Apache Lucene と Solr による位置認識検索

非構造化テキストと空間データを組み合わせて検索アプリケーションを拡張する

Grant Ingersoll
Member, Technical Staff
Lucid Imagination

2010年 1月 12日

GPS 対応のスマートフォンで最寄りのコーヒー・ショップを検索したり、ソーシャル・ネットワーキング・サイトで近所の友達を探したり、あるいは市内で特定の製品を配送しているすべてのトラックの位置を確認したりと、ますます多くの人々とビジネスが位置認識検索サービスを利用するようになっています。これまでは、このような位置認識検索サービスを作成するのは、高価な専用ソリューションと地理空間の専門家たちの領域でしたが、最近になって、よく使われるオープンソースの検索ライブラリー Apache Lucene と、Lucene ベースの強力な検索サーバー Apache Solr に空間機能が追加されました。この記事では Lucene と Solr のコミッターである Grant Ingersoll が空間検索の基本を解説し、皆さんが次に開発する位置認識アプリケーションを空間検索機能によって強化する方法を説明します。

巷には位置の情報が溢れています。不動産では位置が何よりも重要であるのと同じく、検索に位置認識を組み合わせれば、ユーザーが情報を効率的かつ効果的に探す上で多大なメリットがもたらされるはずです。例えば、職業別電話帳のプロバイダー (「イエロー・ページ」のサイトなど) を考えてみてください。ユーザーが配管工を探しているとしたら、サイトはユーザーの自宅近くの配管工を検索結果として返すべきです。旅行サイトを運営している場合には、旅行者が旅行先周辺の見どころを検索できるようにし、滞在を楽しめるようなレジャーを見つけられるようにするとよいかもしれません。また、ソーシャル・ネットワーキング・サイトを構築しているとしたら、位置情報を使ってユーザーが友達とつながりを持てるようにしたいものです。世間には、カー・ナビゲーション・システムや GPS 対応カメラなどの位置認識機能を持った機器が普及しています。さらに、一般公開されたマップ・データも大量にあります。つまり、検索機能を統合した GIS (Geographical Information System) を作成し、エンド・ユーザーの期待に応えるような検索結果を提供する機会は溢れているということです。

検索に空間情報を使用するまでのことはありませんが、この記事では空間情報を生かし、Apache Lucene および Apache Solr の機能を利用して検索アプリケーションを拡張する方法に焦点を当てます。なぜ検索エンジンを使用するのでしょう？数多くの優れた (しかも無料の) GIS ツールが使用できることを考えれば必ずしも検索エンジンを使用する必要はありませんが、アプリケーションの基礎を検索に置くことで、これまで他の多くの手法では実現できなかった強力な機能を実

現できるからです。検索システムは、構造化データと非構造化データを組み合わせ、ユーザーがフリー・フォームのクエリーを入力して説明やタイトルなどのフリー・テキストを検索できるようにすると同時に、地理データに基づいて検索結果を制限、または修正するのに効果を発揮します。旅行サイトを例に取ると、ユーザーがマサチューセッツ州ボストンのすべてのホテルを検索する際に、検索基準として 4 つ星ホテルであること、「寝心地の良いベッド」というレビュー、そして「24 時間ルーム・サービス」という記述があることを指定すると、1 秒未満でレスポンスが返ってくるという機能を実装することができます。さらに Apache Solr のような検索システムには、ファセット検索 (Solr およびファセットについては「参考文献」を参照)、強調表示、そして検索結果でのスペル・チェックといった機能も用意されているため、ユーザーの検索を一層効果的に支援するアプリケーションを実現することができます。

この記事ではまず Lucene の重要な概念を簡単に説明します。さらに掘り下げた詳細については読者の皆さんが調査するのにおまかせします。続いて、地理空間検索の基本概念について説明します。GIS は、この記事の紙面には収まりきらないほど幅広い分野なので、サービス、人、その他の日常的な対象項目を検索する場合を前提として、かなり直観的な基本概念に焦点を絞ることにします。そして記事の締めくくりとして、Lucene と Solr を使用して空間情報に索引を付けて検索するために使用できるいくつかの手法を説明し、記事で説明した概念を、OSM (OpenStreetMap) プロジェクト (「参考文献」を参照) のデータを使用した実際の、ただし単純な例で具体化します。

Lucene の重要な概念について

Apache Lucene は、Java™ をベースとしたハイパフォーマンスの検索ライブラリーです。Apache Solr は、この Lucene を使用して、検索やファセットなどの数多くの機能を HTTP を介して提供する検索サーバーです。この 2 つはどちらも商用に適した Apache Software License の下、使用が許可されます。それぞれが提供する機能および API についての詳細は、「参考文献」を参照してください。

本質的に、Solr と Lucene はどちらもコンテンツを「文書」として表します。文書は 1 つ以上のフィールドからなり、オプションで文書の重要性を示す「ブースト値」が追加されます。フィールドには、索引付けまたは保存の対象となる実際のコンテンツと、Lucene にそのコンテンツの処理方法を指示するメタデータ、そしてそのフィールドの重要性を示すブースト値が含まれます。コンテンツをどのように文書およびフィールドとして表すかは、目的とする検索方法、あるいは文書内の情報にアクセスする方法に応じて決定することができます。コンテンツ単位の相互関係は、1 対 1 にすることも、1 対多にすることもできます。私自身は、Web ページについては、タイトルやキーワード、本文などのフィールドで構成された 1 つの文書として表すようにしています。本の場合には、各ページを個別の文書として表すこともあります。この後わかるように、この違いは、空間データを検索用にエンコードする際に重要となってきます。フィールド内のコンテンツには索引を付けることも、アプリケーションで使用するためにコンテンツをそのまま保存することもできます。コンテンツに索引を付けると、アプリケーションがその索引を検索できるだけでなく、コンテンツを分析して「Term」(通称「トークン」) を生成することもできます。Term は、基本的な検索単位として検索プロセスで使用されます。Term は単語であることがよくありますが、単語でなくてはいけないということはありません。以上の概念については、「参考文献」を調べて詳細を学んでください。

クエリーの面に関して言うと、Lucene と Solr は基本的なキーワード (Term) のクエリーから、フレーズ・クエリーおよびワイルドカード・クエリーに至るまで、ユーザーのクエリーを表現する

