Apache Solr でもっと賢く検索する: 第 2 回 エンタープライズに対応した Solr

管理、構成、そしてパフォーマンス

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

2007年 6月 05日

Solr の紹介を締めくくるこの記事では、Lucene Java ™ のコミッター Grant Ingersoll が、管理インターフェース、高度な構成オプション、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機能を含め、Solr のエンタープライズ向け機能を説明します。

このシリーズの他の記事を見る

連載第 1 回 では、多種多様な Web アプリケーションに簡単に統合できる HTTP をベースとしたオープン・ソースの検索サーバー、Apache Solr の紹介として、索引付け、検索、ブラウジングなどの Solr の基本機能を実例で説明した他、Solr のスキーマを紹介し、Solr のスキーマが Solr の機能を構成する上で果たす役割を説明しました。第 2 回となるこの記事では、Solr を大規模な実動環境に対する理想的なソリューションにしている数々の機能を紹介してこの連載を締めくくることにします。今回の記事で取り上げる話題は、管理、キャッシング、複製、そして拡張性です。

Solr をインストールしてセットアップする手順については、第 1 回を参照してください。

構成と管理

このセクションでは、Solr 機能の監視と制御を行うために用意された数々のオプションを紹介します。まず取り上げるのは、http://localhost:8080/solr/admin/ にある Solr の Administration Start Page です。この開始ページが表示されたら、先に進む前にいったん立ち止まって、ページのさまざまなメニュー・オプションについてよく理解してください。開始ページのオプションは、それぞれが提供する情報によってグループ化されています。

- Solr は、アクティブなスキーマ (第 1 回を参照)、構成、そして現行デプロイメントの統計に関する詳細情報を提供します。
- App serverは、スレッド化情報、全 Java システム・プロパティーのリストを含め、コンテナーの現行状況の詳細を提供します。

- Make a Queryには、クエリーを手っ取り早くデバッグしやすいインターフェース、そして豊富な機能を備えたクエリー・インターフェースへのリンクがあります。
- Assistanceには、Solr 使用中に発生する可能性のある問題を理解し、解決するための有益な外部リソースへのリンクがあります。

以降のセクションでは上記のメニュー・オプションを取り上げ、重要な管理機能を重点に説明していきます。

Solr の構成オプションを使い始めるには、開始ページの `CONFIG` リンクをクリックして現在使用中の solrconfig.xml ファイルを表示してください。このファイルは、サンプル・アプリケーションの dw-solr/solr/conf ディレクトリーにあります。それではまず、索引付けとクエリー処理に関連する一般構成オプションのいくつかを見てみましょう。キャッシング、複製、そしてSolr の拡張に関連する構成オプションについては、その後のセクションで説明します。

索引付けの構成

Solr の索引付けプロセスを制御するのは、 `mainIndex` タグ・セクションで定義される下位レベルの Lucene の係数です。Lucene ベンチマーク・コントリビューション (`contrib/benchmark` の下の Lucene ソース内にあります) には、これらの係数の変更による効果を評価するための豊富なツールが含まれています。また、さまざまな変更に伴うトレードオフについて学ぶには、「参考文献」セクションの「Solr Performance Factors」を参照してください。表 1 で、Solr の索引付けプロセスを制御する係数について概説します。

