Elasticsearch を Java アプリケーションの中で使用する

ハイパフォーマンスの RESTful な検索エンジン兼ドキュメント・ストアのクイックスタート・ガイド

Matt C. Tyson
Architect
Freelance

2016年 7月 07日

使いやすい REST API と、クラスターの自動スケーリングを組み合わせた Elasticsearch は、全文検索の分野を席巻しています。Elasticsearch をコマンド・シェルから、そして Java アプリケーション内から使用する方法について、この実践的な入門チュートリアルで学んでください。

Apache Lucene や Apache Solr を使用した経験があるとしたら、これらを使用するのはかなり大変な作業になる可能性があることはご存知でしょう。皆さんは、Lucene または Solr をベースにしたソリューションをスケーリングしなければならないとしたら尚更ですが、Elasticsearch プロジェクトを利用する動機については理解していることと思います。Elasticsearch (Lucene をベースに作成) は、デフォルトでクラスター構成のスケーリングをサポートした、簡単に扱えるパッケージで、ハイパフォーマンスの全文検索機能を提供します。Elasticsearch の操作には、標準的な REST API を使用することも、プログラミング言語固有のクライアント・ライブラリーを使用することもできます。

このチュートリアルでは、Elasticsearch が実際にどのように機能するかを紹介します。まず、コマンド・ラインから Elasticsearch にアクセスして、REST API の基礎を学びます。次に、ローカル Elasticsearch サーバーをセットアップして、単純な Java アプリケーションから Elasticsearch を操作します。サンプル・コードを入手するには、「ダウンロード」を参照してください。

前提条件

このチュートリアルのすべてのサンプルに取り組むには、システム上に Elasticsearch がインストールされている必要があります。お使いのプラットフォーム向けの最新版 Elasticsearch バンドルをダウンロードして、ダウンロードしたパッケージをアクセスしやすい場所に解凍してください。UNIX または Linux では、以下のコマンドを実行してインスタンスを起動します。

```
/elastic-search-dir/bin/elasticsearch
```

Windows では以下のコマンドを実行します。

```
/elastic-search-dir/bin/elasticsearch.bat
```

ロギング・メッセージに「started」と表示されるようになったら、ノードはリクエストを受け入れられる状態になっています。

Java のサンプルには、Eclipse と Apache Maven も必要になります。この 2 つがまだシステム上にインストールされていない場合は、ダウンロードして両方ともシステムにインストールされた状態にしてください。

さらに、cURL も必要です。Microsoft Windows では、Git Bash シェルを使用して cURL を実行します。

cURL を使用して REST コマンドを実行する

Elasticsearch に対しては cURL リクエストを発行できるので、コマンド・ライン・シェルから簡単にフレームワークを試すことができます。

> *“ Elasticsearch はスキーマレスです。渡されたものを何でも取り込んで、後でクエリーを実行できるような形で処理することができます。 ”*

Elasticsearch はスキーマレスです。つまり、渡されたものを何でも取り込んで、後でクエリーを実行できるような形で処理することができます。Elasticsearch ではあらゆるものがドキュメントとして保管されるので、最初の演習では歌詞が含まれるドキュメントを保管します。それにはまず、インデックスを作成するところから始めます。インデックスとは、すべてのドキュメント・タイプに使用されるコンテナーであり、MySQL などのリレーショナル・データベースに含まれるデータベースのようなものです。インデックスを作成したら、次はドキュメントをそこに挿入して、ドキュメントのデータに対してクエリーを実行できるようにします。

インデックスを作成する

Elasticsearch コマンドの一般的な形式は、`<REST ##> <###>:9200/index/doc-type` です。ここで、`<REST ##>` には `PUT`、`GET`、または `DELETE` のいずれかを使用します。(HTTP メソッドを明示的に指定する場合は、`curl -x<##>` プレフィックスを使用します)。

インデックスを作成するには、シェルで以下のコマンドを実行します。

```
curl -XPUT "http://localhost:9200/music/"
```

