are as follows:
 - `server/contexts`: This is Jetty's WebApp configuration for the Solr setup.
 - `server/etc`: This is Jetty's configuration. Among other things, here you can change the web port used from the presupplied 8983 to 80 (HTTP default).
 - `server/logs`: Logs are by default output here. Introduced in Solr 5 was collecting JVM metrics, which are output to `solr_gc.log`. When you are trying to size your Solr setup they are a good source of information.
 - `server/resources`: The configuration file for Log4j lives here. Edit it to change the behavior of the Solr logging, (though you can also changes levels of debugging at runtime through the **Admin** console).
 - `server/solr`: The configuration files for running Solr are stored here. The `solr.xml` file, which provides overall configuration of Solr lives here, as well as `zoo.cfg` which is required by SolrCloud. The subdirectory `/configsets` stores example configurations that ship with Solr.
 - `example/webapps`: This is where Jetty expects to deploy Solr from. A copy of Solr's WAR file is here, which contains Solr's compiled code and all the dependent JAR files needed to run it.
 - `example/solr-webapp`: This is where Jetty deploys the unpacked WAR file.

Running Solr

Solr ships with a number of example collection configurations. We're going to run one called *techproducts*. This example will create a collection and insert some sample data.



The addition of scripts for running Solr is one of the best enhancements in Solr 5. Previously, to start Solr, you directly invoked Java via `java -jar start.jar`. Deploying to production meant figuring out how to migrate into an existing Servlet environment, and was the source of much frustration.

First, go to the `bin` directory, and then run the main Solr command. On Windows, it will be `solr.cmd`, on *nix systems it will be just `solr`. Jetty's `start.jar` file by typing the following command:

```
>>cd bin
>>./solr start -e techproducts
```

The `>>` notation is the command prompt and is not part of the command. You'll see a few lines of output as Solr is started, and then the *techproducts* collection is created via an API call. Then the sample data is loaded into Solr. When it's done, you'll be directed to the Solr admin at `http://localhost:8983/solr`.

To stop Solr, use the same Solr command script:

```
>>./solr stop
```

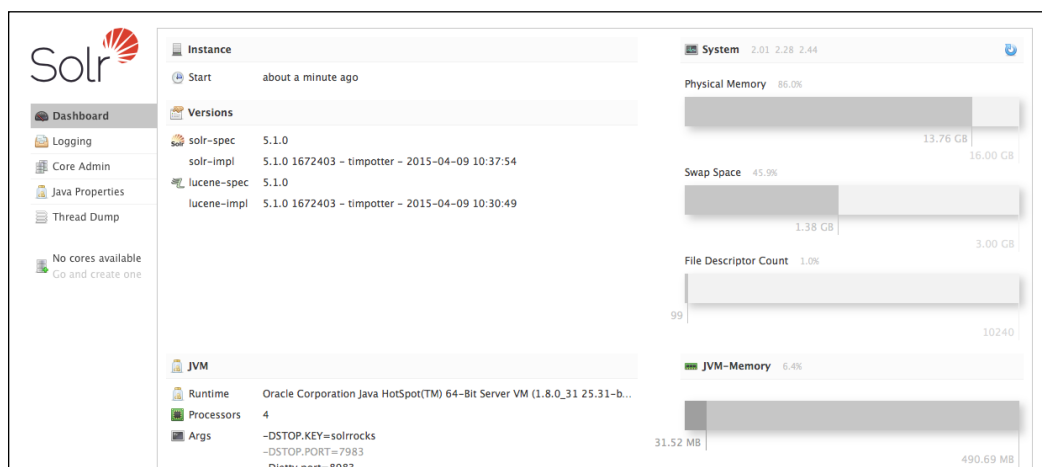
A quick tour of Solr

Point your browser to Solr's administrative interface at `http://localhost:8983/`. The admin site is a single-page application that provides access to some of the more important aspects of a running Solr instance.