表 1. 索引付けのパフォーマンス係数

係数	説明
useCompoundFile	多数の Lucene 内部ファイルを単一のファイルに統合して、使用するファイル数を減らします。Solr が使用するファイル・ハンドルの数を減らすには役立ちますが、パフォーマンスがある程度劣化するという犠牲が伴います。アプリケーションのファイル・ハンドルが不足しているのでない限り、デフォルトの `false` で十分です。
mergeFactor	下位 Lucene セグメントをマージする頻度を決定します。値が小さいほど (最小値は 2) メモリー使用量が少なくなりますが、索引付けにかかる時間は長くなります。値を大きくすると、索引付けにかかる時間は短縮されますが、メモリー使用量が増えます。
maxBufferedDocs	メモリー内の文書をマージして新規セグメントを作成するまでに索引を付ける必要がある最小の文書数を定義します。セグメントとは、索引情報を保管するための Lucene ファイルのことです。値を大きくすると、索引付けにかかる時間は短縮されますが、メモリー使用量が増えます。
maxMergeDocs	Solr がマージする `Document` の最大数を制御します。更新回数が多いアプリケーションには、小さい値 (< 10,000) が適しています。
maxFieldLength	特定 `Document` の `Field` に追加する用語の最大許容数を制御することによって文書を切り詰めます。大きいサイズの文書が見込まれる場合は、この値を増やしてください。ただし、この値を高く設定しすぎるとメモリー不足のエラーが発生する可能性があります。
unlockOnStartup	`unlockOnStartup` は、Solr に対し、マルチスレッド環境で索引の保護に使用されるロック機構を無視するように指定します。場合によっては、不適切なシャットダウンやその他のエラーにより索引がロックされ、追加や更新が行えなくなること

	があります。この値を true に設定すると、追加および更新できるように起動時にロックが無効に設定されます。

クエリーの処理の構成

`<query>` セクションにはcaching関連以外の機能がいくつかありますが、これらの機能については知っておく必要があります。まず、`<maxBooleanClauses>` タグは、クエリーを作成するために結合できる節の最大数を定義します。ほとんどのアプリケーションではデフォルトの 1024 で十分ですが、ワイルドカード・クエリーやレンジ・クエリーを多用するアプリケーションの場合には、この最大数を増やして `TooManyClausesException` がスローされないようにしてください。

ワイルドカード・クエリーとレンジ・クエリー

ワイルドカード・クエリーおよびレンジ・クエリーは、クエリーの仕様に一致する可能性のあるすべての用語を組み込むように、自動的に拡張される Lucene のクエリーです。ワイルドカード・クエリーでは * と ? のワイルドカード演算子を使用できます。一方のレンジ・クエリーは、ある範囲内の文書のみを一致結果とするという指定をします。例えば b* を検索すると、おそらく数千の異なる用語がクエリーに結合されるため、TooManyClausesException がスローされる結果となります。

`Document` から少数の `Field` しか取得しないことが見込まれるアプリケーションの場合には、`<enableLazyFieldLoading>` プロパティーを true に設定できます。遅延ロードの一般的なシナリオは、アプリケーションが検索結果のリストを戻して表示し、ユーザーがリスト内の 1 つをクリックしたときに、索引に保管された元の文書を表示するというものです。大抵の場合、最初の表示には短い情報をいくつか示すだけで済みます。大きな `Document` を取得する際の犠牲を考えれば、必要になるまで文書全体をロードするのは避けるのが賢明です。

`<query>` セクションは、Solr で発生するイベント関連のオプションを定義します。まず説明しておくと、Solr (実際は Lucene) は、`Searcher` という名前の Java クラスを使って `Query` インスタンスを処理します。`Searcher` は索引のコンテンツに関するデータをメモリーにロードしますが、このプロセスは索引のサイズ、CPU、そして使用可能なメモリーの量によって長時間かかる場合があります。この設計を改善してパフォーマンスを大幅に向上させるため、Solr では「ウォーム」手法を採り入れています。つまり、新しい `Searcher` をオンライン状態にして実際のユーザー・クエリーを処理させる前に、ウォームアップ状態にするということです。`Query` セクションの `<listener>` オプションは、`newSearcher` および `firstSearcher` イベントを定義します。これは、新しい `Searcher` または最初の `Searcher` がインスタンス化されるときに実行するクエリーを指定するために使用できます。特定のクエリーが要求されるようなアプリケーションの場合には、これらのセクションのコメントを外せば、最初の `Searcher` または新規 `Searcher` が作成されるときに該当するクエリーを実行することができます。

solrconfig.xml ファイルの残りのセクションは、`<admin>` セクションは別として、キャッシング、複製、そして Solr の拡張またはカスタマイズに関する項目を扱っています。`admin` セクションは、管理インターフェースをカスタマイズするために使用します。`admin` セクションの構成についての詳細は、Solr Wiki、および solrconfig.xml ファイルのインライン・コメントを参照してください。