スキーマ・オプション

Elasticsearch はスキーマレスですが、内部では、スキーマを使用する Lucene が使用されています。ただし、Elasticsearch はこの複雑さをユーザーから隠します。実際のところ、Elasticsearch のドキュメント・タイプを単純なサブインデックス、あるいはテーブル名として扱うことができます。しかし、必要であればスキーマを指定できるので、Elasticsearch はスキーマをオプションで使えるデータ・ストアと見なすことができます。

ドキュメントを挿入する

`/music` インデックスの下にタイプを作成するには、ドキュメントを挿入する必要があります。この最初の例で扱うドキュメントには、「Deck the Halls」に関する (歌詞の行を含む) データを含めます (「Deck the Halls」は伝統的なクリスマス・ソングで、元々はウェールズ人の詩人 John Ceirog Hughes が 1885年に書いたものです)。

「Deck the Halls」のドキュメントをインデックスに挿入するには、以下のコマンドを実行します (このコマンドをはじめ、チュートリアルの他の cURL コマンドも 1 行で入力してください)。

```
curl -XPUT "http://localhost:9200/music/songs/1" -d '
{ "name": "Deck the Halls", "year": 1885, "lyrics": "Fa la la la la" }'
```

上記のコマンドは `PUT` 動詞を使用してドキュメントを `/songs` ドキュメント・タイプに追加し、このドキュメントの ID として 1 を割り当てます。URL パスには、「index/doctype/ID」を指定します。

ドキュメントを表示する

ドキュメントを表示するには、以下の単純な `GET` コマンドを使用します。

```
curl -XGET "http://localhost:9200/music/songs/1"
```

Elasticsearch は、先ほどインデックスに `PUT` で挿入した JSON コンテンツを返して応答します。

```
{"_index":"music","_type":"songs","_id":"1","_version":1,"found":true,"_source":
{ "name": "Deck the Halls", "year": 1885, "lyrics": "Fa la la la la" }}
```

ドキュメントを更新する

例えば、日付が誤っていたことに気付き、年の値を 1886年に変更しなければならないとします。この場合、ドキュメントを更新するには、以下のコマンドを実行します。

```
curl -XPUT "http://localhost:9200/music/lyrics/1" -d '{ "name":
"Deck the Halls", "year": 1886, "lyrics": "Fa la la la la" }'
```

このコマンドで使用している一意の ID は同じく 1 であるため、同じドキュメントが更新されます。

ドキュメントを削除する (今は削除しないでください)

ドキュメントはまだ削除しませんが、削除する方法を覚えておいてください。

```
curl -XDELETE "http://localhost:9200/music/lyrics/1"
```

ファイルからドキュメントを挿入する

技を使った方法をもう 1 つ紹介します。コマンド・ラインから、ファイルの中身を使用してドキュメントを挿入することができます。この方法で、別の伝統的な歌「Ballad of Casey Jones」に関するドキュメントを追加してみてください。そのためには、リスト 1 を caseyjones.json という

名前のファイルにコピーするか、サンプル・コードのパッケージ (「ダウンロード」を参照) に含まれている caseyjones.json ファイルを使用します。このファイルを、cURL コマンドを実行するのに便利な場所に置きます (コードのダウンロードでは、このファイルがルート・ディレクトリーに置かれています)。

リスト 1. 「Ballad of Casey Jones」の JSON ドキュメント

```
{
  "artist": "Wallace Saunders",
  "year": 1909,
  "styles": ["traditional"],
  "album": "Unknown",
  "name": "Ballad of Casey Jones",
  "lyrics": "Come all you rounders if you want to hear
The story of a brave engineer
Casey Jones was the rounder's name....
Come all you rounders if you want to hear
The story of a brave engineer
Casey Jones was the rounder's name
On the six-eight wheeler, boys, he won his fame
The caller called Casey at half past four
He kissed his wife at the station door
He mounted to the cabin with the orders in his hand
And he took his farewell trip to that promis'd land

Chorus:
Casey Jones--mounted to his cabin
Casey Jones--with his orders in his hand
Casey Jones--mounted to his cabin
And he took his... land"
}
```