The administrative interface is currently being completely revamped, and the below interface may be deprecated.

This tour will help you get your bearings in navigating around Solr.



In the preceding screenshot, the navigation is on the left while the main content is on the right. The left navigation is present on every page of the admin site and is divided into two sections. The primary section contains choices related to higher-level Solr and Java features, while the secondary section lists all of the running Solr cores.

The default page for the admin site is **Dashboard**. This gives you a snapshot of some basic configuration settings and stats, for Solr, the JVM, and the server. The **Dashboard** page is divided into the following subareas:

- **Instance:** This area displays when Solr started.
- **Versions:** This area displays various Lucene and Solr version numbers.
- **JVM:** This area displays the Java implementation, version, and processor count. The various Java system properties are also listed here.
- **System:** This area displays the overview of memory settings and usage; this is essential information for debugging and optimizing memory settings.
- **JVM-Memory:** This meter shows the allocation of JVM memory, and is key to understanding if garbage collection is happening properly. If the dark gray band occupies the entire meter, you will see all sorts of memory related exceptions!

The rest of the primary navigation choices include the following:

- **Logging:** This page is a real-time view of logging, showing the time, level, logger, and message. This section also allows you to adjust the logging levels for different parts of Solr at runtime. For Jetty, as we're running it, this output goes to the console and nowhere else. See *Chapter 11, Deployment*, for more information on configuring logging.
- **Core Admin:** This section is for information and controls for managing Solr cores. Here, you can unload, reload, rename, swap, and optimize the selected core. There is also an option for adding a new core.
- **Java Properties:** This lists Java system properties, which are basically Java-oriented global environment variables. Including the command used to start the Solr Java process.
- **Thread Dump:** This displays a Java thread dump, useful for experienced Java developers in diagnosing problems.

Below the primary navigation is a list of running Solr cores. Click on the **Core Selector** drop-down menu and select the **techproducts** link. You should see something very similar to the following screenshot:

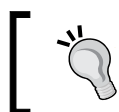
The screenshot displays the Solr Admin UI for the 'techproducts' core. The sidebar on the left contains navigation links: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, techproducts (selected), Overview, Analysis, Dataimport, Documents, Files, Ping, and Plugins / Stats. The main content area is divided into three sections: Statistics, Instance, and Replication (Master). The Statistics section shows metrics: Last Modified: 11 minutes ago, Num Docs: 32, Max Doc: 32, Heap Memory: -1, Usage: Deleted Docs: 0, Version: 3, Segment Count: 1, Optimized: (checked), Current: (checked). The Instance section shows: CWD: /Users/epugh/Downloads/solr-5.1.0/server, Instance: /Users/epugh/Downloads/solr-5.1.0/example/techproducts/solr/techproducts, Data: /Users/epugh/Downloads/solr-5.1.0/example/techproducts/solr/techproducts/data, Index: /Users/epugh/Downloads/solr-5.1.0/example/techproducts/solr/techproducts/data/index, Impl: org.apache.solr.core.NRTCachingDirectoryFactory. The Replication (Master) section shows a table with columns Version, Gen, and Size, listing Master (Searching) and Master (Replicable) nodes. The Admin Extra section is visible at the bottom.

	Version	Gen	Size
Master (Searching)	1430849685735	2	26.82 KB
Master (Replicable)	1430849685735	2	-

The default page labeled **Overview** for each core shows core statistics, information about replication, an **Admin Extra** area. Some other options such as details about **Healthcheck** are grayed out and made visible if the feature is enabled.