監視、ロギング、統計

http://localhost:8080/solr/adminの管理ページには、Solr 管理者が Solr プロセスを監視するために使用するメニュー項目があります。表 2 で、これらの項目について説明します。

表 2. 監視、ロギング、統計のための Solr 管理オプション

メニュー名	管理 URL	説明
Statistics	http:// localhost:8080/solr/admin/stats.jsp	Statistics 管理ページは、Solr のパフォーマンス関連のさまざまな役に立つ統計情報を提供します。統計情報には以下のものがあります。 • 索引がロードされた時刻と索引に含まれる文書の数 • クエリーの実行に使われた `SolrRequestHandler` に関する使用情報 • 追加、削除、コミットなどの回数をはじめとする索引付けプロセスに関するデータ • キャッシュの実装およびヒット/ミス/追い出しに関する情報
Info	http:// localhost:8080/solr/admin/registry.jsp	実行中の Solr バージョンやクエリー、更新、およびキャッシングの現行の実装で使用されているクラスに関する詳細を説明します。Solr サブバージョン・リポジトリー内でのファイルの場所に関する情報やファイルの機能概略も含まれます。
Distribution	http://localhost:8080/solr/admin/distributiondump.jsp	索引の配布および複製に関する情報を表示します。詳細は「配布と複製」を参照してください。
Ping	http://localhost: 8080/solr/admin/ping	solrconfig.xml ファイルの `admin` セクションに指定されたクエリーが含まれる ping 要求をサーバーに発行します。
Logging	http:// localhost:8080/solr/admin/logging.jsp	このメニューでは、現行アプリケーションのロギング・レベルを動的に変更できます。ロギング・レベルを変更すると、実行中に発生する可能性がある問題をデバッグするのに役立ちます。
Java properties	http: //localhost:8080/solr/admin/get-properties.jsp	現行システムで使用されているすべての Java システム・プロパティーを表示します。Solr では、コマンド・ラインでシステム・プロパティーを置換できます。この機能の実装情報については、solrconfig.xml ファイルを参照してください。
Thread dump	http:// localhost:8080/solr/admin/threaddump.jsp	JVM で実行しているすべてのスレッドに関するスタック・トレース情報を表示します。

分析プロセスのデバッグ

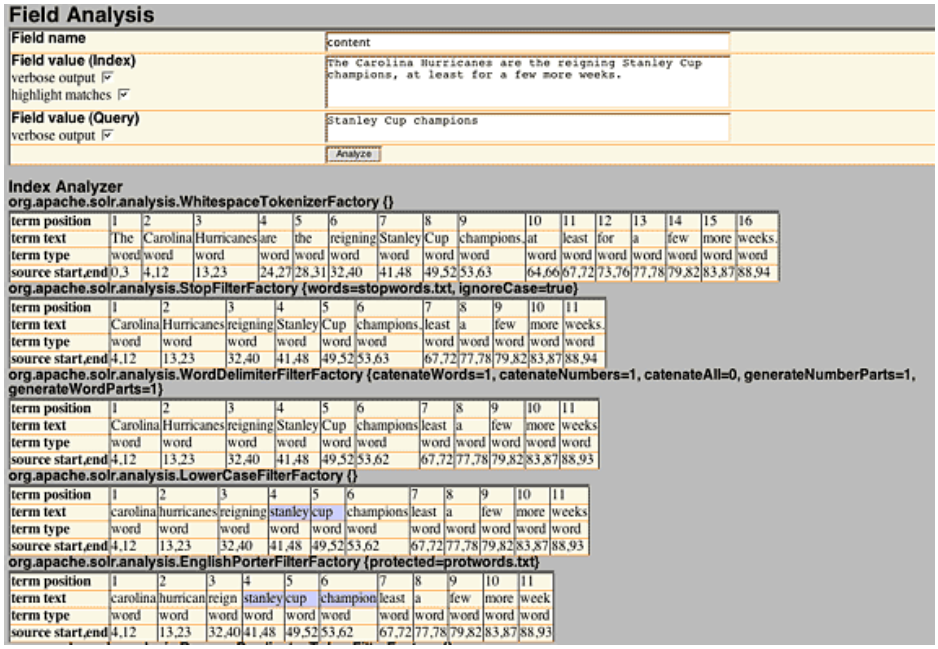
検索の実装を作成しているときに、特定の文書と一致するはずの検索を入力したのにも関わらず、結果のなかにその文書が含まれないことはよくあります。ほとんどの場合、この失敗は以下のいずれかが要因となっています。