ための豊富な機能を提供しています。また、Lucene と Solr ではどちらも、1 つ以上のフィルターを適用して検索対象の範囲を制限できるようになっています。こうした点は、空間検索を行う上で重要になります。範囲を制限するのに不可欠なメカニズムには、レンジ・クエリーとレンジ・フィルターが含まれます。レンジ・クエリー (あるいはレンジ・フィルター) では、自然順ソートによる 2 つの値を指定し、その範囲内で文書を検索するように指定することができます。レンジ・クエリーがよく使用されるのは、昨年あるいは先月作成されたすべての文書を検索するような場合です。内部では、Lucene が文書に含まれる Term を列挙し、範囲内に収まっている文書を識別します。後で説明するように、この設定を効果的に行うことが、空間検索アプリケーションでのフィルタリング・パフォーマンスを左右する 1 つの鍵となります。

Lucene と Solr には、関数クエリーという概念もあります。関数クエリーでは、内部で収集した統計だけを使ってスコアリング・メカニズムの主要部分を構成するのではなく、フィールドの値 (緯度と経度など) をスコアリング・メカニズムの一部として使用することができます。この関数クエリーも、記事の中で Solr の距離ベースの関数を使用するときに活躍します、

地理空間検索の概念

空間検索アプリケーションを作成するときには、何よりもまず、アプリケーションに組み込む空間データのタイプを決定しなければなりません。空間データは大抵、緯度、経度、および高度というジオコード・システムの形式になっているか、あるいは ZIP コードまたは特定の住所です。エンコード・システムを形式化すればするほど、システム内でデータを扱いやすくなります。例えば、「Over the River and Through the Woods」という古いフォーク・ソング (「to Grandmother's house we go」という歌詞が続きます) では、かなりの空間情報が歌詞にエンコードされていますが (「参考文献」を参照)、この情報は GIS システムではまったく役に立ちません。この歌に出てくる森 (Woods) や川 (River) がどこに位置するのかを知る手掛かりは皆無だからです。この歌詞と、完全な住所によって示されるおばあちゃんの家 (Grandmother's house) への詳細な道順とを比較すれば、正確にエンコードされたデータがいかに重要であるかがわかるはずです (興味深い点として、システムがより一般的な道順と地理的な情報 (例えば、川の向こう、あるいは茶色い家の近くなど) を抽出して体系化し、それらの情報によって示される地点を推論できれば非常に役立つはずです。ただし、それに関する説明はこの記事ではしません)。

場所を特定する際に用いられるジオコード化されたデータだけではなく、多くの GIS システムは、実際の場所に関連して発生する情報を追加することができます。例えば、ナビゲーション・システムでは、地図上に順番に並べられた一連の場所を使って A 地点から B 地点に移動するためのルートを作成します。また、気象学者が降雨データや悪天候のデータを地域の地図に重ね、特定の場所の降雨量を検索できるようにすることもあります。人々が近隣の地域をグループ化して ZIP コードや局番を作ったり、さらには町、市、州を形成したりすることは珍しくありません。OSM の場合には、ユーザーが情報を編集し、基本となる地図の上にその情報 (対象の場所や番地など) を重ねることができます。これらの情報の層を組み合わせて相互関係を作り、その情報を追跡し続けることで、驚くほど動的で有能なアプリケーションを作成することができます。

空間データの表現

地図に重ねられた情報であるか、あるいは 1 つ以上の場所に関連付けられた情報であるかに関係なく、検索アプリケーションにはこうした情報を効率的に表現する方法が必要です。位置情報を表現するには何通りもの方法がありますが、ここでは Lucene に即した方法に焦点を絞ります。何よりもまず、地理空間データの多くは「そのまま」のフォーマットで表現することが可

能であり、検索アプリケーションで問題なく機能します。例えば、「`Syracuse`」は、Syracuse (シラキュース) 市を表現する方法としてまったく問題ありません。ユーザーが検索ボックスに「Syracuse」と入力すれば、他の検索用語とまったく同じように、Syracuse という言葉が含まれる文書が見つかります。実際、データを「そのまま表現する」という方法が、名前が付けられた場所 (市や州、そして ZIP コードなど) を表すために最もよく使用されている方法です。けれども注意してほしい点は、ここでは「そのまま表現する」と言っていますが、このようなデータでもあらかじめ変換、または正規化できることです。例えば、New York という記述をすべて NY に変換するのが極めて妥当なケースはよくあります。

Lucene で使用できる表現について説明する前に、すべての表現にはその表現を生み出した空間参照を考慮に入れなければならないことを理解しておくことが重要です (「参考文献」を参照)。米国で最も一般的な基準は、WGS 84 と略されている世界測地系 (World Geodetic System) です (「参考文献」を参照)。一部の測地系の間ではフォーマットの変換が可能ですが、すべてのデータを同じ 1 つの測地系で表現するに越したことはありません。ここからは、測地系は 1 つに統一されているという前提で説明します。

Lucene と Solr での検索で興味深い表現が登場してくるのは、緯度と経度 (省略形は lat/lon) などの数値による空間情報です。緯度と経度はグリニッジ子午線 (英国のグリニッジに位置) を基準とした度、分、秒で表現され、通常は精度を維持するために `double` (またはそれ以上) を使う必要があります。記事のサンプルに含まれているデータを例に取ると、the city of Syracuse, N.Y., United States (米国ニューヨーク州シラキュース市) は、76.150026 East (East が指定されていない場合は -76.150026) および 43.049648 North のあたりに位置します。

アプリケーションによっては、すべての緯度と経度をエンコードした結果、索引を付ける固有の Term が相当な数になることがあります。これは、必ずしも致命的な問題ではありませんが、検索速度は大幅に低下する可能性があります。したがって、この後説明するように、すべての緯度と経度をエンコードすることが常に必要であるとは限りません。実際、多くのマッピング・アプリケーションで重要となるのは、特定の地域に検索を制限することだけなので、その場所に関するおおよその情報を保存すれば、検索結果に悪影響を与えることなく Term の数を大幅に削減することができます。このように精度を犠牲にする場合、緯度と経度を (デカルト) 層に取り込むという手法がよく使われます。各層は地図の特定の部分のズームアップ・レベルだと思ってください。例えば米国全体を中心とした層 2 は北米すべてを網羅する一方、層 19 は誰かの家の裏庭となるといった具合です。さらに具体的に言えば、各層は地図を $2^{tier\ \#}$ のボックス、つまりグリッドに分割します。つまり、文書にボックスごとの番号と索引を指定できるということです。この情報を利用して検索を高速化する方法については、後のセクションで説明します。

