jageocoder

リリース *2.1.5.post1*

Takeshi Sagara

2024 年 04 月 17 日

目次

Jageocoder は日本の住所ジオコーダーの一つです。辞書と Python 解析用モジュールをローカルマシンにインストールすると、オフラインでの住所ジオコーディングを行なうことができます。

```
>>> import jageocoder
>>> jageocoder.init()
>>> jageocoder.search('新宿区西新宿2-8-1')
{'matched': '新宿区西新宿2-8-', 'candidates': [{'id': 5961406, 'name': '8番', 'x': 139.
↪691778, 'y': 35.689627, 'level': 7, 'note': None, 'fullname': ['東京都', '新宿区', '西新
宿', '二丁目', '8番']}]}
```

第 1 章

動作環境

Python 3.7 以上がインストールされた、 Linux, Windows, MacOS で動作します。

第 2 章

ライセンス表示

第 3 章

目次

3.1 クイックスタート

ここでは Python 3.7 以上がインストール済みの Linux, Windows, MacOS 上に、 jageocoder をインストールして基本的なジオコーディング処理を行なうまでの一連の作業を示します。

以下の手順では省略していますが、 venv などを使って仮想環境を作成することをお勧めします。また、 python コマンドは環境によって python3 に読み替えてください。

3.1.1 インストール

pip (または pip3) コマンドで jageocoder パッケージをインストールします。次に、住所辞書をダウンロードしてインストールします。

```
$ pip install jageocoder
$ jageocoder download-dictionary https://www.info-proto.com/static/jageocoder/latest/v2/
↪jukyo_all_v21.zip
$ jageocoder install-dictionary jukyo_all_v21.zip
```

より詳細なインストール手順やアンインストールの方法については**インストール手順** を参照してください。

3.1.2 コマンドラインでジオコーディング

Jageocoder は Python プログラム内から呼びだすパッケージとして利用することを想定していますが、コマンドライン・インタフェースから利用することもできます。

```
$ jageocoder search '新宿区西新宿２－８－１'
{"matched": "新宿区西新宿２－８－", "candidates": [{"id": 12977785, "name": "8 番", "x": 139.
```

(次のページに続く)

(前のページからの続き)

```
↪691778, "y": 35.689627, "level": 7, "priority": 3, "note": null, "fullname": ["東京都",
↪"新宿区", "西新宿", "二丁目", "8 番"]}]}
```

3.1.3 ジオコーディングの簡単なコード

Python プログラムから呼びだしてジオコーディングを行ないます。

```
>>> import json
>>> import jageocoder
>>> jageocoder.init()
>>> results = jageocoder.search('新宿区西新宿２－８－１')
>>> print(json.dumps(results, indent=2, ensure_ascii=False))
{
  "matched": "新宿区西新宿２－８－",
  "candidates": [
    {
      "id": 12977785,
      "name": "8 番",
      "x": 139.691778,
      "y": 35.689627,
      "level": 7,
      "priority": 3,
      "note": null,
      "fullname": [
        "東京都",
        "新宿区",
        "西新宿",
        "二丁目",
        "8 番"
      ]
    }
  ]
}
```

Python コードから jageocoder を利用するより詳しい方法は**コードサンプル** を参照してください。

3.2 インストール手順

3.2.1 パッケージのインストール

pip コマンドでインストールできます。

```
(.venv) $ pip install jageocoder
```

バージョンを指定したい場合は == に続けてバージョン番号を指定してください。

```
(.venv) $ pip install jageocoder==2.1.0
```

3.2.2 住所データベースファイルのインストール

住所データベースファイルは` zip 形式でダウンロード可能です。最新の住所データベースファイル一覧から、必要な地域を含むものを選択してダウンロードしてください。

住所データベースは jageocoder のバージョンによってフォーマットが異なるため、v1 用のファイルは利用できません。v2.1 に対応する住所データベースファイルは末尾が *_v21.zip* です。

```
(.venv) $ jageocoder download-dictionary https://www.info-proto.com/static/jageocoder/
↪latest/v2/jukyo_all_v21.zip
```

注意 住所データベースファイルは圧縮した状態でもファイルサイズが大きいため、ダウンロード・インストールには時間がかかります。

次にダウンロードした zip ファイルをインストールします。

```
(.venv) $ jageocoder install-dictionary jukyo_all_v21.zip
```

3.2.3 アンインストール手順

jageocoder をアンインストールする場合、先に住所データベースを削除してください。住所データベースの場所が分かっている場合はそのディレクトリごと削除しても構いませんが、 `uninstall-dictionary` コマンドを利用すると簡単に削除できます。

```
(.venv) $ jageocoder uninstall-dictionary
```