- クエリー分析と文書分析が一致していないため (滅多に推奨されることではありませんが、クエリーと文書の分析方法を変えてみることが可能です。)
- `Analyzer` が 1 つ以上の用語を期待通りに変更していないため

上記の問題を調べるには、http://localhost:8080/ solr/admin/analysis.jsp にある Solr の Analysis 管理機能を利用できます。Analysis ページにクエリーと文書両方のテキストからのスニペット、そして

テキストの分析方法を特定する `Field` 名を入力すると、変更されているテキストの段階的な結果が返されます。図 1 は、「The Carolina Hurricanes are the reigning Stanley Cup champions, at least for a few more weeks」というセンテンスと、サンプル・アプリケーションの schema.xml に指定されている `content Field` に対して分析されたクエリー「Stanley Cup champions」を分析した結果の一部です。

図 1. 分析のデバッグ



分析画面には、結果の表の上に指定された `Tokenizer` または `TokenFilter` による各用語の処理結果が表示されます。これを見ると、例えば `StopFilterFactory` によって The、are、the の 3 つのワードが取り除かれているのがわかります。`EnglishPorterFilterFactory` は champions の語幹 champion と Hurricanes の語幹 hurrican を抽出しています。指定された文書でクエリー用語と一致している箇所は、紫色で強調表示されます。

クエリーのテスト

管理ページの `Make a Query` セクションには、クエリーを入力して結果を調べるための検索ボックスがあります。この入力ボックスに第 1 回で説明した Lucene クエリー・パーサー構文を入力することもできますが、 `Full Interface` リンクを使うと、戻される結果の数、結果セットに含めるフィールド、出力形式をはじめとする多数の検索機能を制御できます。さらに、このインターフェースでは文書のスコアについての説明を表示することもできるので、どの用語が一致して、どのように用語がスコアされたかを詳しく理解するのに役立ちます。説明を表示するには、`Debug: enable` オプションにチェックマークを付けて検索結果の終わりまでスクロールします。

インテリジェント・キャッシング

インテリジェント・キャッシングは、Solr を検索サーバーとして際立たせている重要なパフォーマンス機能の 1 つです。例えば、Solr ではキャッシュの自動ウォームを行えます。キャッシュを

使い始める前に古いキャッシュ内の情報を利用するというこの方法により、既存のユーザーに引き続きサービスを提供しながらもパフォーマンスを向上することができます。Solr に用意された 4 種類のキャッシュ・タイプは、いずれも solrconfig.xml ファイルの `<query>` セクションで構成します。表 3 に、solrconfig.xml ファイルで使用しているタグ名に従ってキャッシュ・タイプを記載します。

