Apache Solr でもっと賢く検索する: 第 1 回 基本機能と Solr スキーマ

Solr を使った索引付け、検索、ファセット・ブラウジング

Grant Ingersoll
Senior software engineer
Center for Natural Language Processing at Syracuse University

2007年 5月 29日

Solr は Lucene をベースにしたエンタープライズ対応の検索サーバーで、ファセット検索、検索結果の強調表示、そして複数の出力形式をサポートします。Solr を紹介するこの 2 回の連載記事では、Lucene Java コミッターの Grant Ingersoll が Solr ならではの優れた全文検索機能を簡単に Web アプリケーションに取り込む方法を伝授します。

このシリーズの他の記事を見る

ユーザーが必要なときに必要なコンテンツに接続できるという機能は、もうオプションではありません。Google をはじめとする高性能の検索エンジンが進歩するなか、ユーザーは探しているものが素早く簡単に見つかる高度な検索結果を期待するようになっています。オンライン・ショッピング・サイトに大きな期待を抱いているのは、皆さんの上司にしても同じです。上司は、スケーラブルで可用性にも管理のしやすさにも優れ、しかも莫大な費用をかけなくてもインストールできる検索ソリューションを期待しています。皆さん自身の望みと言えば、キャリアをアップし、雇用主と顧客を満足させて心の平静を保ちたいということに尽きます。

そんなすべてのニーズに応えてくれるのが、オープン・ソースの検索サーバー、Apache Solr です。Web アプリケーションに簡単に統合できる Lucene Java ベースの Solr は、ファセット検索機能と検索結果の強調表示機能、そして XML/XSLT および JSON を含めた複数の出力形式のサポートを提供します。また、インストールして構成するのも簡単で、HTTP ベースの管理インターフェースが備わっています。Solr の基本的な検索機能はそのままでも使うことができますが (この検索機能は感動的です)、会社の要求に応じて拡張することも可能です。Solr には活発な開発者コミュニティーもあるため、必要なときには助けを求められます。

この 2 回連載の記事では Solr を紹介し、その機能を披露するとともに サンプル Web アプリケーションに完全に統合する方法を説明します。今回最初に取り上げるのは、インストール手順と構成手順を含めた Solr の基礎です。続いて Solr の機能に慣れ親しんでもらうためのサンプル・アプリケーション (ブログ・インターフェース) を紹介し、Solr でコンテンツに索引を付けて検索する

方法、Solr でのファセット・ブラウジングのサポートについて説明します。そして最後に Solr のスキーマと、Solr のスキーマがサンプル・アプリケーションの索引構造に合わせてどのように構成されているかを説明して第 1 回を締めくくることにします。

Solr の歴史

当初、CNET Networks で開発された Solr は、2006 年の初めに Apache Software Foundation に寄贈され、Lucene のトップ・プロジェクトの傘下に入りました。2007 年までのインキュベーション期間で、Solr は着実に機能を増やし、ユーザー、コントリビューター、コミッターからなる確固としたコミュニティーを築き上げました。インキュベーション期間を終えた Solr は現在、Lucene (Apache による Java ベースの全文検索エンジン・ライブラリー) のサブプロジェクトとなっています。

インストールと構成

Solr を使い始めるには、以下のソフトウェアがインストールされている必要があります。

- Java 1.5 以降
- Ant 1.6.x 以降
- Web ブラウザー。これは、管理ページを表示するために使用します。推奨されるブラウザーは Firefox です。Internet Explorer では手順が異なる場合があります。
- Tomcat 5.5 などのサーブレット・コンテナー。この記事のサンプルでは、Tomcat がポート 8080 (Tomcat のデフォルト) で実行されていることを前提とします。他のサーブレット・コンテナーを実行している場合、または別のポートで実行している場合には、指定された URL を変更しないとサンプル・アプリケーションと Solr にアクセスできないことがあります。この記事は、サンプル・アプリケーションのすべてのパーツが Tomcat のローカル・ホストで実行されているという前提で作成しました。また、Solr には Jetty が付属していることに注意してください。

