geocoder.ja における
ジオコーディング方法

２００７．１．５

小林一英

１．概要

この資料はオープンソースのジオコーダ「goecoder.ja」において、住所から緯度経度に変換する方法について解説したものである。

住所変換の主な流れは次のとおりである。

１）市区町村の特定

その住所がどの市区町村の住所かを特定する。特定できた場合、結果を市区町村コードで表す。

２）町・大字名の特定

特定できた市区町村の中の、どの町・大字かを特定する。特定できた場合、結果を町大字コードで表す。

３）丁目・字名の特定

特定できた町・大字の中のどの丁目または字かを特定する。特定できた場合、結果を丁目字コードで表す。

４）地番・街区符号の特定

特定できた丁目・字のどの地番または街区かを特定する。特定できた場合、結果を地番街区符号コードで表す。

５）枝番・住居番号の特定

地番に枝番がある場合は枝番を、また住居表示エリアでは住居番号を特定する。特定できた場合、結果を枝番住居番号コードで表す。

６）住所コードの生成

上記の処理により住所を順次特定し、特定できた段階までの各コードをまとめて住所コードとして表す。

７）緯度経度の取得

得られた住所コードに対応する緯度経度を取得する。

８）住所の正規化

住所を一定の書式で表したい場合は、住所コードを住所文字列に変換する。

以上の流れでは、大きく、住所文字列を住所コードに変換する処理と、住所コードを緯度経度や住所文字列に変換する処理とに分ける事ができる。元の住所文字列を、一旦、住所コードに変換する事により、住所文字列を単に緯度経度に変換するという用途だけでなく、色々な検索、変換に利用しやすい構造となっている。

２．住所検索方法

2.1　住所検索上の問題

１）住所文字列に区切りが無い

住所は都道府県名、市区町村名、町大字名、丁目・字名、地番と言うように階層的に表

される。住所を緯度経度に変換するというジオコーディングなど、住所に関する処理をコンピュータ上で行う場合、住所を各階層毎に分け、それぞれの階層において住所を特定していかねばならない。

この階層に分ける際に問題になるのが、各階層を区切る明確な指標が無い事である。例えば、仮に、「八日市市川町」という住所があった場合、これは「八日市の市川町」なのか、「八日市市の川町」なのか、住所文字列だけでは判定ができない。判定するためには、市の名称に「八日市」や「八日市市」が存在するかどうかを調べなければならない。

２）県名や郡名の省略

また、別の問題として、その市区町村の名前が全国で一つしか無い場合、県名や郡名などが省略される事がある。省略されない場合は住所の最初は都道府県という前提で検索ができるが、省略される事があると、まず住所の最初の文字列は都道府県名なのか郡名なのか市区町村名なのかの判定が必要となる。この判定では、「小県郡」や「郡山市」など判定に利用できそうな文字（県や郡など）が各階層の名称に含まれるので、単純な方法では判定できない。

３）字体の違い

さらに住所検索を難しくするのが、「富が丘」「富ヶ岡」「冨ケ岡」など住所文字列の字の形が違う場合である。正式な町名以外に、同じ読みや形のよく似た字を使って住所が表される場合もあり、それらも同じ町名と判断しなければならない。

以上の他に、「丸の内一丁目２番３号」を「丸の内１－２－３」と表すなど表記上の問題もいくつかある。

これらの問題は、プログラムで一つ一つ対処する事はできるが、色々な場合があるので対処方法が複雑になり、処理に時間がかかるのと、正常にマッチングができない問題となる。

2.2　頭２文字ハッシュ法

上記の問題に対応するため、このジオコーダ（geocoder.ja）では、ハッシュ法による検索を行っている。ハッシュ法は、文字列を一定の変換方法で整数（ハッシュ値）に変換し、２つの文字列の一致をハッシュ値で判定する方法である。

このジオコーダで採用しているハッシュ法は次のとおりである。

１）各階層地名のハッシュ値計算

都道府県名や市区町村名、町大字名など、各階層の地名についてそのハッシュ値を計算しデータベース化する。その際、地名の頭２文字についてのみハッシュ値を計算する。また、都道府県名、支庁名、郡名、市区町村名は、行政区域名として一まとめにしてデータベース化する。

現在使用しているハッシュ関数は、漢字２文字を４バイトの符号なし整数とみなして、その整数を関数値としている。

地名の字体の違いに対処するために、「富が丘」「富ヶ岡」「冨ケ岡」など字体を変えた地名のハッシュ値を計算し、そのハッシュ値に対応する地名を同じ地名に対応するようにデータベース化している。この字体の変換をどの字について行うかは、geocoder.jaのソースコードと同時に配布されるgcj_variation.csvというファイルで設定している。