表 3. Solr キャッシュ・タイプ

キャッシュのタグ名	説明	自動ウォームの適用
filterCache	フィルターを使用すると、Solr は指定されたクエリーに一致するすべての文書の id を順不同のセットとして保管し、クエリーのパフォーマンスを効果的に向上できるようになります。これらのフィルターをキャッシュに入れると、Solr の呼び出しを繰り返すときに結果セットの検索が迅速になります。一般的なシナリオは、フィルターをキャッシュに入れてから、このフィルターによって絞り込みを行うクエリーを連続で実行して、検索対象の文書数を制限するというものです。	可
queryResultCache	クエリー、ソート基準、および要求された文書数に対応する文書 id の順序付きセットをキャッシュに入れます。	可
documentCache	内部 Lucene 文書 id を使用して (Solr の固有 id と混同しないように)、Lucene の `Document` をキャッシュに入れます。Lucene の内部 `Document` id は索引操作により変更されることがあるので、このキャッシュの自動ウォームは行えません。	不可
指定キャッシュ	指定キャッシュはユーザーが定義するキャッシュで、カスタム Solr プラグインで使用できます。	`org.apache.solr.search.CacheRegenerator` が実装されている場合は可

それぞれのキャッシュ宣言には、以下の最大 4 つの属性を使用します。

- `class` はキャッシュ実装の Java 名です。
- `size` はエントリーの最大数です。
- `initialSize` はキャッシュの初期サイズです。
- `autoWarmCount` は新規キャッシュのウォームを行うために古いキャッシュから使用するエントリー数です。自動ウォームのエントリー数が多ければヒットするキャッシュも多くなりますが、ウォーム時間が長くなります。

あらゆるキャッシング・スキームと同様に、キャッシュ・パラメーターを設定する際には、メモリー、CPU、そしてディスク·アクセスにおけるトレードオフを検討しなければなりません。キャッシュのヒット対ミスの比率、そして明け渡し統計値を調べられる Statistics 管理ページは、キャッシュのサイズを微調整するのに大いに役立ちます。また、すべてのアプリケーションにキャッシングが有益であるとは限りません。実際に一部のアプリケーションでは、絶対に使用されることのないキャッシュに項目を保管するための追加ステップが逆効果になる場合もあります。

配布と複製

大量のクエリーを受け取るアプリケーションでは、単一の Solr サーバーではパフォーマンスの要件に対処しきれない場合があります。そのため Solr には、負荷分散されたクエリー・サーバー・グループに含まれる複数のサーバーで Lucene 索引を複製するためのメカニズムが用意されています。複製プロセスは、solrconfig.xml ファイルといくつかのシェル・スクリプト (サンプル・アプリケーションの dw-solr/solr/bin に配置) で有効にされたイベント・リスナーの組み合わせで処理されます。

複製アーキテクチャーでは、1 つの Solr サーバーがマスター・サーバーとして機能し、クエリー要求を処理する 1 つ以上のスレーブ・サーバーに索引のコピー (########) を配布します。索引付けコマンドはマスター・サーバーに送信され、クエリーはスレーブ・サーバーに送信されます。マスター・サーバーでは手動でスナップショットを作成することも、solrconfig.xml の `<updateHandler>` セクションを構成して、`commit` イベントや `optimize` イベントの受信時にスナップショットの作成が起動されるようにすることもできます (リスト 1 を参照)。手動プロセスまたはイベント駆動型プロセスのいずれにしても、`snapshooter` スクリプトがマスター・サーバーで呼び出され、サーバー上に `snapshot.yyyymmddHHMMSS` というディレクトリーが作成されます (`yyyymmddHHMMSS` は、スナップショットが実際に作成された時刻)。するとスレーブ・サーバーが、rsync を使用して Lucene 索引内の変更されているファイルだけをコピーします。

リスト 1. 更新ハンドラーのリスナー

```
<listener event="postCommit" class="solr.RunExecutableListener">
<str name="exe">snapshooter</str>
<str name="dir">solr/bin</str>
<bool name="wait">true</bool>
<arr name="args"> <str>arg1</str> <str>arg2</str> </arr>
<arr name="env"> <str>MYVAR=val1</str> </arr>
</listener>
```