上記のアプリケーションのダウンロードおよびインストール方法については、「参考文献」を参照してください。

Solr のセットアップ手順

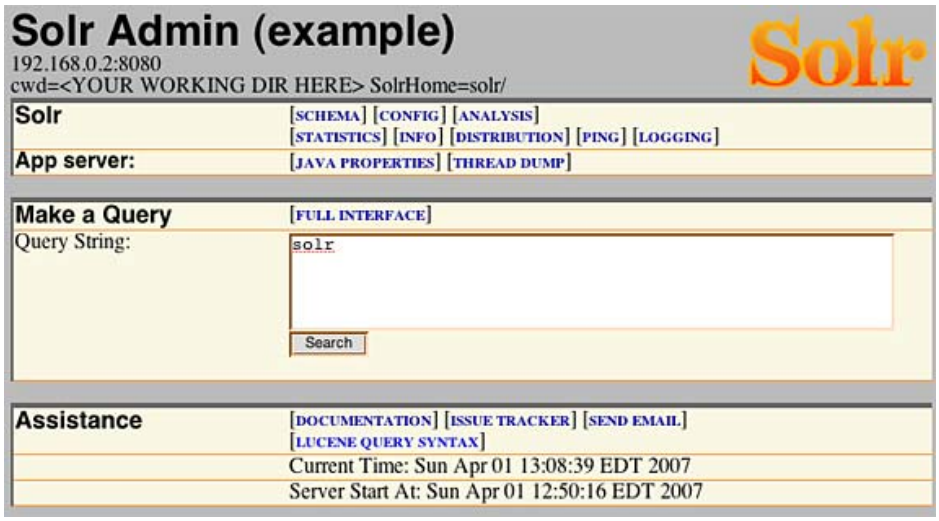
前提条件のアプリケーションをインストールしたら、Apache Mirrors Web siteサイトから Solr バージョン 1.1 をダウンロードしてください。次に、以下の手順に従います。

1. サーブレット・コンテナーを停止します。
2. 作業ディレクトリーとして使用するディレクトリーで、コマンド・ラインから `mkdir dw-solr`を実行します。
3. `cd dw-solr`を実行します。
4. Solr WAR ファイル (example/webapps/solr.war にあります) をサーブレット・コンテナーの webapps ディレクトリーにコピーします。これにより、`apache-solr-1.1.0-incubating` ディレクトリーが作成されます。incubating タグは無視してください。Solr のインキュベーション期間はもう終わっています
5. サンプル・アプリケーションをダウンロードして、カレント・ディレクトリーにコピーします。解凍すると、現行の作業ディレクトリーに `solr` ディレクトリーが作成されます。これが、この記事で一貫して使用する Solr のホーム・ディレクトリーです。
6. 以下のいずれかの方法で、Solr のホーム・ロケーションを設定します。

- java システム・プロパティーを `solr.solr.home` に設定します (`solr.solr.home` は誤植ではありません)。
- `java:comp/env/solr/home` の JNDI ルックアップを構成して `solr` ディレクトリーを指すようにします。
- solr ディレクトリーが含まれるディレクトリーでサーブレット・コンテナーを起動します (デフォルトの Solr ホームは、現行の作業ディレクトリーの下にある solr です)。

7. `cd solr`
8. Create the sample WAR file: `ant war`.
9. Solr WAR ファイルの場合と同じように、サンプル WAR ファイル (dw-solr/solr/dist/dw.war 内にあります) をサーブレット・コンテナーの webapps ディレクトリーにコピーします。この WAR ファイルの Java コードは dw-solr/solr/src/java 内に、JSP とその他の Web ファイルは dw-solr/solr/src/webapp 内にあります。
10. すべてが正しく機能することを確認するため、サーブレット・コンテナーを起動してブラウザーで http://localhost:8080/solr/admin/ を指定します。すべてが正しければ、図 1 のようなページが表示されます。管理ページが表示されない場合は、コンテナーのログでエラー・メッセージを確認してください。また、サーブレット・コンテナーを dw-solr ディレクトリーから起動したことも確認してください。dw-solr ディレクトリーから起動しないと、Solr ホーム・ロケーションは正しく設定されません。

図 1. Solr 管理画面の例



Lucene について

Solr を理解するということは、すなわち Lucene を理解するということです。Lucene は Java ベースのパフォーマンスに優れたテキスト検索エンジン・ライブラリーで、元々は Doug Cutting が作成し、その後 Apache Software Foundation に寄贈されました。Lucene の速さ、使いやすさ、そして活発なコミュニティーといった理由から、多くのアプリケーションでは Lucene を使って検索機能の強化を図っています。この Lucene をベースにした Solr は、必要最小限のプログラミングで Lucene をエンタープライズ対応に変身させます。Lucene についての詳細は、「参考文献」を参照してください。

Solr の基礎

Lucene を包含して拡張する Solr では、使用する用語もほとんど共通しています。さらに重要なことは、Solr が作成する索引は Lucene 検索エンジン・ライブラリーと完全に互換するという

点です。つまり、Solr を正しく構成し、場合によっては多少のコーディング作業を行えば、他の Lucene アプリケーションに組み込まれた索引を読み取って使用できるようになります。その上、Luke をはじめとする多数の Lucene ツールは Solr で作成された索引でも同じく有効に機能します。