このドキュメントを `music` インデックスに `PUT` するには、以下のコマンドを実行します。

```
$ curl -XPUT "http://localhost:9200/music/lyrics/2" -d @caseyjones.json
```

ついでに、walking.json という名前を付けたファイルに、リスト 2 の中身 (別のフォーク・ソング「Walking Boss」に関するもの) を保存してください。

リスト 2. 「Walking Boss」JSON

```
{
  "artist": "Clarence Ashley",
  "year": 1920
  "name": "Walking Boss",
  "styles": ["folk","protest"],
  "album": "Traditional",
  "lyrics": "Walkin' boss
Walkin' boss
Walkin' boss
I don't belong to you

I belong
I belong
I belong
To that steel driving crew

Well you work one day
Work one day
Work one day
Then go lay around the shanty two"
```

```
}
```

以下のコマンドで、このドキュメントをインデックスにプッシュします。

```
$ curl -XPUT "http://localhost:9200/music/lyrics/3" -d @walking.json
```

REST API に対して検索を行う

次は、基本的なクエリーを実行します。基本的なクエリーと言っても、前に「Deck the Halls」のドキュメントを見つけるために実行した単純な `GET` よりは高度なことを行います。この目的のために、ドキュメントの URL には `_search` エンドポイントが組み込まれています。歌詞に単語「you」が含まれる歌をすべて見つけるには、以下のクエリーを実行します。

```
curl -XGET "http://localhost:9200/music/lyrics/_search?q=lyrics:'you'"
```

`q` パラメーターはクエリー (query) を意味します。

上記のクエリーに対するレスポンスは以下のとおりです。

```
{"took":107,"timed_out":false,"_shards":{"total":5,"successful":5,"failed":0},"hits":{"total":2,"max
_score":0.15625,"hits":[{"_index":"music","_type":"songs","_id":"2","_
score":0.15625,"_source":{"artist": "Wallace Saunders","year": 1909,"styles":
["traditional"],"album": "Unknown","name": "Ballad of Casey Jones","lyrics": "Come all you rounders
if you want to hear The story of a brave engineer Casey Jones was the rounder's name.... Come all
you rounders if you want to hear The story of a brave engineer Casey Jones was the rounder's name
On the six-eight wheeler, boys, he won his fame The caller called Casey at half past four He kissed
his wife at the station door He mounted to the cabin with the orders in his hand And he took his
farewell trip to that promis'd land Chorus: Casey Jones--mounted to his cabin Casey Jones--with his
orders in his hand Casey Jones--mounted to his cabin And he took his... land"
}},{"_index":"music","_type":"songs","_id":"3","_score":0.06780553,"_source":{"artist": "Clarence
Ashley","year": 1920,"name": "Walking Boss","styles": ["folk","protest"],"album":
"Traditional","lyrics": "Walkin' boss Walkin' boss Walkin' boss I don't belong to you I belong I
belong I belong To that steel driving crew Well you work one day Work one day Work one day Then go
lay around the shanty two"}}]}}
```

他のコンパレーターを使用する

他にもさまざまなコンパレーターを使用することができます。例えば、1990年より前に作られたすべての歌を検索するには、以下のクエリーを実行します。

```
curl -XGET "http://localhost:9200/music/lyrics/_search?q=year:<1900
```

このクエリーにより、完全な「Casey Jones」ドキュメントおよび「Walking Boss」ドキュメントが返されます。

フィールドを制限する

結果に表示されるフィールドを制限するには、クエリーに `fields` パラメーターを追加します。

```
curl -XGET "http://localhost:9200/music/lyrics/_search?q=year:>1900&fields=year"
```

検索によって返されたオブジェクトを調べる

リスト 3 に、上記のクエリーによって Elasticsearch から返されたデータを示します。

リスト 3. クエリーの実行結果

```
{
    "took": 6,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "failed": 0
    },
    "hits": {
        "total": 2,
        "max_score": 1.0,
        "hits": [{
            "_index": "music",
            "_type": "lyrics",
            "_id": "1",
            "_score": 1.0,
            "fields": {
                "year": [1920]
            }
        }, {
            "_index": "music",
            "_type": "lyrics",
            "_id": "3",
            "_score": 1.0,
            "fields": {
                "year": [1909]
            }
        }]
    }
}
```