リスト 1 は、`commit` イベントの受信後にマスター・サーバーでスナップショットを作成するための構成です。`optimize` イベントを処理する場合の構成もこれに似ています。この構成例では、`commit` が完了した後に Solr が `solr/bin` ディレクトリー内にある `snapshooter` スクリプトを呼び出し、指定された引数と環境変数を渡します。`wait` 引数は、Solr にスレッドが戻ってから処理を続けるように指示するためのものです。`snapshooter` やその他の構成スクリプトの詳細な実行方法については、Solr Web サイトに掲載されている「Solr Collection and Distribution Scripts」資料を参照してください (「参考文献」を参照)。

スレーブ・サーバー側では、`snappuller` シェル・スクリプトを使用してマスター・サーバーからスナップショットを取得します。`snappuller` がマスター・サーバーから必要なファイルを取得した後は、`snapinstaller` シェル・スクリプトを使用してスナップショットをインストールし、Solr に新規スナップショットが使用可能になったことを通知します。スナップショットが作成される頻度に応じて、システムが定期的にこれらのステップを実行するようにスケジュールすることをお勧めします。スレーブ・サーバーがスナップショットをプルできるようにするには、マスター・サーバー側で rsync デーモンを起動する必要があります。rsync デーモンは `rsyncd-enable` シェル・スクリプトによって有効にされた後、`rsyncd-start` コマンドを使用して起動します。スレーブ・サーバーでは、`snappuller-enable` シェル・スクリプトが実行されるまでは、`snappuller` シェル・スクリプトを呼び出すことはできません。

配布のトラブルシューティング

索引更新の配布を最適化するためにあらゆる取り組みが行われていますが、Solr で問題の原因となる共通したシナリオがいくつかあります。

- サイズの大きな索引の最適化には極めて時間がかかることがありますが、索引更新の頻度が少ないとしても索引の最適化は行わなければなりません。最適化すると多くの Lucene 索引ファイルが単一のファイルにマージされます。つまり、スレーブ・サーバーは索引全体をコピーしなければならなくなりますが、このような方法で最適化したほうがスレーブ・サーバーのそれぞれで索引を最適化しようとするよりは遥かに賢明です。この場合、スレーブ・サーバーがマスター・サーバーと同期されていなければ、新しいコピーを取得し直さなければならなくなります。
- 新しいスナップショットをマスター・サーバーから頻繁にプルすると、スレーブ・サーバーのパフォーマンスが劣化する可能性があります。その原因は、`snappuller` で変更をコピーする際のオーバーヘッド、そして新しい索引がインストールされる際のキャッシュのウォームによるものです。頻繁な索引更新に伴うトレードオフについての詳細は、「参考文献」の「Solr Performance Factors」リンクを参照してください。

最終的には、変更を追加、コミット、そしてスレーブ・サーバーにプルする頻度はビジネスのニーズとハードウェアの性能によって決まります。さまざまなシナリオを十分にテストすることが、スナップショットを作成してマスター・サーバーからプルするタイミングを決定するのに役立ちます。Solr の配布機能と複製機能の設定および実行方法についての詳細は、「参考文献」の「Solr Collection and Distribution」資料を参照してください。

Solr のカスタマイズ

Solr には、Solr の処理を拡張または変更するために独自の機能を追加できるプラグイン・ポイントが複数あります。さらに Solr はオープン・ソースなので、別の機能が必要な場合にはいつでもソース・コードを変更できます。Solr にプラグインを組み込むには、以下の 2 つの方法があります。

- Solr WAR を解凍し、`WEB-INF/lib` ディレクトリーの下に独自のライブラリーを追加してからファイルを再パッケージ化します。この WAR ファイルをサーブレット・コンテナーにデプロイします。
- JAR を Solr ホームの `lib` ディレクトリーに配置して、サーブレット・コンテナーを起動します。この方法はカスタムの `ClassLoader` を使うため、すべてのサーブレット・コンテナーに有効だとは限りません。