Solr と Lucene では、索引は 1 つ以上の `Document` で構成されます。`Document` は 1 つ以上の `Field` で構成され、`Field` は名前、コンテンツ、そして Solr にコンテンツの処理方法を指示するメタデータで構成されます。`Field` には、ストリング、数値、ブール値、日付はもとより、他にも型を追加したければどんな型でも含められます。例えば `Field` の記述に、索引付けと検索の際に Solr がコンテンツをどのように処理するかを指定するオプションをいくつか使用することもできます。これらのオプションについては後で詳しく説明するので、とりあえずは表 1 にリストした重要な属性の抜粋を見てください。

表 1. Field の属性

属性名	説明
indexed	索引付き `Field` は、検索およびソートすることが可能です。Solr の分析プロセスを索引付き `Field` で実行すれば、コンテンツを変更して結果を改善または変更することもできます。Solr の分析プロセスについては、次のセクションで詳しく説明します。
stored	保管済み `Field` のコンテンツは索引に保存されます。この属性はコンテンツを検索して強調表示するのには役立ちますが、実際の検索には必要ありません。多くのアプリケーションでは、例えばファイルに含まれる実際のコンテンツではなくコンテンツの場所へのポインターが保管されるからです。

Solr の分析プロセスについて

Solr の分析プロセスを実行すると、アプリケーションのコンテンツを変更してから索引を付けることができます。Solr と Lucene の `Analyzer` は、1 つの `Tokenizer` と 1 つ以上の `TokenFilter` で構成されます。`Tokenizer` の役割は `Token` を作成することです。`Token` は大抵の場合、索引を付ける対象の単語に相当します。`TokenFilter` は `Tokenizer` から `Token` を取り込み、索引を付ける前に `Token` を変更または削除することができます。例を挙げると、Solr の `WhitespaceTokenizer` は空白文字で単語を分割し、`StopFilter` で検索結果から共通の単語を削除します。他の種類の分析としては、ステミング (語幹の抽出)、同義語拡張、そして大文字変換もあります。アプリケーションに特有の方法で分析を行う必要がある場合、Solr にはそのニーズを満たすトークナイザーとフィルターが 1 つ以上見つかるはずです。

分析は、検索操作中にクエリーに対して適用することもできます。原則として、クエリーに対して実行される分析は、索引を付ける文書に対して実行される分析と同じでなければなりません。このようなコンセプトが初めてのユーザーは、文書トークンではステミングを行ってクエリー・トークンでは行わないという誤りを犯しがちですが、それによって多くの場合は検索結果がゼロ件になってしまいます。Solr の XML 構成では、後で説明するように単純な宣言を使うので、`Analyzer` を簡単に作成することができます。

Solr および Lucene の分析ツールならびに索引構造とその他の機能についての詳細は、「参考文献」を参照してください。

サンプル・アプリケーション

以降のセクションでは、現実的なサンプル・アプリケーションを用いて Solr の機能を紹介します。このサンプル・アプリケーションは Web ベースのブログ・インターフェースです。このインターフェースを使って、エントリーをログに記録してメタデータを割り当て、それからエントリーに索引を付けて検索します。索引付けと検索プロセスのステップごとに、オプションで Solr に送信中のコマンドを表示することもできます。

サンプル・アプリケーションを表示するには、ブラウザーで http://localhost:8080/dw/index.jsp を指定します。すべてが正しくセットアップされていれば (「Solr のセットアップ手順」で説明)、「Sample Solr Blog Search」というタイトルとその直下にメニュー項目がある単純なユーザー・インターフェースが表示されるはずです。メニューについては、2 回の記事を通してすべての項目を説明します。

索引付けの操作

Solr で索引付けと検索を開始するには、サーブレット・コンテナーにデプロイされた Solr Web アプリケーションに HTTP 要求を送信します。Solr は要求を取り込むと、その要求に適切だと判断した `SolrRequestHandler` を使用して要求を処理します。応答は同じようにして HTTP で返されます。デフォルト構成で返されるのは Solr の標準 XML 応答ですが、Solr を構成して別の応答形式にすることもできます。要求と応答の処理をカスタマイズする方法については、記事の後半で説明します。

索引付けは、入力 (この例では、ブログのエントリー、キーワード、そしてその他のメタデータ) を受け取り、`HTTP Post` という XML メッセージで Solr に渡して索引を付けるというプロセスです。Solr の索引付けサーブレットには、以下の 4 種類の索引付け要求を渡すことができます。