その後、 jageocoder パッケージを pip でアンインストールしてください。

```
(.venv) $ pip uninstall jageocoder
```

3.2.4 住所データベースディレクトリを指定する

住所データベースは、特に指定しない場合 Python 環境内に作成されます（参考： **住所辞書ディレクトリの取得**）。

このデータベースは数 GB のサイズがあるため、複数の Python 環境で jageocoder を利用する際には共用したいことがあります。そのような場合には、環境変数 JAGEOCODER_DB2_DIR をセットすると、住所データベースのディレクトリを指定することができます。

```
(.venv) $ export JAGEOCODER_DB2_DIR=$HOME/jageocoder/db2
(.venv) $ jageocoder get-db-dir
/home/sagara/jageocoder/db2
```

ただし jageocoder のバージョンは住所データベースのバージョンと互換性がある必要があります。

3.3 コマンドライン・インタフェース

ここでは jageocoder モジュールをコマンドラインから呼びだして利用する方法を説明します。

同じ内容はオンラインヘルプでも確認できます。

```
(.venv) $ jageocoder -h
```

3.3.1 ジオコーディング

住所文字列をキーとして住所辞書データベースを検索し、先頭から最長一致すると解釈できるレコードを取得し、そのレコードの経緯度を含む情報を返します。

最長一致する長さが同じ候補が複数存在する場合、全ての候補を返します。

コマンド

search

パラメータ 住

所文字列

オプション

-d デバッグ情報を表示します。

--area=<area> 検索する都道府県や市区町村を指定します。省略した場合は全国を対象とします。複数の都道府県・市区町村を指定する場合は , で区切ります。

--db-dir=<dir> 住所辞書データベースを配置したディレクトリを指定します。

実行例

```
# 「落合１－１５－２」を検索します。
# 「栃木県下都賀郡壬生町落合一丁目 15 番 2 号」
# 「広島県広島市安佐北区落合一丁目 15 番 2 号」
# 「東京都多摩市落合一丁目 15 番地 2」
# 「広島県広島市安佐北区落合一丁目 15 番地 2」が返ります。
(.venv) $ jageocoder search '落合１－１５－２'
{"matched": "落合１－１５－２", "candidates": [{"id": 38651481, "name": "2 号", "x": 139.
↪82020568847656, "y": 36.450565338134766, "level": 8, "priority": 4, "note": "",
↪"fullname": ["栃木県", "下都賀郡", "壬生町", "落合", "一丁目", "15 番", "2 号"]}, {"id":␣
↪106341148, "name": "2 号", "x": 132.51043701171875, "y": 34.47321319580078, "level": 8,
↪"priority": 4, "note": "", "fullname": ["広島県", "広島市", "安佐北区", "落合", "一丁目",
↪"15 番", "2 号"]}, {"id": 50058048, "name": "2", "x": 139.4287567138672, "y": 35.
↪62576675415039, "level": 8, "priority": 7, "note": "", "fullname": ["東京都", "多摩市",
↪"落合", "一丁目", "15 番地", "2"]}, {"id": 106341162, "name": "2", "x": 132.5104217529297,
 "y": 34.47317123413086, "level": 8, "priority": 7, "note": "", "fullname": ["広島県", "広
島市", "安佐北区", "落合", "一丁目", "15 番地", "2"]}]}

# ``--area`` オプションで検索対象を東京都に限定し、
# 「落合１－１５－２」を検索します。
# 「東京都多摩市落合一丁目 15 番地」だけが返ります。
(.venv) $ jageocoder search --area=東京都 '落合１－１５－２'
{"matched": "落合１－１５－２", "candidates": [{"id": 50058048, "name": "2", "x": 139.
↪4287567138672, "y": 35.62576675415039, "level": 8, "priority": 7, "note": "", "fullname
↪": ["東京都", "多摩市", "落合", "一丁目", "15 番地", "2"]}]}
```

3.3.2 逆ジオコーディング

経度・緯度を指定し、その地点を囲む３点のレコードを検索し、そのレコードリストの情報と地点からの距離を指定地点に近い順に返します。

岬の突端や離島など、３点のレコードが存在しない場合はリストに含まれるレコード数が３より小さい場合もあります。

住所データベースには「住所に対応する代表点」の座標しか含まれておらず、その住所が表す範囲（形状）の情報は利用できないため、「指定した地点に近い代表点」を検索していることに注意してください。