以降のセクションでは、Solr を拡張する可能性の高い領域を取り上げます。

要求処理

Solr では、既存の機能がビジネスのニーズを満たさない場合、アプリケーション固有の要求処理機能を実装することができます。例えば、独自のクエリー言語をサポートする必要がある場合や Solr をユーザー・プロファイルと統合してパーソナライズした結果を提供したいという場合が考えられます。カスタム要求処理を実装する際に必要となるメソッドを定義しているのは、`SolrRequestHandler` インターフェースです。実は、Solr には第 1 回で使用したデフォルト

の「標準」要求ハンドラーの他にも要求ハンドラーがいくつか定義されています。以下に示すのが、Sorl のすべての要求ハンドラーです。

- デフォルトの `StandardRequestHandler` は、Lucene クエリー・パーサー構文を使用してクエリーを処理する他、ソートとファセット・ブラウジングを追加で行います。
- `DisMaxRequestHandler` は、大幅に単純化した構文を使って複数の `Field` を検索するように設計されています。この要求ハンドラーはソート (標準ハンドラーとはわずかに異なる構文を使用) とファセット・ブラウジングもサポートします。
- `IndexInfoRequestHandler` は、索引に関する情報 (索引に含まれる文書の数、索引内の `Field` など) を取得できます。

要求ハンドラーは、要求内の `qt` パラメーターで指定します。Solr サーブレットはこのパラメーターの値を使用して指定された要求ハンドラーを検索し、入力をその要求ハンドラーに渡して処理させます。要求ハンドラーの宣言およびネーミングを指定するのは solrconfig.xml の `<requestHandler>` タグです。独自の要求ハンドラーを追加するには、スレッドセーフな独自の `SolrRequestHandler` インスタンスを実装し、これを上記で定義したように Solr に追加して、前に説明した方法でクラスパスに含めます。すると、`HTTP GET` または `POST` メソッドを使ってその要求ハンドラーに要求を送信できるようになります。

応答処理

要求処理と同じく、応答出力をカスタマイズすることも可能です。レガシー検索出力をサポートしなければならないアプリケーションや、バイナリーまたは暗号化された出力フォーマットを必要とするアプリケーションでは、`QueryResponseWriter` を実装して必要なフォーマットで出力するようにできます。ただし、独自の `QueryResponseWriter` を追加する前に、表 4 に概説する Solr 付属の実装を調べてください。

表 4. Solr のクエリー応答書き出しプログラム

クエリー応答書き出しプログラム	説明
XMLResponseWriter	最も汎用的な応答フォーマットで、第 1 回のブログ・アプリケーションで説明したように XML で結果を出力します。
XSLTResponseWriter	`XSLTResponseWriter` は、指定された XSLT 変換を `XSLTResponseWriter` の出力に適用します。要求の `tr` パラメーターで、使用する XSLT 変換の名前を指定します。指定する変換は、Solr ホームの conf/xslt ディレクトリー内になければなりません。XSLT 応答書き出しプログラムの詳細は、「参考文献」を参照してください。
JSONResponseWriter	結果を JSON (JavaScript Object Notation) フォーマットで出力します。JSON は単純で人間が解読できるデータ交換形式で、マシンが解析するのにも簡単です。
RubyResponseWriter	`RubyResponseWriter` は JSON フォーマットを拡張し、Ruby で安全に結果を評価できるようにします。Solr で Ruby を使うことを考えている場合は、「参考文献」の acts_as_solr および Flare のリンクにアクセスしてください。
PythonResponseWriter	JSON 出力フォーマットを拡張して Python の `eval` メソッドで安全に使用できるようにします。

`QueryResponseWriter` を Solr に追加するには、solrconfig.xml ファイルの `<queryResponseWriter>` タグおよび関連する属性を使用します。応答タイプは要求の `wt` パラメーターで指定します。

デフォルトは「標準」で、つまり solrconfig.xml 内で `XMLResponseWriter` に設定されます。最後に、`QueryResponseWriter` のインスタンスが、応答の作成に使用する `write()` メソッドと `getContentType()` メソッドのスレッドセーフな実装を提供する必要があります。