この結果には、Elasticsearch が複数の JSON オブジェクトを渡しています。最初のオブジェクトに含まれているのは、リクエストに関するメタデータです。ミリ秒単位でのリクエストの所要時間 (`took`) と、リクエストがタイムアウトしたかどうか (`timed_out`) を確認してください。`_shards` フィールドは、Elasticsearch がクラスター構成のサービスであるという点に関連します。この単一ノードのローカル・デプロイメントの場合でさえも、Elasticsearch は複数のシャードに論理的にクラスタリングされます。

引き続きリスト 3 の検索結果を見ていくと、`hits` オブジェクトには以下の要素が含まれています。

- `total` フィールド。取得された結果の数を示します。
- `max_score`。全文検索で関与してきます。
- 実際の結果。

実際の結果に `fields` プロパティーが含まれているのは、`fields` パラメーターをクエリーに追加したためです。そうでなければ、結果には `source` およびマッチするドキュメント全体が含まれるはずです。`_index`、`_type`、および `_id` は文字通りの内容であり、`_score` は全文検索のヒットの精度を表しています。これら 4 つのフィールドは、必ず結果の中に含まれて返されます。

JSON クエリー DSL を使用する

クエリーのストリングに基づく検索は、瞬く間に複雑化していきます。より高度なクエリーを実行するために、Elasticsearch には完全に JSON ベースのドメイン特化言語 (DSL) が用意されていま

す。例えば、`album` の値が `traditional` となっているすべての歌を検索するには、以下の内容の query.json ファイルを作成します。

```
{
    "query" : {
        "match" : {
            "album" : "Traditional"
        }
    }
}
```

その上で、以下のコマンドを実行します。

```
curl -XGET "http://localhost:9200/music/lyrics/_search" -d @query.json
```

Java コードから Elasticsearch を使用する

> “ プログラミング言語の *API* を介して *Elasticsearch* を使用すると、Elasticsearch の能力が最大限に発揮されます。 ”

プログラミング言語の API を介して Elasticsearch を使用すると、Elasticsearch の能力が最大限に発揮されます。ここでは Java API で Elasticsearch を使用する方法を紹介するために、Java アプリケーションから検索を行います。サンプル・コードを入手するには、「ダウンロード」セクションを参照してください。サンプル・アプリケーションは Spark マイクロフレームワークを使用するため、アプリケーションをセットアップするのは簡単です。

サンプル・アプリケーション

新規プロジェクト用のディレクトリーを作成してから、以下のコマンドを実行します (このコマンドは 1 行で入力します)。

```
mvn archetype:generate -DgroupId=com.dw -DartifactId=es-demo
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Eclipse の中で使用できるようにプロジェクトを生成するには、`cd` を実行し、Maven によって作成されたプロジェクト・ディレクトリーにカレント・ディレクトリーを変更した後、`mvn eclipse:eclipse` を実行します。

Eclipse で、「File (ファイル)」 > 「Import (インポート)」 > 「Existing Project into Workspace (既存プロジェクトをワークスペースへ)」の順に選択します。Maven を使用したフォルダーにナビゲートしてプロジェクトを選択し、「Finish (完了)」をクリックします。

Eclipse で、ルートの pom.xml ファイルと、com.dw.App.java メイン・クラス・ファイルが含まれる、基本的な Java プロジェクトのレイアウトを確認することができます。必要な依存関係は、pom.xml ファイルに追加します。リスト 4 に、完全な pom.xml ファイルを示します。

リスト 4. 完全な pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.dw</groupId>
```

```
  <artifactId>es-demo</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>es-demo</name>
  <url>http://maven.apache.org</url>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <compilerVersion>1.8</compilerVersion>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
    <groupId>com.sparkjava</groupId>
    <artifactId>spark-core</artifactId>
    <version>2.3</version>
</dependency>
<dependency>
    <groupId>com.sparkjava</groupId>
    <artifactId>spark-template-freemarker</artifactId>
    <version>2.3</version>
</dependency>
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>2.1.1</version>
</dependency>
  </dependencies>
</project>
```