- add/update は、Solr に文書を追加または更新する場合に使われます。追加と更新は、commit が行われるまで検索には有効になりません。
- commit は、最後の commit 以降に行われたすべての変更を検索時に有効にするよう Solr に指示します。
- optimize は、Lucene のファイルを再構成して検索のパフォーマンスを改善します。最適化は通常、索引付けが完了したときに行うと効果的です。更新が頻繁に行われる場合は、使用率の低い時間帯に最適化をスケジュールしてください。ただし、索引は最適化しなくても適切に機能します。最適化のプロセスには時間がかかる場合があります。
- delete は id またはクエリーを基準に指定することができます。id による削除操作では、指定された id を持つ特定の文書が削除されます。一方、クエリーによる削除操作では、クエリーによって返されたすべての文書が削除されます。

索引付けの例

索引付けプロセスの詳細を見るには、http://localhost:8080/dw/index.jsp をブラウズしてください。まず、各フィールドに適切なエントリーを入力してフォームを完成させ、Submit ボタンを押します。サンプル・アプリケーションがエントリーを受け取ると、Solr 要求を作成して次の画面に表示します。リスト 1 は、Submit ボタンが押されると Solr に送信される `add` コマンドの例です。

リスト 1. Solr の add コマンドの例

```
<add>
<doc>
<field name="url">http://localhost/myBlog/solr-rocks.html</field>
<field name="title">Solr Search is Simply Great</field>
<field name="keywords">solr,lucene,enterprise,search</field>
<field name="creationDate">2007-01-06T05:04:00.000Z</field>
<field name="rating">10</field>
<field name="content">Solr is a really great open source search server. It scales,
it's easy to configure and the Solr community is really supportive.</field>
<field name="published">on</field>
</doc>
</add>
```

リスト 1 に記載された `<doc>` の各フィールド・エントリーが、作成した文書の Lucene 索引に追加する `Field` を Solr に指示します。`add` コマンドには複数の `<doc>` を追加することもできます。これらのフィールドを Solr がどのように処理するかは後で説明するので、ここではリスト 1 に指定された 7 つのフィールドを含む 1 つの文書に索引が付けられるということがわかれば十分です。

上記のコマンドを「Solr XML Command」ページから実行依頼すると、結果が Solr に送信されて処理されます。`HTTP POST` がコマンドを送信する先は、http://localhost:8080/solr/update で実行している Solr Update Servlet です。すべてが順調に運ぶと、`<result status="0"/>` という結果の XML 文書が返されます。Solr は同じ URL を持つ文書を自動的に更新します (サンプル・アプリケーションでの URL は、その文書が以前に追加されたことがあるかどうかを識別するために Solr が使用する固有の id です)。

索引付けの練習

早速 2、3 の文書を追加してコミットし、次のセクションで検索できるようにしてください。`add` コマンドの構文に慣れたら、Index ボタンの近くにある「Display XML...」チェック・ボックスからチェック・マークを外して、「Solr XML Command」ページがスキップされるようにしても構いません。この記事付属のサンプル・ファイルには、記載している例の多くで使われている索引のバージョンが含まれています。

文書を削除するには、文書の URL を http://localhost:8080/dw/delete.jsp ページに入力し、コマンドを実行依頼して表示してから、Solr にコマンドを実行依頼します。索引付けコマンドと削除コマンドについての詳細は、「参考文献」に記載した「Solr Wiki」を参照してください。

検索コマンド

それではいよいよ、追加した文書を検索してみましょう。Solr ではクエリーとして `HTTP GET` メッセージと `HTTP POST` メッセージの両方を受け付けます。送られてきたクエリーは該当する `SolrRequestHandler` で処理されます。ここでの説明ではデフォルトの `StandardRequestHandler` を使います。Solr を構成して他の `SolrRequestHandler` に対応させる方法については、次回の記事で説明します。

実際の検索操作を見るには、サンプル・アプリケーションに戻って http://localhost:8080/dw/searching.jsp までブラウズしてください。この画面は、検索に関するいくつかのオプションが追加されているだけで、索引付け画面とほとんど同じです。索引付け操作と同じように、この画面ではさまざまな入力フィールドに値を入力し、検索パラメーターを選択してクエリーをサンプ

ル・アプリケーションに実行依頼することができます。サンプル・アプリケーションでは、Solr でよく使用されるクエリー・パラメーターの一部が強調表示されています。これらのパラメーターは以下のとおりです。

クエリー構文についての注意

StandardRequestHandler のSolr クエリー構文は、Lucene の QueryParser でサポートされるクエリー構文にソートのサポートがいくつか追加された形になります。サンプル・アプリケーションでは入力された値の検証をほとんど行いません。また、ブースティング、フレージング、範囲フィルターなどの機能も実行しませんが、いずれの機能も Solr と Lucene では使用できます。Lucene の QueryParser についての詳細は、「参考文献」を参照してください。連載第 2 回では、クエリー構文と結果のデバッグに役立つ管理インターフェースのツールを紹介します。