コマンド

reverse

パラメータ

経

度 緯度 （WGS1984 の十進度表記）

オプション

-d デバッグ情報を表示します。

--level=<level> 指定した住所レベルまで検索します。デフォルトは 6 (字レベル) です。

--db-dir=<dir> 住所データベースを配置したディレクトリを指定します。

実行例

```
# 経度 139.6917, 緯度 35.6896 の地点付近の住所を検索します。
# 「東京都新宿区西新宿二丁目」、「東京都新宿区西新宿六丁目」、
# 「東京都新宿区西新宿四丁目」が返ります。
(.venv) $ jageocoder reverse 139.6917 35.6896
[{"candidate": {"id": 50372222, "name": "二丁目", "x": 139.6917724609375, "y": 35.
↪689449310302734, "level": 6, "priority": 2, "note": "aza_id:0023002/postcode:1600023",
↪"fullname": ["東京都", "新宿区", "西新宿", "二丁目"]}, "dist": 17.959975373852735}, {
↪"candidate": {"id": 50373915, "name": "六丁目", "x": 139.6909637451172, "y": 35.
↪693424224853516, "level": 6, "priority": 2, "note": "aza_id:0023006/postcode:1600023",
↪"fullname": ["東京都", "新宿区", "西新宿", "六丁目"]}, "dist": 429.5116877067265}, {
↪"candidate": {"id": 50372614, "name": "四丁目", "x": 139.6876220703125, "y": 35.
↪687538146972656, "level": 6, "priority": 2, "note": "aza_id:0023004/postcode:1600023",
↪"fullname": ["東京都", "新宿区", "西新宿", "四丁目"]}, "dist": 434.2648526035473}]

# 経度 139.6917, 緯度 35.6896 の地点付近の住所を
# 街区（◯番）レベルで検索します。
# 「東京都新宿区西新宿二丁目 8 番」、「東京都新宿区西新宿二丁目 10 番」、
# 「東京都新宿区西新宿四丁目 15 番」が返ります。
(.venv) $ jageocoder reverse 139.6917 35.6896 --level=7
[{"candidate": {"id": 50372232, "name": "8 番", "x": 139.6917724609375, "y": 35.
↪68962860107422, "level": 7, "priority": 3, "note": "", "fullname": ["東京都", "新宿区",
↪"西新宿", "二丁目", "8 番"]}, "dist": 7.286211365075872}, {"candidate": {"id": 50372224,
 "name": "10 番", "x": 139.689697265625, "y": 35.687679290771484, "level": 7, "priority":␣
↪3, "note": "", "fullname": ["東京都", "新宿区", "西新宿", "二丁目", "10 番"]}, "dist": 279.
↪78246727626146}, {"candidate": {"id": 50372715, "name": "15 番", "x": 139.
↪68817138671875, "y": 35.68926239013672, "level": 7, "priority": 3, "note": "",
 "fullname": ["東京都", "新宿区", "西新宿", "四丁目", "15 番"]}, "dist": 321.58463778054926}]
```

注釈: リバースジオコーディング用のインデックスは、初めてリバースジオコーディングを実行した時に自動的に作成されます。この処理には辞書データーベースのサイズによっては非常に長い時間がかかる（1 時間以上）ので、辞書データベースのインストール後に `jageocoder reverse 135 34` のように実行して構築しておくことをお勧めします。

インデックスを削除したい場合は、辞書データベースのディレクトリにある `rtree.dat` `rtree.idx` という 2 つのファイルを削除してください。

3.3.3 住所辞書ディレクトリの取得

実行中の Python 環境で、住所辞書データベースがインストールされているディレクトリを取得します。

辞書データベースは `{sys.prefix}/jageocoder/db2/` の下に作成されますが、ユーザが書き込み権限を持っていない場合には`{site.USER_DATA}/jageocoder/db2/` に作成されます。

上記以外の任意の場所を指定したい場合、環境変数 `JAGEOCODER_DB2_DIR` でディレクトリを指定することができます。

コマンド

`get-db-dir`

パラメータ

（なし）

オプション

`-d` デバッグ情報を表示します。

実行例

```
(.venv) $ jageocoder get-db-dir
/home/sagara/.local/share/virtualenvs/jageocoder-kWBL7Ve6/jageocoder/db2/
```

3.3.4 住所辞書ファイルのダウンロード

住所データベースファイルをウェブからダウンロードします。v2.0 より URL は省略不可になりました。

住所データベースファイルのリストからダウンロードするファイルを選択し、その URL を指定してください。

このコマンドは `curl` や `wget` コマンドなどが利用できない場合を想定して用意しているものなので、これらのコマンドやブラウザでダウンロードしても構いません。

コマンド

download-dictionary

パラメータ

<url> ダウンロードする URL を指定します（省略不可）。

オプション

-d デバッグ情報を表示します。

実行例

```
# 街区レベルまでの全国住所辞書ファイルをダウンロードします
(.venv) $ jageocoder download-dictionary https://www.info-proto.com/static/jageocoder/
↪latest/v2/gaiku_all_v21.zip
```