緯度と経度は、Lucene の Term では 2 つの異なるフィールドとして表現されることがよくありますが、一部のアプリケーションでは、これがパフォーマンスに暗黙的意味を持つことがあります。単一のフィールドにしたい場合には、ジオハッシュ・エンコード方式 (「参考文献」を参照) を使って lat/lon を 1 つの `String` にエンコードすることもできます。ジオハッシュには、ハッシュの末尾から文字を削っていくことで、任意の精度で表現できるという利点があります。多くの場合、近隣にある 2 つの場所は共通のプレフィックスを持ちます。例えば、geohash.org に「`Syracuse, NY`」と入力した場合、ハッシュは `dr9ughxjkrt4b` となります。Syracuse 近郊の「`Cicero, NY`」を入力した場合のハッシュは `dr9veggs4ptd3` です。このように、この 2 つは `dr9` というプレフィックスを共有しています。

ここまで、さまざまな地点に絞って説明を進めてきましたが、多くの地理空間アプリケーションには地形やルート、そしてデータに含まれるその他の相互関係も関わってきます。けれども、これらの概念は Lucene および Solr には該当しないので、詳細については「参考文献」を参照してください。

空間データとテキストを組み合わせた検索

索引を付けてデータを表現した後、検索アプリケーションがデータを操作するためには、少なくとも以下の 5 つの基本的な処理が必要になります。

- **距離の計算**: ある地点を基準として、他の地点との距離を計算します。
- **境界ボックス・フィルター**: 特定の地域内で一致するすべて (すべての文書) を検索します。
- **ソート**: ある地点からの距離を基準に検索結果をソートします。
- **関連性による追加処理**: 距離をスコアのブースト係数として使用するとともに、他の係数もスコアに関与できるようにします。
- **クエリーの構文解析**: 索引を付けたデータの検索に使用できるように、住所、またはユーザーがその場所に関して指定したその他の情報を基にエンコードした表現を作成します。

上記の処理のそれぞれが、位置をベースとしたアプリケーションで重要な役割を果たしますが、とりあえずここで焦点とするのは、距離の計算、境界ボックスによるフィルタリング、そしてクエリーの構文解析です。ソートと関連性による追加処理 (ブースティング) は距離の計算だけを使用するので、これらの処理が実際にどのような役割を果たすかについては、記事の後のほうで説明します。

距離の計算

GIS アプリケーションで用いる距離を計算する際には、さまざまな計算手法があり、そのそれぞれに利点と欠点があることを理解することが重要です。距離の計算は、アプリケーションが地球をどのようにモデル化するかによって 3 つのグループに分類することができます。ある条件の下では、地球の平面体モデルを前提とし、速度と引き換えにある程度の精度を犠牲にしてもまったく問題はありません。平面体モデルでの距離計算のほとんどは、ピタゴラスの定理のさまざまな変形を使うことになります。2 地点間の距離が短く、その 2 地点が南極や北極の近くでなければ、この平面体モデルの手法で十分有効です。そうでない場合は、球面体モデルが使用されることもあります。球面体モデルで使用される主な距離計算は大圏距離です (「参考文献」を参照)。大圏距離では球面上の 2つの地点の最短距離が計算されるので、距離が大きく離れていて、より正確な精度が求められる場合には球面体モデルが適しています。残された最後のモデルである地球楕円体モデルは、楕円上で極めて正確な距離 (0.5 ミリメールまでの精度) を求めるために Vincenty の式 (「参考文献」を参照) と併せて使用することができますが、ほとんどのアプリケーションにとって、ここまで複雑な計算を行う価値はありません。

インチを指定するか、マイルを取るか

近隣を検索するアプリケーションでは、多くの場合、どれだけの精度が必要となるかは、それぞれのアプリケーション次第です。マイル単位の精度でまったく問題ないこともあれば、数ミリメートルでさえも問題になることもあります。例えば、検索範囲が広範にわたる場合 (2 つの州にまたがる場合など)、ユークリッド距離の精度では対応しきれません。半正矢 (大圏) 手法の精度でも、場合によっては不十分です。それは、地球は球体ではなく、楕円体としてモデル化したほうが正確だからです。そのような場合には、Vincenty の式を使用する意味があります。一方、唯一問題になるのは結果の順序付けだけというアプリケーションもあ

ります。そのようなアプリケーションではユークリッド平方距離 (実際には距離ではありません) のような手法を使用して、平方根の計算を省略することができます。

当然、他の距離尺度も役に立つことはよくあります。その一例は、マンハッタン距離です。マンハッタン距離は、(タクシーがニューヨーク市のマンハッタンを運転するときのように) 碁盤の目のように区分された都市で移動するときの距離を反映します。この記事では、地球の平面体モデルと大圏距離を使用した距離の例を用い、これ以外の距離尺度については読者の皆さんが調査するのにおまかせします。また、記事では高度を係数として考慮していませんが、アプリケーションによっては高度を考慮に入れる必要もあることに注意してください。地理学上の距離についての詳細は、「参考文献」を参照してください。

境界ボックス・フィルター

位置をベースとしたアプリケーションの多くでは、数百万件の位置データを検索できるようになっています。検索キーワードが含まれていて、なおかつ、ユーザーが指定した地点から一定の距離内にあるような文書一式を見つけるためだけに、このすべてのデータを繰り返し処理するとなると、極めて長い時間がかかることになります。そこで然るべき対策として、最初に特定の文書セットに絞り込んだ上で、関連するサブセットを評価するという方法が考えられます。緯度と経度の情報だけが保存されているとしたら、文書セットを絞り込むための一番の方法は、特定の地点の周辺地域を包含した範囲を渡すことです。この方法を図 1 に図解します。この図に示されたやや不透明なボックスが表すのは、サウスカロライナ州チャールストンのダウンタウンを囲む境界ボックスです。