Analyzer、Tokenizer、TokenFilter、および FieldType

新規の `Analyzer`、`Tokenizer`、および `TokenFilter` を使用することで、Solr の索引付け出力をカスタマイズして新しい分析機能を提供することができます。独自の `Tokenizer` または `TokenFilter` が必要なアプリケーションでは、アプリケーション固有の `TokenizerFactory` あるいは `TokenFilterFactory` を実装してから、schema.xml 内の `<tokenizer>` または `<filter>` タグで `<analyzer>` タグの一部として宣言しなければなりません。既存のアプリケーションの `Analyzer` がある場合は、それを `<analyzer>` タグの `class` 属性で宣言すれば使用できるようになります。他の Lucene アプリケーションで `Analyzer` を使用しようと計画しているのでない限り、`Analyzer` を新規に作成する必要はありません。schema.xml の `<analyzer>` タグを使って `Analyzer` を宣言するほうがずっと簡単だからです。

アプリケーションに特殊化されたデータが必要な場合に、そのデータを処理するための `FieldType` を追加したくなるかもしれません。例えば、レガシー・アプリケーションのバイナリー・フィールドを処理する `FieldType` を追加して、Solr で検索できるようにする場合には、単純に、`<fieldtype>` 宣言を使って `FieldType` を schema.xml に追加し、クラスパスで有効になっていることを確認するだけです。

パフォーマンスについての検討事項

Solr のパフォーマンスはそのままでもかなり優れていますが、さらに改善を図るためのヒントと秘訣があります。どんなアプリケーションでもそうですが、データ・アクセスに関するビジネスのニーズを慎重に検討することは、非常に大きな効果があります。例えば、索引付き `Field` を追加すればするほど必要なメモリーは増え、索引のサイズも大きくなるので索引の最適化に時間がかかるようになります。同様に、保管された `Field` を検索すると、入出力処理によってサーバーの処理速度が落ちてしまいます。フィールドの遅延ロードを使うか、あるいは大きなコンテンツは別の場所に保管すると、検索要求のために CPU を解放することができます。

検索について言うと、サポートするクエリーのタイプを検討することが重要です。大抵のアプリケーションでは、ワイルドカードの使用やその他の高度なクエリーをはじめ、Lucene のクエリー・パーサー構文の機能をフルに活用する必要はありません。ここで極めて有益な手段となるのは、ログを分析し、頻繁に実行されるクエリーが確実にキャッシュに入れられるようにすることです。また、共通のクエリーに `Filter` を使用すればサーバーの負荷が大幅に削減されることになります。あらゆるアプリケーションと同じく、アプリケーションを徹底的にテストすることによって、Solr がパフォーマンス要件を満たしていることを確実にできます。Lucene (および Solr) のパフォーマンスについての詳細は、「参考文献」に記載した ApacheCon Europe での私のスライド「Advanced Lucene」を参照してください。

前途有望な Solr の未来

Lucene の処理速度と長所をベースにした Solr は、エンタープライズ対応の非常に有効な検索ソリューションです。さまざまな大規模エンタープライズ環境ですでに Solr を採用している人たち

からなる活発で確固としたコミュニティーはすでに出来上がっています。また、Solr をサポートする献身的な開発者コミュニティーが常に Solr の改善方法を探し求めています。

この 2 回連載の記事では、Solr のそのまま使える索引付け機能と検索機能、そして機能の構成に使用される XML スキーマを紹介しました。また、Solr をほとんどすべてのエンタープライズ・アーキテクチャーに追加できるようにしている構成機能と管理機能についても説明しました。さらに、Solr を採用する際に必要なパフォーマンスについての検討事項、そして Solr を拡張するためのフレームワークについても理解してもらえたはずです。Solr についてもっとよく知りたい方は、「参考文献」の資料を参照してください。

ダウンロード

内容	ファイル名	サイズ
Sample Solr application	j-solr2.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。