You probably noticed the subchoice menu that appeared below **techproducts**. Here is an overview of what those subchoices provide:

- **Analysis:** This is used for diagnosing query and indexing problems related to text analysis. This is an advanced screen and will be discussed later.
- **Data Import:** Provides information about the DataImport handler (the DIH). Like replication, it is only useful when DIH is enabled. The DataImport handler will be discussed in more detail in *Chapter 4, Indexing Data*.
- **Documents:** Provides a simple interface for creating a document to index into Solr via the browser. This includes a Document Builder that walks you through adding individual fields of data.
- **Files:** Exposes all the files that make up the core's configuration. Everything from core files such as `schema.xml` and `solrconfig.xml` to `stopwords.txt`.
- **Ping:** Clicking on this sends a ping request to Solr, displaying the latency. The primary purpose of the ping response is to provide a health status to other services, such as a load balancer. The ping response is a formatted status document and it is designed to fail if Solr can't perform a search query that you provide.
- **Plugins / Stats:** Here you will find statistics such as timing and cache hit ratios. In *Chapter 10, Scaling Solr*, we will visit this screen to evaluate Solr's performance.
- **Query:** This brings you to a search form with many options. With or without this search form, you will soon end up directly manipulating the URL using this book as a reference. There's no data in Solr yet, so there's no point in using the form right now.
- **Replication:** This contains index replication status information, and the controls for disabling. It is only useful when replication is enabled. More information on this is available in *Chapter 10, Scaling Solr*.
- **Schema Browser:** This is an analytical view of the schema that reflects various statistics of the actual data in the index. We'll come back to this later.
- **Segments Info:** Segments are the underlying files that make up the Lucene data structure. As you index information, they expand and compress. This allows you to monitor them, and was newly added to Solr 5.



You can partially customize the admin view by editing a few templates that are provided. The template filenames are prefixed with `admin-extra`, and are located in the `conf` directory.

Loading sample data

Solr comes with some sample data found at `example/exampledocs`. We saw this data loaded as part of creating the *techproducts* Solr core when we started Solr. We're going to use that for the remainder of this chapter so that we can explore Solr more, without getting into schema design and deeper data loading options. For the rest of the book, we'll base the examples on the digital supplement to the book – more on that later.

We're going to re-index the example data by using the `post.jar` Java program, officially called SimplePostTool. Most JAR files aren't executable, but this one is. This simple program takes a Java system variable to specify the collection: `-Dc=techproducts`, iterates over a list of Solr-formatted XML input files, and HTTP posts it to Solr running on the current machine – `http://localhost:8983/solr/techproducts/update`. Finally, it will send a `commit` command, which will cause documents that were posted prior to the commit to be saved and made visible. Obviously, Solr must be running for this to work. Here is the command and its output:

```
>> cd example/exampledocs
>> java -Dc=techproducts -jar post.jar *.xml
SimplePostTool version 5.0.0
Posting files to [base] url http://localhost:8983/solr/techproducts/
update using
content-type application/xml...
POSTing file gb18030-example.xml
POSTing file hd.xml
etc.
14 files indexed.
COMMITting Solr index changes to http://localhost:8983/solr/techproducts/
update...
```

If you are using a Unix-like environment, you have an alternate option of using the `/bin/post` shell script, which wraps the SimplePostTool.



The `post.sh` and `post.jar` programs could be used in a production scenario, but they are intended just as a demonstration of the technology with the example data.

Let's take a look at one of these XML files we just posted to Solr, `monitor.xml`:

```
<add>
  <doc>
    <field name="id">3007WFP</field>
    <field name="name">Dell Widescreen UltraSharp 3007WFP</field>
    <field name="manu">Dell, Inc.</field>
    <!-- Join -->
    <field name="manu_id_s">dell</field>
    <field name="cat">electronics</field>
    <field name="cat">monitor</field>
    <field name="features">30" TFT active matrix LCD, 2560 x 1600,
    .25mm dot pitch, 700:1 contrast</field>
    <field name="includes">USB cable</field>
    <field name="weight">401.6</field>
    <field name="price">2199</field>
    <field name="popularity">6</field>
    <field name="inStock">true</field>
    <!-- Buffalo store -->
    <field name="store">43.17614,-90.57341</field>
  </doc>
</add>
```

The XML schema for files that can be posted to Solr is very simple. This file doesn't demonstrate all of the elements and attributes, but it shows the essentials. Multiple documents, represented by the `<doc>` tag, can be present in series within the `<add>` tag, which is recommended for bulk data loading scenarios. This subset may very well be all that you use. More about these options and other data loading choices will be discussed in *Chapter 4, Indexing Data*.