図 1. サウスカロライナ州チャールストンのダウンタウンを中心とした境界ボックスの例

さらに、アプリケーションが層情報またはジオハッシュ情報を使用している場合には、これらの値を使用して、さらに効率的に検索対象の文書数を絞り込むことができます。その方法については、Lucene と Solr での索引付けと検索についての詳細を説明するときに、例を用いて説明します。

クエリーの構文解析

クエリーの構文解析によって、クエリーのどの部分に検索キーワードが含まれていて、どの部分に位置情報が含まれているのかがわかります。後者は、ジオコーディングと呼ばれるプロセスです (「参考文献」を参照)。ここではクエリーの構文解析のコンテキストでジオコーディングを説

明しますが、ジオコーディングは索引付けの際にも役立ちます。例として、ユーザーが挙げた以下のクエリーについて考えてみましょう。

- 1600 Pennsylvania Ave. Washington, DC という住所
- 1 Washington Av. Philadelphia Pennsylvania という住所
- 60 East Broadway Bloomington, MN 55425 にある Mall of America というショッピング・モール
- Mall of America の近くにあるレストラン
- Mall of America 内にあるレストラン

このなかで最初に挙げた 2 つのクエリーを検討すると、興味深い点がいくつか浮き彫りになります。

- Term の順序は必ずと言っていいほど重要です。その一方、純粋なテキスト・ベースの検索では順序が重要でないこともあります。
- GeoNames (「参考文献」を参照) などの空間リソースや Gazetteer (地名辞典) は、住所を位置に変換する上で極めて役に立ちます。これらのリソースには大抵、対象となる場所の一覧が含まれています (例えばホワイト・ハウスのようなランドマークなど)。
- ユーザーが入力した住所情報の中で使われている表現が多岐にわたってしまうことに対応するには、Ave. や DC などの略語を正規化するか、同じ概念を表す特定の語を使用することが重要になります。

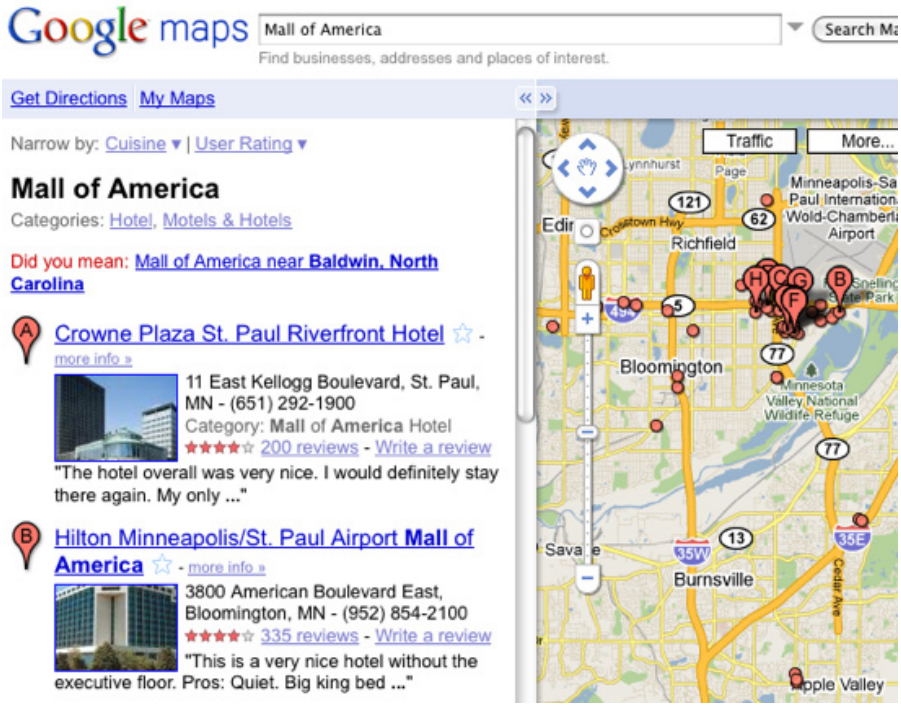
残る 3 つのクエリーは、その他の微妙な点をいくつか明らかにしています。例えば 3 番目のクエリーで、ユーザーは完全な住所を指定していますが、名前、住所、市、州、ZIP コードのそれぞれに対応するフィールドを検索するとしたら、これらのプロパティーを慎重に構文解析で抽出しなければなりません。

最後の 2 つのクエリーでは、ユーザーが特定の地点の近くを選んでいるのか、それとも特定の地点そのものを選んでいるのかという微妙な違いが重要になります。4 番目のクエリーでは、ショッピング・モールから一定の距離の範囲内にあるレストランであれば、このクエリーを挙げたユーザーが提示した条件を満たしていますが、最後のクエリーを挙げたユーザーが知りたいのは、ショッピング・モール内にあるレストランだけです。スペル・ミスや言語の曖昧さ、そして誤ったデータは言うまでもなく、位置との関係で対象を記述する難しさから、クエリーの構文解析はかなりの難問になり得ます。

このようにジオコーディングにはさまざまな困難が伴うとは言え、住所を位置に変換するサービスは存在します。なかでもよく使用されているサービスは、Google マップで公開されている API と GeoNames の 2 つです (「参考文献」を参照)。残念ながら、これらの Web サービスを使用する場合、それぞれの使用条件 (使用の制限が含まれることがよくあります) に縛られると同時に、インターネットのトラフィックに関する問題を伴うことになります。したがって、実際の本番システムには独自の機能を実装したほうが上手くいくはずです。そのような実装についての話題は、この記事では取り上げませんが、GeoNames のデータはすべて無料でダウンロードできること、そして CIA Factbook (「参考文献」を参照) をはじめとする他の多くの空間リソースでも同じく無料でデータをダウンロードできることを気に留めておいてください。優れたリソースを利用できるのであれば、基本情報 (住所、市、州) を基礎として、対象となる場所と堅牢な例外処理を組み込むに越したことはありません。時が経つと、ユーザーが何を入力しようともグレースフルに処理する堅牢なクエリー・パーサーを作成する上で、クエリー・ログが極めて役に立つことが明ら

かになるはずです。あらゆる検索アプリケーションと同様、最も妥当な推測を行う一方で、ユーザーに検索対象を明確にするよう求めることが妥当であることを忘れないでください。図 2 に示す Google マップのスクリーン・キャプチャーがその一例です。