- Boolean operator: デフォルトでは、検索語と組み合わせて使うブール演算子はOR になっています。ブール演算子を AND に設定すると、検索結果の文書ですべての検索語が表示されます。
- Number of results: 返される結果の最大数を指定します。
- Start: 結果セットでの開始オフセットです。ページネーションに役立ちます。
- Highlight: 文書のフィールドに含まれる一致した検索語を強調表示します。リスト 2 の終わりの方にある <lst> というノードを見るとわかるように、強調表示される検索語は でマークされます。

値を入力して実行依頼すると、ブログ・アプリケーションが、Solr に実行依頼するために作成したクエリー・ストリングを返します。実行依頼を受けた Solr は、すべてが正しく、一致する文書がある場合には、結果、強調表示情報、そしてクエリーに関するメタデータなどが含まれる XML 応答を返します。リスト 2 は、検索結果の一例です。

リスト 2. 検索結果の例

```
<response>
<lst name="responseHeader">
<int name="status">0</int>
<int name="QTime">6</int>
<lst name="params">
<str name="rows">10</str>
<str name="start">0</str>
<str name="fl">*,score</str>
<str name="hl">true</str>
<str name="q">content:"faceted browsing"</str>
</lst>
</lst>
<result name="response" numFound="1" start="0" maxScore="1.058217">
<doc>
<float name="score">1.058217</float>
<arr name="all">
<str>http://localhost/myBlog/solr-rocks-again.html</str>
<str>Solr is Great</str>
<str>solr,lucene,enterprise,search,greatness</str>
<str>Solr has some really great features, like faceted browsing
and replication</str>
</arr>
<arr name="content">
<str>Solr has some really great features, like faceted browsing
and replication</str>
</arr>
<date name="creationDate">2007-01-07T05:04:00.000Z</date>
```

```
<arr name="keywords">
<str>solr,lucene,enterprise,search,greatness</str>
</arr>
<int name="rating">8</int>
<str name="title">Solr is Great</str>
<str name="url">http://localhost/myBlog/solr-rocks-again.html</str>
</doc>
</result>
<lst name="highlighting">
<lst name="http://localhost/myBlog/solr-rocks-again.html">
<arr name="content">
<str>Solr has some really great features, like <em>faceted</em>
<em>browsing</em> and replication</str>
</arr>
</lst>
</lst>
</response>
```

クエリー・メッセージにはいくつものパラメーターを含めることができます。そのうちの代表的なパラメーターを表 2 に記載します。完全なリストについては、「参考文献」に記載した「Solr Wiki」を参照してください。

表 2. 代表的なクエリー・パラメーター

パラメーター	説明	例
q	Solr での検索を行う際に使用するクエリー。構文の詳しい説明については、「参考文献」に記載した「Lucene QueryParser Syntax」を参照してください。情報をソートするには、セミコロンを追加し、その後にトークン化されていない索引付きフィールド名を続けます (右記を参照)。デフォルトのソートはスコアの降順でソートする `score desc` です。	`q=myField:Java AND otherField:developerWorks; date asc` このクエリーは指定された 2 つのフィールドを検索し、date フィールドを基準に結果をソートします。
start	結果セットでの開始オフセットを指定します。結果のページングに便利です。デフォルト値は 0 です。	`start=15` 15 番目にランクされた結果から返します。
rows	結果として返される文書の最大数です。デフォルト値は 10 です。	`rows=25`
fq	オプションのフィルタリング・クエリーを指定します。クエリーの結果は、フィルタリング・クエリーによって返された結果のみの検索に絞り込まれます。フィルタリングされたクエリーは Solr によってキャッシュに入れられます。このパラメーターは、複雑なクエリーを素早く行うのに非常に役立ちます。	q パラメーターに渡すことが可能で、ソート情報が含まれない有効なクエリー。
hl	`hl=true` の場合、クエリー応答でスニペットを強調表示します。デフォルトは false です。強調表示パラメーターのその他のオプションについては、「参考文献」に記載した「Solr Wiki」を参照してください。	`hl=true`
fl	文書結果で返される `Field` のセットをコンマで区切られたリストとして指定します。デフォルトは`*`で、これはすべてのフィールドを意味します。「score」を指定すると、スコアも返されます。	`*,score`