リスト 4 で追加されている依存関係は、Spark フレームワークのコア、Spark Freemarker のテンプレート・サポート、そして Elasticsearch を取得します。`<source>` バージョンを Java 8 に設定している点にも注目してください。Spark には Java 8 が必要です (Spark はラムダを多用するため)。

皆さんはどうかわかりませんが、私は最近多くの RESTful アプリケーションを作成しているので、このアプリケーションでは気分を変えるために、サブミットしてロードするという従来の形の UI を使用します。

Eclipse のナビゲーターでプロジェクトを右クリックし、表示されるコンテキスト・メニューで「Configure (構成)」 > 「Convert to Maven Project (Maven プロジェクトに変換)」の順に選択して、Eclipse が Maven の依存関係を解決できるようにします。最後にプロジェクトを右クリックして、「Maven」 > 「Update Project (プロジェクトの更新)」の順に選択します。

Java クライアントの構成

Elasticsearch の Java クライアントは強力なので、必要に応じて組み込みインスタンスを起動し、管理タスクを実行することができます。しかしここでは、すでに実行中のノードに対してアプリケーション・タスクを実行する方法にフォーカスします。

Elasticsearch を使用して Java アプリケーションを実行する場合、選択できる動作モードは 2 つあります。それは、Elasticsearch クラスター内でアプリケーションがよりアクティブな役割を担う

か、よりパッシブな役割を担うかのどちらかです。「Node Client」と呼ばれる、よりアクティブな役割を担う場合、アプリケーション・インスタンスは通常のノードと同様に、クラスターからリクエストを受信して、そのリクエストを処理すべきノードを判断します (この動作モードでは、アプリケーションがインデックスをホストしてリクエストに対応することすらできます)。「Transport Client」と呼ばれる、もう一方の動作モードでは、最終的な宛先を決定する別の Elasticsearch ノードにすべてのリクエストを転送するだけに過ぎません。

Transport Clientを取得する

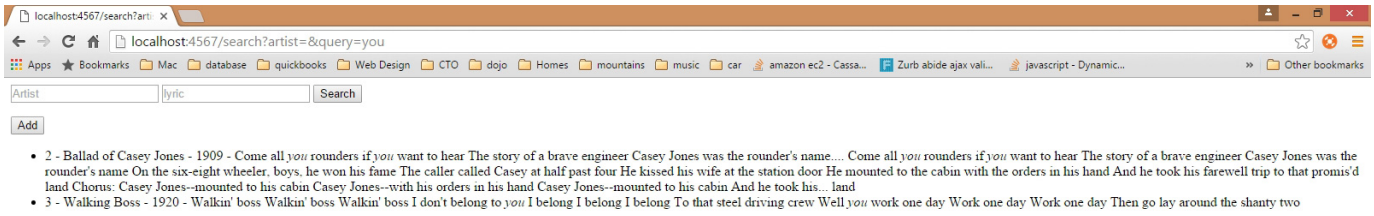
このデモ・アプリケーションでは、(App.java で行われる初期化によって) Transport Clientの役割を選んで、Elasticsearch の処理を最小限に抑えます。

```
Client client = TransportClient.builder().build()
   .addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("localhost"), 9300));
```

Elasticsearch クラスターに接続されている場合、ビルダーは複数のアドレスを受け入れることができます (この例で存在するのは、単一の localhost ノードだけです)。cURL から REST API にアクセスしたときはポート 9200 に接続しましたが、今回はポート 9300 に接続します。Java クライアントはこの特殊なポートを使用するため、ポート 9200 を使用すると機能しません (他の Elasticsearch クライアント (一例として、Python クライアント) はポート 9200 を使用して REST API にアクセスします)。

サーバーが起動したらクライアントを作成し、そのクライアントを、リクエストを処理するために一貫して使用します。Spark が Mustache テンプレート・エンジンの Java 実装を使用してページをレンダリングするとともに、Spark がリクエストのエンドポイントを定義しますが、これらの単純な使用ケースについて詳しく解説することはしません (Spark の詳細については、「参考文献」に記載されているリンクにアクセスしてください)。