図 2. Google マップでの最も妥当な推測とユーザーに検索対象を明らかにするよう求めるリクエスト



この記事では基本的なクエリー・パーサーの例として、GeoNames サービスを使用し、その他少数の機能を備えたクエリー・パーサーについて説明しますが、本番に対応可能なバージョンの実装については、読者の皆さんにお任せします。ここまでの説明で、皆さんには作業に取り掛かるだけの十分な基礎知識が身に付いているはずです。記事の残りでは、Lucene と Solr を使用して空間情報に索引を付け、検索する方法に話題を絞ります。

サンプル・コードのインストール

サンプル・コードを実行するには、以下のものがインストールされている必要があります。

- JDK 1.5 以降
- Ant 1.7 以降
- 最新の Web ブラウザー (Firefox など)

さらに、この記事のサンプル・コードも必要です (「ダウンロード」を参照)。このサンプル・コードに、Apache Solr とその依存関係が含まれています。サンプル・コードをインストールするには、以下の手順に従ってください。

1. `unzip sample.zip` を実行します。
2. `cd geospatial-examples` を実行します。
3. `ant install` を実行します。

4. `ant start-solr` を実行して Solr を起動します (後で Solr を停止するときには、`ant stop-solr` を実行します)。
5. ブラウザーで `http://localhost:8983/solr/admin` にアクセスし、Solr が正常に起動されていることを確認します。基本管理画面には、クエリーを実行するためのボックスが表示されているはずです。

Solr が稼働状態になったら、Lucene の空間データでの最初のステップに取り掛かれます。インストール手順を実行すると、OSM プロジェクトから生成されたサンプル・データ (http://people.apache.org/~gsingers/spatial/ に用意してあります) がダウンロードされます。このデータには、私が興味を持っている米国内の以下の 4 つの場所のサンプル OSM データが含まれています (OSM へのパーマリンクはファイルにリストされています)。

- Syracuse, N.Y. (ニューヨーク州シラキュース)
- Downtown Minneapolis, Minn. (ミネソタ州ミネアポリスのダウンタウン)
- Around the Mall of America in Bloomington, Minn. (ミネソタ州ブルーミントンにある Mall of America の周辺地域)
- Downtown Charleston, S.C. (サウスカロライナ州チャールストンのダウンタウン)

この記事で取り上げている概念の多くを説明するために、OSM データに索引を付けるためのコードを Solr に書き込み、単純なファクト・データを特定の場所に関連付けておきました (例として、data ディレクトリーに置かれた syracuse.facts ファイルを見てください)。この作業の目的は、非構造化テキストと空間データを組み合わせて、効果的な検索アプリケーションを作成する方法を示すことです。もう 1 つ注意する点として、この作業では最近リリースされた Solr 1.4 ではなく、Solr 1.5-dev バージョンを使用しています。

Lucene で空間情報に索引を付ける方法

Lucene 2.9 には、空間検索で重要な役割を果たす 2 つの特徴が追加されました。まず 1 つは、数値によるレンジ・クエリー機能とフィルタリング機能をより優れた実装として提供したことです。この 2 つの機能は、境界ボックスの手法でよく使われます。2 つ目の特徴として、Lucene には新たに contrib モジュールが追加されています。このモジュールには、これまで Local Lucene と呼ばれていたスタンド・アロンのプロジェクト (「参考文献」を参照) が含まれています (コードは Lucene の contrib/spatial にあります。JAR ファイルは、サンプル・コードに組み込んでおきました)。空間 contrib は、デカルト層とジオハッシュ・コードを作成するためのツール、そして Lucene のクエリーおよびフィルター・オブジェクトを作成するためのツールを提供します。

データに索引を付けるコードについて説明する前に、お使いのアプリケーションでは、どのような方法でデータを操作しようとしているのか、また処理しなければならないデータ量はどれぐらいなのかを決めておくことが重要です。例えば、文書の数が少量から中程度 (例えば 1000 万未満) であれば、緯度と経度に索引を付け、単純なレンジ・クエリーを使用するだけで十分なパフォーマンスを得られます。その一方、大規模なアプリケーションの場合には、フィルタリングしてスコアを付ける Term および文書の数を減らすために、より具体的な対策 (デカルト層を追加するなど) が必要になってきます。また、情報をどのフォーマットで保存するかを考慮することも重要です。多くの空間距離アルゴリズムでは数値の単位がラジアンになっていなければなりませんが、度を単位として使えるアルゴリズムもあります。したがって、検索のたびに lat/lon の値を変換するのではなく、索引付けの際に lat/lon の値をラジアンに変換しておくと功を奏するかもしれません。もちろん、両方のフォーマットが必要な場合には、その代償としてより多くの保存スペース

(ディスクや、場合によってはメモリー) が必要になります。最後にもう 1 つ、位置の情報を使って単純にフィルタリングするだけなのか、あるいはファセットを使用して位置の情報をソートしてスコアを付けるのかを決定してください。後者の場合には、代わりに使用する表現も必要になる可能性があります。

この記事の目的は概念を具体的に説明することで、コードを本番で使用することではないため、サンプル Java コードでジオハッシュ、デカルト層、緯度、経度に一度にまとめて索引を付ける方法を説明します。そこで、OSM データを取得するための数々の値を Solr スキーマ (geospatial-examples/solr/conf/schema.xml に配置) に定義しておきました。リスト 1 に、位置に関する主なフィールドを記載します。

リスト 1. Solr スキーマのサンプル

```
<!-- Latitude -->
   <field name="lat" type="tdouble" indexed="true" stored="true"/>
<!-- Longitude -->
   <field name="lon" type="tdouble" indexed="true" stored="true"/>
<!--
   lat/lon in radians
   In a real system, use a copy field for these instead of sending over the wire -->
   <field name="lat_rad" type="tdouble" indexed="true" stored="true"/>
   <field name="lon_rad" type="tdouble" indexed="true" stored="true"/>
<!-- Hmm, what about a special field type here? -->
   <field name="geohash" type="string" indexed="true" stored="true"/>
<!-- Elevation data -->
   <field name="ele" type="tfloat" indexed="true" stored="true"/>
<!-- Store Cartesian tier information -->
   <dynamicField name="tier_*"  type="double" indexed="true"  stored="true"/>
```