２）住所文字列のハッシュ値計算

ジオコーディングする住所文字列が与えられた時、まず先頭２文字のハッシュ値を計算する。そして、そのハッシュ値に対応する行政区域名をデータベースで検索する。

対応する行政区域名が複数ある場合、即ち頭２文字が一致する行政区域名、例えば、「京都府」と「京都市」などの場合は、それぞれの名称が与えられた住所文字列の先頭の文字列と完全に一致するか調べ、完全に一致した文字列をマッチした行政区域名とする。対応する行政区域名が一つの場合は、その行政区域名がマッチした行政区域名となる。

次にマッチした行政区域名を住所文字列の先頭から取り去り、上記と同じ操作を続ける。住所が省略しないで書かれている場合は、県名、郡名、町村名といった各段階でマッチングを行う事になる。そして、マッチした行政区域名が、政令指定都市では区名、その他は市町村名まで達したら、行政区域名の検索が完了する。

以上の検索で必要な処理ステップは,当ジオコーダでは数回の演算とメモリアクセス、比較であり、一般的な方法に比べ、高速な処理が可能となっている。また、都道府県名などがあってもなくても、また字体が違っていても、殆ど検索速度に影響を与えない効率的な検索ができるようになっている。

市区町村名が特定できた後は、その市区町村の町大字名に関して同様にハッシュ値を計算し町大字を特定する。

３）ハッシュ値を２文字で計算している理由

このジオコーダで、ハッシュ値を２文字で計算している理由は次のとおりである。

１）住所文字列には県名や市町村名の区切りとなる文字が無く、どこまでを単独の地名とするかが分からない。そのため、ハッシュ値の計算を多くの文字で行うと複数の地名のハッシュ値を計算する事になってしまう。通常、地名は２～３文字が多いので、２ないしは３文字で計算すれば単独の地名である確率が高く、マッチング処理が簡単になる。

２）ハッシュ関数として、２文字であれば４バイトの整数として扱う事で処理が簡単であり、また計算速度が速い。

以上の理由で２文字で計算しているが、３文字で行った場合との比較はしていないので、２文字がベストかどうかは分からない。

2.3　検索効率

一つのハッシュ値に複数の文字列が対応する事をハッシュの衝突というが、このジオコーダでの衝突が起きている程度は1.5程度である。この数字は一つのハッシュ値に平均いく

つの文字列が対応しているかの数値で、1.5 というのは衝突が全然無いかあっても１回の衝突であり、１回の比較でどちらか判定できる事を示している。行政区域名の数は全国では 2000 程度あり、バイナリサーチの場合は 11 回程度の比較が必要となるので、このハッシュによる方法は十分効率的と言える。但し、住所の他の部分ではハッシュを使っても速度が上がらない場合もあり、全体を通してハッシュの速さが生かされる訳ではない。

３．ジオコーディング処理

このジオコーダで住所を緯度経度に変換する手順は次のとおりである。

3.1　市区町村の特定

ジオコーディングする住所文字列（以下単に住所文字列）がどの市区町村の住所かを最初に特定する。

住所文字列は場合によって、都道府県名や郡名が無かったり、名称の字体が違ったりするが、それらの対処も含めて、上記「２．住所検索方法」で述べた方法により、市区町村名の特定が可能である。

特定ができると、どの市区町村かを５桁の市区町村コードで表す。このジオコーダで使っている市区町村コードは JIS で決められた市区町村コードに準拠しているが、郡名に関して、北海道の一部などで、独自のコードを割り当てている場合もある。

3.2　町大字名の特定

住所文字列がどの市区町村か特定できた後、住所文字列から市区町村名までの部分文字列を取り去る。そうすると残っている住所文字列は、一般的には町大字名が先頭に来ている。ただ、町大字名の無い町村もあり、その場合は残りの文字列は地番のみの表現となっている。

このジオコーダの住所データベースでは、町大字名は市区町村別に管理されている。それぞれの町大字名は、頭２文字についてのハッシュ値が計算され、ハッシュ値をインデックスととして、町大字名が特定できるようになっている。

それで次の処理としては、まず残りの住所文字列の頭２文字のハッシュ値を計算する。そしてそのハッシュ値に対応する町大字名を取得する。同じハッシュ値に対応する町大字名が複数ある場合は、それぞれの町大字名が住所文字列に完全にマッチするか判定する。この場合の判定では頭２文字でなく、その町大字名の全ての文字について一致するかをチェックする。そして完全に一致したものが求める町大字名である。