アプリケーションの index ページに、Java クライアントの機能が提示されます。



UI の機能は以下のとおりです。

- 既存の歌のリストをレンダリングする
- 歌を追加するためのボタンを提供する
- アーティストと歌詞による検索を可能にする
- マッチした部分が強調表示された状態の結果を返す

検索して結果を処理する

リスト 5 では、ルート URL (/) が index.mustache ページにマッピングされています。

リスト 5. 基本的な検索

```
Spark.get("/", (request, response) -> {
        SearchResponse searchResponse =
            client.prepareSearch("music").setTypes("lyrics").execute().actionGet();
        SearchHit[] hits = searchResponse.getHits().getHits();

            Map<String, Object> attributes = new HashMap<>();
            attributes.put("songs", hits);

            return new ModelAndView(attributes, "index.mustache");
        }, new MustacheTemplateEngine());
```

リスト 5 で興味深い部分は、以下の行から始まります。

```
SearchResponse searchResponse = client.prepareSearch("music").setTypes("lyrics").execute().actionGet();
```

この 1 行には、検索 API の簡単な使用方法が示されています。つまり、`prepareSearch` メソッドを使用してインデックス (この例では `music`) を指定して、クエリーを実行するだけです。このクエリーは基本的に、「`music` インデックスに含まれるすべてのレコードを返す」よう指示しています。また、ドキュメント・タイプを `lyrics` に設定していますが、この単純な例では、インデックスにドキュメント・タイプは 1 つしか含まれていないため、ドキュメント・タイプを設定する必要はありません。サイズの大きいアプリケーションでは、ドキュメント・タイプの設定が必要になります。この API 呼び出しは、前に説明した `curl -XGET "http://localhost:9200/music/lyrics/_search"` という呼び出しに似ています。

`SearchResponse` オブジェクトには興味深い機能 (ヒット数、スコアなど) がありますが、目下必要なのは結果の配列だけです。結果の配列を取得するには、`searchResponse.getHits().getHits();` を使用します。

最後に、結果の配列をビュー・コンテキストに追加して、Mustache にレンダリングさせます。以下に、Mustache テンプレートを示します。

リスト 6. index.mustache

```
<html>
<body>
<form name="" action="/search">
  <input type="text" name="artist" placeholder="Artist"></input>
  <input type="text" name="query" placeholder="lyric"></input>
  <button type="submit">Search</button>
</form>
<button onclick="window.location='/add'">Add</button>
<ul>
{{#songs}}
  <li>{{id}} - {{getSource.name}} - {{getSource.year}}
    {{#getHighlightFields}} -
      {{#lyrics.getFragments}}
        {{#.}}{{{.}}}{{/.}}
      {{/lyrics.getFragments}}
    {{/getHighlightFields}}
  </li>
{{/songs}}
</ul>

</body>
</html>
```

高度なクエリーの実行とマッチした部分の強調表示

高度なクエリーの実行とマッチした部分の強調表示をサポートするには、`/search` を以下に示すように使用します。

リスト 7. 検索および強調表示

```
Spark.get("/search", (request, response) -> {
        SearchRequestBuilder srb = client.prepareSearch("music").setTypes("lyrics");

        String lyricParam = request.queryParams("query");
        QueryBuilder lyricQuery = null;
        if (lyricParam != null && lyricParam.trim().length() > 0){
            lyricQuery = QueryBuilders.matchQuery("lyrics", lyricParam);
        }
        String artistParam = request.queryParams("artist");
        QueryBuilder artistQuery = null;
        if (artistParam != null && artistParam.trim().length() > 0){
          artistQuery = QueryBuilders.matchQuery("artist", artistParam);
        }

        if (lyricQuery != null && artistQuery == null){
          srb.setQuery(lyricQuery).addHighlightedField("lyrics", 0, 0);
        } else if (lyricQuery == null && artistQuery != null){
          srb.setQuery(artistQuery);
        } else if (lyricQuery != null && artistQuery != null){
          srb.setQuery(QueryBuilders.andQuery(artistQuery,
              lyricQuery)).addHighlightedField("lyrics", 0, 0);
        }

        SearchResponse searchResponse = srb.execute().actionGet();

SearchHit[] hits = searchResponse.getHits().getHits();

    Map<String, Object> attributes = new HashMap<>();
    attributes.put("songs", hits);

    return new ModelAndView(attributes, "index.mustache");
}, new MustacheTemplateEngine());
```