Lucene と Solr

ここでは索引を付けるフィールドを説明するために Solr スキーマを使用していますが、このすべての概念は、Lucene でも容易に使うことができます。例えば `tdouble` は、Lucene 2.9.1 では精度ステップ 8 の `NumericField` に置き換えられます。

lat/lon の値は、`tdouble` フィールドとして保存しています。`tdouble` は内部でトライ (Trie) 構造を使って表現される `double` です。実際にはさらに多くの Term を索引に追加するという事実に反し、Lucene はこのフィールドを使って、範囲の計算 (境界ボックス) を行う際に評価する必要のある Term の数を大幅に減らすことができます。その一方で、ジオハッシュを単純な (分析されない) `string` として保存している理由は、ジオハッシュでは完全一致を行わなければならないためです。高度については、厳密に言うと、ここで行っているような計算には必要ありませんが、`tfloat` として保存します。これは、トライ構造に保存される単なる `float` にすぎません。最後の `tier_*` という動的フィールドによって、アプリケーションはデカルト層フィールドをそれぞれ事前に宣言することなく、動的に追加することができます。この索引付けプロセスで収集されるその他のメタデータ・フィールドについては、読者の皆さんが調べてください。

データに索引を付けるためのコードは、sample.zip ファイルのソース・ツリーに置かれています。`Driver` クラスは索引付けプロセスを起動するためのコマンドライン・ユーティリティーです。実際の索引付けは、`OSMHandler` という SAX の `ContentHandler` 実装の一部として行われます。`OSMHandler` コードのなかで注目すべき重要な行は、`startElement()` メソッドに含まれています。これからこれらの行を 3 つのセクションに分けて説明します。リスト 2 に記載する最初の例では、緯度と経度に `double` の索引を付けており、その間に単位をラジアンに変換しています。

リスト 2. 緯度/経度に索引を付けるサンプル

```
//... current is a SolrInputDocument
double latitude = Double.parseDouble(attributes.getValue("lat"));
double longitude = Double.parseDouble(attributes.getValue("lon"));
current.addField("lat", latitude);
current.addField("lon", longitude);
current.addField("lat_rad", latitude * TO_RADS);
current.addField("lon_rad", longitude * TO_RADS);
```

このように、lat/lon に索引を付けるのは至って簡単です。次に、lat/lon のペアに対応するジオハッシュに索引を付けます (リスト 3 を参照)。

リスト 3. ジオハッシュに索引を付けるサンプル

```
//...
//See http://en.wikipedia.org/wiki/Geohash
String geoHash = GeoHashUtils.encode(latitude, longitude);
current.addField("geohash", geoHash);
```

リスト 3 のジオハッシュ・コードでは、Lucene の空間 contrib パッケージに提供されている `GeoHashUtils.encode()` メソッド (このメソッドに対応する `decode()` メソッドもあります) を使用して lat/lon のペアを 1 つのジオハッシュ・ストリングに変換しています。Solr には、この変換後のジオハッシュ・ストリングを追加します。最後にデカルト層のサポートを追加するために、`OSMHandler` コードで以下の 2 つの操作を行いました。

- コンストラクターに、`CartesianTierPlotter` クラスの `n` インスタンス (索引を付けたい層ごとに 1 つのインスタンス) をセットアップしました。
- `startElement()` メソッドで `n` プロッターをループ処理し、現行の OSM 要素の緯度と経度を含む各グリッド要素の ID を取得しました。このコードは、リスト 4 のようになります。

  ### リスト 4. デカルト層に索引を付けるサンプル

  ```
  //...
  //Cartesian Tiers
  int tier = START_TIER; //4
  //Create a bunch of tiers, each deeper level has more precision
  for (CartesianTierPlotter plotter : plotters)
     {current.addField("tier_" + tier, plotter.getTierBoxId(latitude, longitude));
  tier++;
  }
  ```

一般に、クエリーは 1 度に 1 つの層に対して行えばよいため、複数の層があっても問題になることはありません。必要な層の数は、検索の粒度をどれだけ高くするかに応じて選んでください。この索引付けコードの残りの部分に目を通すと、OSM ファイル内のデータ・ポイントに関連するさまざまメタデータ値が追加されていることがわかります。現時点で索引を付けている OSM データには、node と way の 2 つのタイプしかありません。node とは単に、特定の緯度と経度の位置のことで、way とは、(例えばストリート名や町名など) 何らかの形で相互に関連するノードの組み合わせのことです (OSM ファイルについて詳しく学ぶには、「参考文献」に記載されている OSM Data Primitives リンクを参照してください)。

CartesianTierPlotter について

`CartesianTierPlotter` の役目は、地球の投影法 (私の場合、サンソン図法を使用しています。「参考文献」を参照) と lat/lon 情報を受け取り、層システムで使用するグリッドに変

換して、それぞれのグリッド・セルに対して固有の番号を割り振ることです。アプリケーションは検索の際に、このグリッド ID を指定して検索対象を絞り込むことができます。

空間情報が含まれる Solr 文書を作成する基本的な方法が理解できれば、あとはこの文書を実行するだけの話です。`Driver` クラスはデータとファクト・ファイル、そして Solr が実行されている URL を引数に取り、`OSM2Solr` クラスにその処理を任せます。`OSM2Solr` クラスは Solr の Java クライアントである SolrJ を使用して、SAX パーサー `OSMHandler` によって作成された文書を取得し、これらの文書をまとめて Solr サーバーに送信します。`Driver` クラスはコマンドラインを使って実行することもできますが、`ant index` を実行すれば、Ant がこのドライバーを実行するために必要な処理を行ってくれます。ドライバーが実行されたら、ブラウザーで http://localhost:8983/solr/select/?q=*:* にアクセスし、Solr が 68,945 の文書を検出したことを確認します。ここで返されてきた結果をじっくり読み、そのコンテンツをよく理解してください。

ここまでは、OSM データを処理する無数の方法のうち、ほんのわずかな部分に触れただけですが、ここからは、このデータをアプリケーションで利用する方法について説明します。

位置を基準に検索する方法

索引にデータが用意できたところで、今度はこのデータを使用する各種の方法を再検討します。すなわち、索引の空間情報に基づいて文書をソート、ブースト、フィルタリングする方法です。