A simple query

Point your browser to `http://localhost:8983/solr/#/techproducts/query`—this is the query form described in the previous section. The search box is labeled **q**. This form is a standard HTML form, albeit enhanced by JavaScript. When the form is submitted, the form inputs become URL parameters to an HTTP GET request to Solr. That URL and Solr's search response is displayed to the right. It is convenient to use the form as a starting point for developing a search, but then subsequently refine the URL directly in the browser instead of returning to the form.

Run a query by replacing the `*:*` in the `q` field with the word `lcd`, then clicking on the **Execute Query** button. At the top of the main content area, you will see a URL like this `http://localhost:8983/solr/techproducts/select?q=monitor&wt=json&indent=true`. The URL specifies that you want to query for the word `lcd`, and that the output should be in indented JSON format.

Below this URL, you will see the search result; this result is the response of that URL.

By default, Solr responds in XML, however the query interface specifies JSON by default. Most modern browsers, such as Firefox, provide a good JSON view with syntax coloring and hierarchical controls. All response formats have the same basic structure as the JSON you're about to see. More information on these formats can be found in *Chapter 4, Indexing Data*.

The JSON response consists of a two main elements: `responseHeader` and `response`. Here is what the header element looks like:

```
"responseHeader": {
  "status": 0,
  "QTime": 1,
  "params": {
    "q": "lcd",
    "indent": "true",
    "wt": "json"
  }
}
...
```

The following are the elements from the preceding code snippet:

- `status`: This is always zero, unless there was a serious problem.
- `QTime`: This is the duration of time in milliseconds that Solr took to process the search. It does not include streaming back the response. Due to multiple layers of caching, you will find that your searches will often complete in a millisecond or less if you've run the query before.
- `params`: This lists the request parameters. By default, it only lists parameters explicitly in the URL; there are usually more parameters specified in a `<requestHandler/>` in `solrconfig.xml`. You can see all of the applied parameters in the response by setting the `echoParams` parameter to `true`.



More information on these parameters and many more is available in *Chapter 5, Searching*.

Next up is the most important part, the results:

```
"response": {
  "numFound": 5,
  "start": 0,
```

The `numFound` value is the number of documents matching the query in the entire index. The `start` parameter is the index offset into those matching (ordered) documents that are returned in the response below.

Often, you'll want to see the score of each matching document. The document score is a number that represents how relevant the document is to the search query. This search response doesn't refer to scores because it needs to be explicitly requested in the `fl` parameter—a comma-separated field list. A search that requests the score via `fl=*,score` will have a `maxScore` attribute in the "response" element, which is the maximum score of all documents that matched the search. It's independent of the sort order or result paging parameters.

The content of the result element is a list of documents that matched the query. The default sort is by descending score. Later, we'll do some sorting by specified fields.

```
{
  "id": "9885A004",
  "name": "Canon PowerShot SD500",
  "manu": "Canon Inc.",
  "manu_id_s": "canon",
  "cat": [
    "electronics",
    "camera"
  ],
  "features": [
    "3x zoop, 7.1 megapixel Digital ELPH",
    "movie clips up to 640x480 @30 fps",
    "2.0\" TFT LCD, 118,000 pixels",
    "built in flash, red-eye reduction"
  ],
  "includes": "32MB SD card, USB cable, AV cable, battery",
  "weight": 6.4,
  "price": 329.95,
  "price_c": "329.95,USD",
  "popularity": 7,
  "inStock": true,
  "manufacturedate_dt": "2006-02-13T15:26:37Z",
  "store": "45.19614,-93.90341",
  "_version_": 1500358264225792000
},
...
```