ファセット・ブラウジング

最近の傾向として、人気の高いショッピング・サイトにはユーザーがメーカーや価格、著者などを基準に結果を絞り込めるようにする便利な検索基準のリストが加えられています。これらのリストは、ファセット・ブラウジングの結果です。ファセット・ブラウジングとは、返された結果を存在が保証されている有意義なカテゴリーに分類する 1 つの方法で、ファセットを使用してユーザーが検索結果を絞り込めるようにします。

http://localhost:8080/dw/facets.jsp までブラウズすると、実際のファセットを見ることができます。このページでファセット・ブラウジングに関連するのは以下の 2 つの入力フォームです。

- テキスト域。ここに、索引の `all` フィールドに対して実行するクエリーを入力することができます。`all` フィールドとは、索引付けされたその他すべてのフィールドの連結のことだと思ってください (この後、詳細を説明します)。
- フィールドのドロップダウン・リスト。ファセット分類に使用できます。ここにはすべての索引付きフィールドがリストされるわけではありません。その理由については、この後説明します。

クエリーを入力し、ドロップダウン・リストからファセット・フィールドを選択して Submit をクリックし、生成されたクエリーと併せて Solr に渡します。するとブログ・アプリケーションが結果を解析して、図 2 のような結果を返します。

図 2. ファセット・ブラウジングの結果の例

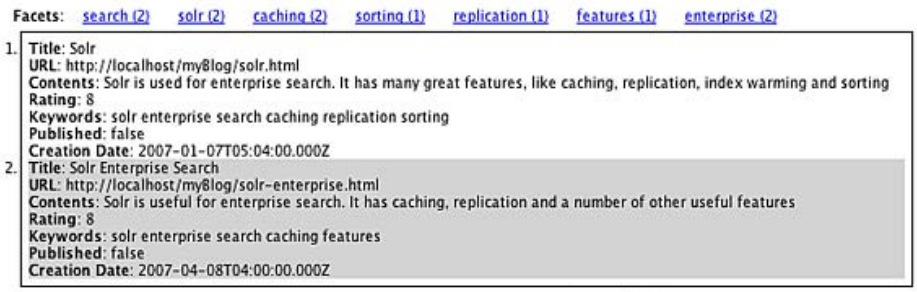


図 2 では、上端にゼロ以外の値についてのファセット・カウントが表示され、その下に実行依頼されたクエリーと一致した 2 つの検索結果が表示されています。上記の例でいずれかのファセット・リンクをクリックすると、元のクエリーにそのファセットを新しいキーワードとして組み合わせたクエリーが実行依頼されます。例えば元のクエリーが `q=Solr` でファセット・フィールドが `keywords` だとすると、図 2 の `replication` リンクをクリックした場合、実行依頼される新しいクエリーは q=Solr AND keywords:replication となります。

ファセット分類を機能させるには、Solr 内で有効にしたり、構成したりする必要はありませんが、アプリケーションのコンテンツに新しい方法で索引を付けなければならない場合があります。ファセット分類は索引付きフィールドを基準に行われ、トークン化されていない小文字以外の単語で最も効果的に機能します (`content` フィールドや文書に追加されたその他のフィールドをファセット・フィールドのドロップダウン・リストに入れなかったのはそのためです)。ファセット分類の本質は人間が読み取れる値を表示することなので、通常はファセット・フィールドを保管する必要はありません。

Solr はファセットのカテゴリーを作成しないという点にも注意してください。カテゴリーは、索引付けの際にアプリケーション自体によって追加されなければなりません。これは、アプリケーションに索引を付けるときに、文書にキーワードを割り当てたのと同じことです。ファセット・フィールドが指定されると、Solr はファセットの内容とそのカウントを解明するためのロジックを提供します。

Solr のスキーマ

Up to now, I have introduced you to Solr's features without actually explaining how they are configured. The remainder of the article focuses on configuration, first introducing the Solr schema (schema.xml) and then showing you how it relates to Solr's features.

エディター (できれば XML タグを強調表示できるもの) で、<INSTALL_DIR>/dw-solr/solr/conf に置かれた schema.xml ファイルを開いてください。まず気付くことは、コメントの多さです。オープン・ソースのソフトウェアを使ったことがある方なら、スキーマ・ファイルと Solr 全体として文書化されていることは、非常にありがたいと思うはずです。schema.xml には十分にコメントが書かれているので、この記事ではファイルの重要な属性を抜粋して説明し、詳細はコメントを参照してもらうことにします。最初に、`<schema>` タグに含まれたスキーマの名前 (dw-solr) に注目してください。Solr がデプロイメントごとにサポートするスキーマの数は 1 つです。今後は複数のスキーマがサポートされるようになると思いますが、現時点では 1 つのスキーマしか許可されません (1 つのコンテナーで複数のデプロイメントに対応できるように Tomcat やその他のコンテナーを簡単に構成する方法については、「参考文献」に記載した「Solr Wiki」を参照してください)。