距離関連の計算

距離を基準にブースティングを行って文書をソートするのは、多くの空間アプリケーションにとって一般的な要件です。これを受けて、Lucene と Solr には距離を計算するためのさまざまな機能が備わっています (「参考文献」を参照)。Lucene には大圏 (半正矢) 式に基づいてソートとフィルタリングを行うツールが組み込まれている一方 (`DistanceUtils` および `DistanceFieldComparatorSource` を参照)、Solr には以下に示すような、距離を計算する複数の FunctionQuery 関数があります。

- 大圏 (半正矢およびジオハッシュ半正矢)
- ユークリッドおよびユークリッド平方
- マンハッタン距離および他の p-ノルム

距離に基づくブースティングは、Solr の距離関数を使用することで、ごく簡単に行えます。最も簡単に使用できること、そしてプログラミングを利用する必要がないことから、ここでは Solr の関数クエリーに焦点を絞りますが、これらの関数クエリーは Lucene でも簡単に使用することができ、また Lucene への移植も簡単に行えます。

前に説明したように、OSM データを保存するための `lat/lon`、`lat_rad/lon_rad`、`geohash` などのフィールドはセットアップ済みなので、以下の値を検索してブースティングを行うことができます。

- `hsin` (大圏): `http://localhost:8983/solr/select/?q=name:Minneapolis AND _val_:"recip(hsin(0.78, -1.6, lat_rad, lon_rad, 3963.205), 1, 1, 0)"^100`
- `dist` (ユークリッド、マンハッタン、p-ノルム): `http://localhost:8983/solr/select/?q=name:Minneapolis AND _val_:"recip(dist(2, lat, lon, 44.794, -93.2696), 1, 1, 0)"^100`

- `sqedist` (ユークリッド平方): `http://localhost:8983/solr/select/?q=name:Minneapolis AND _val_:"recip(sqedist(lat, lon, 44.794, -93.2696), 1, 1, 0)"^100`
- `ghhdist` (ジオハッシュ半正矢): `http://localhost:8983/solr/select/?q=_val_:"recip (ghhsin(geohash(44.79, -93), geohash, 3963.205), 1, 1, 0)"^100`

上記のそれぞれの例で、キーワード・クエリーと、距離に基づく `FunctionQuery` とを組み合わせ、キーワード・スコアと距離スコアを計算に入れた結果セットを生成しています。部分ごとの効果を確認するには、各クエリーに `&debugQuery=true` を追加し、Solr によって生成される説明をじっくり検討してください。上記は単なるクエリーの使用例にすぎません。これらの関数クエリーやその他の `FunctionQuery` 関数の完全なシグニチャーおよびドキュメントについては、「参考文献」を参照してください。もちろん、ある部分を他の部分に優先してブースティングを行うことも、必要に応じて呼び出しを変更することもできます。

距離を基準としたソートに関して、Solr は主要な方法を 1 つ提供しています。この方法は、Solr には関数を基準としたソート機能がまだないこと、そして現時点ではカスタムの `FieldType` が定義されていないことに対する次善策のようなものです。けれどもこの次善策は複雑なものではありません。関数を基準にソートするには、上記のようなクエリーを作成し、ただしキーワード節は (ブーストにより) ゼロで埋めます (例えば、`q=name:Minneapolis^0 AND _val_:....`)。これで、キーワード・スコアはゼロとなり (ただしそれでも、一致する文書は返されます)、関数の値がスコアの唯一のコンポーネントとなります。やがては、Solr が `FieldType` を追加して、メインのクエリーにゼロを設定することなく、ソートをより効果的にサポートできるようになるはずです。

ソートとスコアについての説明は以上です。続いてフィルタリングの説明に移ります。

フィルタリング

Solr で位置を基準にフィルタリングする場合、アプリケーション作成者は表 1 に記載する 3 つの主要なメカニズムを使用して文書の範囲を制限することができます。

表 1. フィルタリング手法

フィルタリング手法	説明	例
範囲	境界ボックスの lat/lon を含むレンジ・フィルターを作成します。パフォーマンス上の理由により、この手法では Solr の `TrieField` (`NumericField`) 機能を使用することが重要です。	`http://localhost:8983/solr/ select/?q=*:*&fq=lon:[-80 TO -78]&fq=lat:[31 TO 33]`
デカルト層	lat/lon および距離に基づき、中心点をとりまくグリッド・セルを特定して、これらのセルが含まれる文書に範囲を絞り込みます。このソースの実装についての詳細は、「`QParserPlugin` について」を参照してください。	`http://localhost:8983/solr/ select/?q=*:*&fq={! tier x=32 y=-79 dist=50 prefix=tier_}`
距離	Solr の `frange` (関数範囲) の `QParserPlugin` 機能と距離関数 (「上記の「距離関連の計算」に詳細を説明) を使用して、2 地点間の距離を判断し、文書の範囲を制限します。	`http://localhost:8983/solr/ select/?q=*:*&fq={! frange l=0 u=400}hsin(0.57, -1.3, lat_rad, lon_rad, 3963.205)`

密度について一言

特定の範囲内にあるデータの密度が、ユーザーの検索エクスペリエンスにおいて重要な役割を果たすことはよくあります。例えば、ニューヨーク州ニューヨーク市のマンハッタン

内のビジネスを検索できるようにしているアプリケーションのデータ密度は、ミネソタ州バッファローを検索範囲とするアプリケーションのデータ密度を遥かに上回ります (「参考文献」を参照)。実際、適切な検索結果が得られるようにするには、効果的な距離をアプリケーションが選択できるように、データ密度の情報をフィルタリング関数に組み込むようにすると有効です。しかし残念ながら、この方法についての説明はこの記事では行いません。

どの手法が適切であるかは、その検索対象とする地点のデータの密度によりますが (「密度について一言」を参照)、まずは単純な範囲の手法を使用し、必要に応じて層の手法に移行することをお勧めします。ここで重要な要素となるのは、範囲を計算するときに毎回評価する必要のある Term の数を調べることです。この数が、結果セットを絞り込むために Lucene が行う処理の量を直接左右します。