特定できた場合、どの町大字かを町大字コードで表す。この町大字コードは当ジオコーダで独自に割り当てたコードで、３桁程度の数字である。

3.3　丁目・字名の特定

町大字名まで特定でき、住所文字列から町大字部分を取り除くと、残りの住所文字列は多くの場合、丁目や字名が先頭に来ている。次はこの丁目や字名の特定を行う。

住所の検索は確率としては都市部での検索が多い。また、都市部では町大字名の次は丁目の場合が多く、その丁目の数も平均すると 10 以下である。比較する数が多くないので、このジオコーダでは、丁目・字名の検索は順次検索を行っている。即ち、単純にその町大字に属する丁目や字名と、住所文字列の先頭の文字列を比較し、完全に一致する丁目・字名を探している。

但し、丁目の場合、「一丁目２番３号」を「１－２－３」と表記したりするので、漢数字を算用数字にしたり、“丁目”や“－”を区切り文字として数字を取り出し、数字だけで比較をするなどの処理を行っている。

特定できた場合、どの丁目・字かを丁目字コードで表す。この丁目字コードは当ジオコーダで独自に割り当てたコードで、２桁程度の数字である。

3.4　地番・街区符号の特定

丁目・字名が特定でき、住所文字列から丁目・字名部分を取り除くと、残りの住所文字列は地番または街区符号が先頭に来ている。

地番・街区符号部分は殆どの場合、数字で始まるのだが、偶に,“甲、乙、A、左、右”などの文字で始まる地番もある。そのため、最初に、この区別をし、文字の場合は文字列で完全一致するかどうかを比較する。数字の場合は丁目の場合と同じようにして数字で比較する。この数字の比較については、丁目と違い、比較する数が多いので、バイナリサーチをすべきだが、現在の geocoder.ja のバージョンでは行ってなくて、順次検索で行っている。

特定できた場合、どの地番・街区符号かを地番街区符号コードで表す。この地番街区符号コードは当ジオコーダで独自に割り当てたコードで、２桁程度の数字である。このコードが正数の場合、数値は地番または街区符号の数字そのものである。負数の場合は、対応する地番・街区符号が数字でなく文字列である事を表し、数値は文字列を表すコードになっている。

3.5　枝番・住居番号の特定

地番・街区符号まで特定できると、残りは枝番・住居番号の特定のみとなる。この枝番・住居番号の表記方法は地番・街区符号と構造的には同じなので、特定方法も同じ方法で行っている。

特定できた場合、地番・街区番号と同様の方法で、結果を枝番・住居番号コードとして表す。

3.6　住所コードの生成

上記の各段階の処理により、住所文字列が階層毎に特定され、特定結果が各階層毎のコード番号で表される。このジオコーダでは、それらのコード番号の集まりを一まとめにして処理し、住所コードという名前で呼んでいる。住所コードには各コード番号の他に、どの段階までマッチングができたかを示すレベル番号も格納されている。レベル番号は都道府県レベルを１とし、枝番・住居番号レベルの８までの８レベルとなっている。

3.7　緯度経度の取得

当ジオコーダの住所データベースには、都道府県から枝番・住居番号までの各階層の住所データが格納されているが、それらは住所コードでアクセスできるようになっている。

住所文字列を当ジオコーダで解析し、住所コードに変換できると、その住所コードから住所データベースにアクセスできるようになる。そして、住所データベースには住所コードに対応する点の位置座標が緯度経度で格納されているので、住所文字列が示す点の緯度経度を取得する事ができる。

当ジオコーダでは、緯度経度の測地系は元の住所データの測地系のままであり、使用する元データにより測地系が変わるので、必要な場合は測地系の変換をアプリ側で行うようにする。因みに、元データが国土交通省の街区データの場合は世界測地系であり、（株）インクリメントＰ製のデータの場合は東京測地系となっている。

3.8　住所の正規化

実際に使用されている住所の表記方法は様々であり、コンピュータで処理する上で表記方法を統一したい場合がある。また、書類や郵便物などに印刷する際、できるだけ省略のない表記方法で行いたい場合もある。

このような場合、元の住所文字列をジオコーダで住所コードに変換し、その住所コードから、再度、住所文字列を作成するという手順を取る事で、住所の表記方法を統一できる。このような住所表記の統一を住所の正規化と呼んでいる。

当ジオコーダでは正規化の際に出力形式を指定できるようになっていて、用途に応じた表記方法の統一が可能である。

以上。