スキーマは、以下の 3 つのセクションに分類できます。

- 型
- フィールド
- その他の宣言

`<types>` セクションに含まれるのは、Solr (および Lucene) による `Field` の処理方法を定義した一般的で繰り返し使える内容です。サンプル・スキーマには、`string` から `text` まで、名前別に 13 のフィールド型があります。`<types>` セクションの先頭に宣言された `sint` や `boolean` などのフィールド型は、Solr にプリミティブ型を保管するために使用します。大抵の場合、Lucene が扱うのはストリングだけなので、整数、浮動小数点、日付、倍精度を検索するには特殊な操作が必要です。これらのフィールド型を使用すると、索引を付けるコンテンツを適切な特殊操作で扱うよう Solr にアラートが出されるため、ユーザーが介入する必要はありません。

型

この記事の前半で Solr の分析プロセスの基本について触れましたが、`text` フィールド型の宣言を詳しく見ていくと、Solr がどのように分析を処理するかが具体的にわかります。リスト 3 に、`text` フィールド型の宣言を示します。

リスト 3. text フィールド型の宣言

```
<fieldtype name="text" class="solr.TextField" positionIncrementGap="100">
<analyzer type="index">
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
```

```
<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
<filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="1"
catenateNumbers="1" catenateAll="0"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.EnglishPorterFilterFactory"
protected="protwords.txt"/>
<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer>
<analyzer type="query">
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
<filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
generateNumberParts="1" catenateWords="0"
catenateNumbers="0" catenateAll="0"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.EnglishPorterFilterFactory" protected="protwords.txt"/>
<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer>
</fieldtype>
```

クラス属性

Solr スキーマでは多くの場合、クラス属性は省略されます。例えば `solr.TextField` とあれば、これは `org.apache.solr.schema.TextField` の単なる省略表現です。連載第 2 回で説明しますが、クラスパス内の `org.apache.solr.schema.FieldType` クラスを拡張する有効なクラスであればどれでも使用できます。

まず、リスト 3 では 2 つの異なる `Analyzer` を宣言していることに注目してください。索引付けと検索の場合の `Analyzer` はまったく同じわけではありませんが、その違いはクエリーの分析中に同義語が追加されるかされないかという点だけです。索引付けや検索を行う前には、ステミング、ストップワードの削除、そして同様の操作がすべてトークンに適用されるので、同じタイプのトークンが使用されることになります。次に注目する点は、tokenizer を宣言してから使用する filter を宣言していることです。サンプル・アプリケーションの Solr 構成は、以下のように設定されています。

- 空白文字でトークン化してから、共通の単語をすべて削除 (StopFilterFactory)
- ダッシュ付きの特殊な大/小文字、大/小文字の変換などを処理し (WordDelimiterFilterFactory)、すべての語を小文字に変換 (LowerCaseFilterFactory)
- Porter Stemming アルゴリズムによるステミングを実行 (EnglishPorterFilterFactory)
- すべての重複を削除 (RemoveDuplicatesTokenFilterFactory)

上記の分析例には検索結果を改善するための多くの一般的手法を盛り込んでいますが、これが唯一のテキスト分析方法だとは思わないでください。それぞれのアプリケーションには、この例はもとより、Solr または Lucene の既存の `Analyzer`でも扱っていないような独自の分析が必要になることがあります。分析関連のさまざまなオプション、および他の `Analyzer` の使用方法については、「参考文献」に記載した「Solr 分析」を参照してください。

フィールド

続いてスキーマの `<fields>` セクションを見ると、索引付けと検索に使用する 8 つのフィールド (実際は 7 つですが、`all` を加えています) を Solr がどのように処理するかがわかります。これらのフィールドはリスト 4 のとおりです。

リスト 4. ブログ・アプリケーション用に宣言されたフィールド

```
<field name="url" type="string" indexed="true" stored="true"/>
<field name="title" type="text" indexed="true" stored="true"/>
<field name="keywords" type="text_ws" indexed="true" stored="true"
multiValued="true" omitNorms="true"/>
<field name="creationDate" type="date" indexed="true" stored="true"/>
<field name="rating" type="sint" indexed="true" stored="true"/>

<field name="published" type="boolean" indexed="true" stored="true"/>

<field name="content" type="text" indexed="true" stored="true" />
<!-- catchall field,
containing many of the other searchable text fields
(implemented via copyField further on in this schema) -->
<field name="all" type="text" indexed="true" stored="true" multiValued="true"/>
```