リスト 7 で最初に注目すべき興味深い API の使用法は、`QueryBuilders.matchQuery("lyrics", lyricParam);` です。ここで、`lyrics` フィールドに対するクエリーを設定します。また、`QueryBuilders.andQuery(artistQuery, lyricQuery)` にも注目してください。クエリーの `artist` と `lyrics` の部分を 1 つの AND クエリーに結合するには、このようにします。

`.addHighlightedField("lyrics", 0, 0);` 呼び出しで Elasticsearch に対して指示しているのは、`lyrics` フィールドに対する検索でヒットしたものを、マッチした部分を強調表示した結果として生成することです。2 番目と 3 番目のパラメーターには、それぞれ無制限のサイズのフラグメント、無制限の数のフラグメントを指定しています。

検索結果がレンダリングされると、強調表示された結果が HTML に挿入されます。Elasticsearch は親切にも、`<em>` タグを使用してストリングがマッチした場所を強調表示する、有効な HTML を生成します。

ドキュメントを挿入する

プログラムによってドキュメントをインデックスに挿入する方法を見て行きましょう。リスト 8 に、ドキュメントの追加を処理するコードを示します。

リスト 8. インデックスへの挿入

```
Spark.post("/save", (request, response) -> {
      StringBuilder json = new StringBuilder("{");
      json.append("\"name\":\""+request.raw().getParameter("name")+"\",");
      json.append("\"artist\":\""+request.raw().getParameter("artist")+"\",");
      json.append("\"year\":"+request.raw().getParameter("year")+",");
      json.append("\"album\":\""+request.raw().getParameter("album")+"\",");
      json.append("\"lyrics\":\""+request.raw().getParameter("lyrics")+"\"}");

      IndexRequest indexRequest = new IndexRequest("music", "lyrics",
          UUID.randomUUID().toString());
      indexRequest.source(json.toString());
      IndexResponse esResponse = client.index(indexRequest).actionGet();

      Map<String, Object> attributes = new HashMap<>();
      return new ModelAndView(attributes, "index.mustache");
    }, new MustacheTemplateEngine());
```

このコードでは、`StringBuilder` によって直接 JSON ストリングを生成していますが、本番アプリケーションでは、Boon や Jackson のようなライブラリーを使用してください。

Elasticsearch の処理を行う部分は以下のとおりです。

```
IndexRequest indexRequest = new IndexRequest("music", "lyrics", UUID.randomUUID().toString());
```

この例では、UUID を使用して ID を生成しています。

まとめ

このチュートリアルでは、Elasticsearch をコマンド・シェルから、そして Java アプリケーション内で使用するための近道として、インデックスの作成、クエリーの実行、強調表示、マルチフィールド検索について学びました。Elasticsearch は、比較的簡単に使えるパッケージで、圧倒的な量の機能を提供してくれます。プロジェクトとしての Elasticsearch には、皆さんの興味をそそるような派生ツールがいくつかあります。特に、いわゆる ELK スタック (Elasticsearch、ログイン管理用の Logstash、そしてレポート作成/視覚化用の Kibana) は、その勢いを増しています。

ダウンロード

内容	ファイル名	サイズ
Sample code	es-demo.zip	15KB

著者について

Matt C. Tyson

チベットのヒマラヤ山脈をハイキングしていようと、最新のソフトウェア技術について調べていようと、Matt Tyson はその行為を崇高な冒険であると見なします。Matt がフルスタックの開発を行ってきた期間は、10 年を超えており、JavaWorld の「Jump into Java microframeworks」をはじめとするいくつかの記事の著者でもあります。