単純な geonames.org クエリー・パーサー

この記事では、空間を対象とした本格的なクエリー・パーサーを作成することはしませんが、代わりに、GeoNames から位置情報の結果を取得するという作業をこなす単純な `QParserPlugin` を作成します。このパーサーで前提となるのは、アプリケーションがユーザー入力を事前にキーワード・クエリーと空間クエリーの 2 つに分割することです。実際、近隣を検索するアプリケーションの多くは、ユーザーにこのように入力するよう求めるために 2 つの入力ボックスを使います。

`QParserPlugin` について

`QParserPlugin` とは、Solr で言うクエリー・パーサー・プラグイン・モジュールのことです。Solr の多くの構成要素と同じく、クエリー・パーサーの実装もプラガブルになっています。この記事で使用している 3 種類のクエリー・パーサー・プラグインのうち、1 つは Solr に付属のプラグイン (`FunctionRangeQParserPlugin` (`{!frange}`) です。他の 2 つ (`CartesianTierQParserPlugin` (`{!tier}`) と `GeonamesQParserPlugin`) は私が作成しました。この 2 つのプラグインのソースは、サンプル・ダウンロードの src ツリーの配下に置かれていて、どちらも、Solr では既に solrconfig.xml ファイルで構成済みになっています。`QParserPlugin` をクエリーで呼び出すには、例えば `{!tier x=32 y=-79 dist=50 prefix=tier_}`、`{!frange l=0 u=400}hsin(0.57, -1.3, lat_rad, lon_rad, 3963.205)` のように、`{!parserName [パラメーター]}[クエリー]` を指定します (このうち、パラメーターとクエリーはオプションです)。

このパーサーは以下のパラメーターを取ることができます。

- `topo`: toponym の略 (GeoNames マニュアルを参照)。GeoNames で検索する位置を指定します。これは必須パラメーターです。
- `rows`: GeoNames から取得する行の数を指定します。これはオプションのパラメーターで、デフォルトは 1 です。
- `start`: 結果の開始位置を指定します。これもオプションのパラメーターで、デフォルトは 0 です。
- `lat`: `FunctionQuery` の `ValueSource` として使用する緯度フィールドの名前を指定します。このパラメーターを指定する場合は、`lon` も併せて設定する必要があります。
- `lon`: `FunctionQuery` の `ValueSource` として使用する経度フィールドの名前を指定します。このパラメーターを指定する場合は、`lat` も併せて設定する必要があります。
- `gh`: `FunctionQuery` の `ValueSource` として使用するジオハッシュ・フィールドの名前を指定します。このパラメーターを指定する場合は、`lat`/`lon` も併せて設定する必要があります。
- `dist`: 使用する距離関数を指定する `String` です。`[hsin, 0-Integer.MAX_VALUE, ghhsin]` のいずれかを指定します。ジオハッシュ・フィールドが指定されている場合は、このフィール

ドの値は無視され、自動的に `ghhsin` が使用されるようになります。デフォルトは、2-ノルム (ユークリッド) を表す 2 です。
- `unit - KM|M`: 使用する単位を指定します。`KM` はメートル、`M` は英語を表します。デフォルトは `M` です。
- `boost - float`: 関数クエリーのブースト係数を指定します。デフォルトは 1 です。

このサンプルのコードは、サンプル・ダウンロードの GeonamesQParserPlugin.java に含まれています (Solr サーバーは、ダウンロードに含まれている Solr バージョンに構成済みです)。このコードを呼び出す方法は、上記で概説している `CartesianTierQParserPlugin` を呼び出す方法とほとんど同じです。例えば、ミネソタ州ブルーミントンに関連する索引でショッピング・モールを検索するには、`http://localhost:8983/solr/select/?q=text:mall AND _query_:"{!geo topo='Bloomington, MN' lat=lat_rad lon=lon_rad dist=hsin}"` とします。

`QParserPlugin` の手法を使うことで、自分にとって重要な特定の構文だけを処理できる一方、位置に関しては、あらゆるテキスト・ベースのクエリー構文解析機能を通常通りに行うことが可能になりました。

今後、`GeonamesQParserPlugin` は郵便番号やその他多くの位置の指定を処理するように大幅に拡張されることが考えられます。当然、エラー処理を一層充実させる必要があることは言うまでもありません。さらに、Web サービス呼び出しに依存しないように、GeoNames データ・セット (「参考文献」を参照) を使用するように変換する必要もあるでしょう。Solr にしても同じく、空間クエリー・パーサーのサポートを追加するための問題追跡機能という問題が残っています (「参考文献」を参照)。

次のステップは？

この記事では、ある地点をベースにした位置モデルに基づいてテキスト文書を検索、ソート、フィルタリングする Lucene と Solr の機能について説明しました。真の近隣検索アプリケーションであれば、ここから先でユーザー・クエリーのスケーリングと処理、そして結果の可視化を最適に行う方法を調査しなければなりません。スケーリングについての問題は、境界ボックス・フィルターを作成するときに列挙しなければならない Term の数によって答えが出るものもありますが、スケーリングに対応したフィルタリングの問題という枠を超えて、例えば索引を分散させるか、あるいは単純に複製するかどうかなどの検索に関する一般的な要素も関わってくるので、Lucene および Solr の参考文献を参照してください。

価値の高い GIS アプリケーションの構築を目指しているとしたら、ルートを検索できるようにしたり、地形の情報を重ね合わせたり、さらに多くの高度な機能を追加しなければならないでしょう。けれども、ある地点をベースにした位置の構造に非構造化テキストを組み合わせた堅実な検索アプリケーションを構築できればよいというのであれば、Lucene と Solr がその必要を満たしてくれます。

謝辞

この記事について洞察をし、レビューを行ってくれた Lucene/Solr コミッターの Ryan McKinley、Yonik Seeley の両氏に深く感謝いたします。

ダウンロード

内容	ファイル名	サイズ
Spatial examples	j-spatial.zip	52.4 MB

著者について

Grant Ingersoll

Grant Ingersoll は Lucid Imagination の設立者であり、技術スタッフの 1 人でもあります。彼は情報取得、機械学習、テキストのカテゴリー分け、抽出などなどのプログラミングに関心を持っています。彼は Apache Mahout 機械学習プロジェクトの共同設立者の 1 人であり、Apache Lucene プロジェクトと Apache Solr プロジェクト両方のコミッターでもあり、講演者でもあります。また彼は自然言語処理のためのオープンソース・ツールを解説する『Taming Text』(Manning より出版予定) の共著者の 1 人でもあります。