フィールド型の内容がわかれば、それぞれのフィールドがどのように処理されるかは明白です。例えば `url` フィールドは、索引を付けて保管される `string` フィールドで、分析対象ではありません。一方、`text` フィールドはリスト 3 で宣言された `Analyzer` によって分析されます。それでは `all` フィールドについてはどうでしょう。`all` フィールドは `title` や `content` と同じく `text` フィールドですが、代替検索メカニズムを容易にするために、ここには複数のフィールドが連結されて含まれます (ファセット検索で `all` フィールドを使用したことを思い出してください)。

フィールドの属性に関しては、表 1 で `indexed` および `stored` の意味を説明しました。一方、`multiValued` 属性は特殊な例で、この属性は、`Document` で同じ名前が複数追加された `Field` を使用できることを意味します。この記事のサンプルの場合、例えば `keywords` を何度も追加できるということになります。`omitNorms` は、Solr (および Lucene) にノルムを保管しないよう指示する属性です。ノルムを省略すると、スコアに影響しない `Field` (ファセットの計算に使用する `Field` など) でメモリーを節約するのに役立ちます。

`<fields>` セクションの説明を終える前に、フィールド宣言の後に続く `<dynamicField>` 宣言について簡単に説明しておきます。動的フィールドは、フィールド宣言で定義された属性を使って任意の時点で任意の文書に追加できる特殊なフィールドです。動的フィールドと通常のフィールドとの重要な違いは、動的フィールドには schema.xml にあらかじめ宣言された名前が必要ないという点です。Solr はまだ宣言されていないフィールド名を受け取ると名前宣言でのグロブのようなパターンを適用し、`<dynamicField>` 宣言で定義されたセマンティクスに従ってフィールドを処理します。例えば、`<dynamicField name="*_i" type="sint" indexed="true" stored="true"/>` が意味するのは、`myRating_i` という名前のフィールドはフィールドとして宣言されていないとしても、Solr ではこのフィールドを `sint` として処理するということです。これは例えば、ユーザーに検索対象のコンテンツを定義させる場合に役立ちます。

その他の宣言

schema.xml ファイルの最後の部分には、フィールドとクエリーの実行に関するさまざまな宣言が含まれます。なかでも最も重要な宣言は、`<uniqueKey>url</uniqueKey>` です。この宣言は、前に宣言された`url` フィールドが文書を追加または更新するタイミングを決定するために使用する固有の ID であることを Solr に指示します。`defaultSearchField` は、クエリーの語の先頭に検索フィールドが付けられていない場合にSolr がクエリーで使用する `Field`です。この例では `q=title:Solr` のようなクエリーを使用しますが、代わりに `q=Solr` と入力すると、デフォルトの

検索フィールドが適用されます。最終的な結果は `q=all:Solr` と同等になります。これは、ブログ・アプリケーションでは `all` がデフォルトの検索フィールドだからです。

`<copyField>` メカニズムにより、文書のコンテンツすべてを別のフィールドに手作業で追加しなくても、`all` フィールドを作成することができます。同じコンテンツに何通りもの索引を付けるには、コピー・フィールドが便利です。例えば、大/小文字を考慮した完全一致と大/小文字を無視した一致を同時に指定する必要がある場合、コピー・フィールドを使えば送られてきたコンテンツを自動的に分析することができます。分析が完了すると、受信したとおりのコンテンツの文字をすべて小文字にした状態で索引が付けられます。

次回の予告: エンタープライズに対応した Solr

これまでのところで、Solr をインストールし、サンプル・アプリケーションで Solr を使って文書に索引を付けて検索する方法を説明しました。また、Solr ではファセット・ブラウジングがどのように機能するか、そして Solr の schema.xml ファイルで索引構造を宣言する方法も理解できたはずです。この記事に付属のサンプル・アプリケーションでこれらの機能を実際に実行してみると、Solr のコマンドをフォーマット設定する方法がわかります。

連載第 2 回では、Solr を単純な検索インターフェースからエンタープライズ対応の検索ソリューションに拡張する機能を紹介します。Solr の管理インターフェースと拡張構成オプション、そしてキャッシング、複製、ロギングなどのパフォーマンス関連の機能を説明するとともに、企業それぞれのニーズに合わせて Solr を拡張する方法についても簡単に触れる予定です。それまでは、サンプル・アプリケーションをあれこれ使って Solr の基本機能に慣れ親しんでください。

ダウンロード

内容	ファイル名	サイズ
Sample Solr application	j-solr1.zip	500KB

著者について

Grant Ingersoll

Grant Ingersoll は、シラキュース大学の Center for Natural Language Processing でシニア・ソフトウェア・エンジニアを務めています。専門とするプログラミング分野は、情報検索、質問応答、テキスト・カテゴリー化、抽出です。彼は Lucene Java プロジェクトのコミッター兼議長でもあります。