3.3.5 住所辞書ファイルのインストール

住所データベースファイルを展開し、住所データベースを作ります。

コマンド

install-dictionary

パラメータ

<path> インストールする住所データベースファイルのパスを指定します（省略不可）。

オプション

-d デバッグ情報を表示します。

--db-dir 住所データベースを作るディレクトリを指定します。

実行例

```
# ダウンロード済みの住所データベースファイルをインストールします
(.venv) $ jageocoder install-dictionary gaiku_all_v21.zip
```

3.3.6 住所データベースのアンインストール

住所データベースをアンインストール（削除）します。

コマンド

uninstall-dictionary

パラメータ

（なし）

オプション

-d デバッグ情報を表示します。

--db-dir=<dir> 住所データベースのディレクトリを指定します。

実行例

```
# 住所データベースをアンインストールします
(.venv) $ jageocoder uninstall-dictionary
INFO:jageocoder.module:248:Removing directory ...
INFO:jageocoder.module:251:Dictionary has been uninstalled.
```

3.3.7 住所辞書ファイルのマイグレーション

この機能は v2.0 で廃止になりました。

3.4 コードサンプル

ここでは jageocoder モジュールを Python コード内から呼びだして利用する方法を、サンプルを用いて説明します。

3.4.1 住所から経緯度を調べる

住所文字列を指定し、その住所の経緯度を調べる処理を実装します。

より厳密には、指定された住所文字列にもっとも長く一致するレコードを住所辞書データベースから検索し、そのレコードに格納されている経緯度の値を返します。

```
>>> import jageocoder
>>> jageocoder.init()
>>> results = jageocoder.searchNode('新宿区西新宿2-8-1')
>>> if len(results) > 0:
...     print(results[0].node.x, results[0].node.y)
...
139.6917724609375 35.68962860107422
```

jageocoder.searchNode() は、指定した住所文字列に最長一致すると解釈された *Result* クラスのオブジェクトリストを返します。

```
>>> type(results[0])
<class 'jageocoder.result.Result'>
```

このクラスのオブジェクトは、一致した文字列を *matched* 属性に、住所ノードを *node* 属性に持っています。

```
>>> results[0].matched
'新宿区西新宿2-8-'
>>> results[0].node
{'id': 50357162, 'name': '8番', 'x': 139.6917724609375, 'y': 35.68962860107422, 'level':␣
↪7, 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目', '8番
↪']}
```

住所ノードは *AddressNode* クラスのオブジェクトなので、x 属性に経度、 y 属性に緯度、level 属性に住所レベルを持ちます。

```
>>> results[0].node.x
139.6917724609375
>>> results[0].node.y
35.68962860107422
>>> results[0].node.level
7
```

住所レベルの数値の意味は *jageocoder.address.AddressLevel* の定義を参照してください。

3.4.2 住所検索条件を変更する

jageocoder.set_search_config() を利用すると、住所検索の条件を変更することができます。

たとえば「中央区中央 1」を検索すると、次のように「千葉県千葉市」と「神奈川県相模原市」にある「中央区中央一丁目」の住所が見つかります。

```
>>> import jageocoder
>>> jageocoder.init()
>>> results = jageocoder.searchNode('中央区中央1')
>>> [x.node.get_fullname(" ") for x in results]
['千葉県 千葉市 中央区 中央 一丁目', '神奈川県 相模原市 中央区 中央 一丁目']
```

もし対象の住所が神奈川県にあることがあらかじめ分かっている場合には、target_area で検索範囲を神奈川県に指定しておくことで千葉市の候補を除外できます。

```
>>> jageocoder.set_search_config(target_area=['神奈川県'])
>>> results = jageocoder.searchNode('中央区中央1')
>>> [x.node.get_fullname(" ") for x in results]
['神奈川県 相模原市 中央区 中央 一丁目']
```

設定した target_area を初期値に戻したい場合は [] または None をセットしてください。また、設定条件を確認するには *jageocoder.get_search_config()* を呼んでください。

```
>>> jageocoder.set_search_config(target_area=[])
>>> jageocoder.get_search_config()
{
   'debug': False,
   'aza_skip': None,
   'best_only': True,
   'target_area': [],
   'require_coordinates': True,
   'auto_redirect': True
}
```

3.4.3 経緯度から住所を調べる

地点の経緯度を指定し、その地点の住所を調べることができます（いわゆるリバースジオコーディング）。

jageocoder.reverse() に調べたい地点の経度と緯度を渡すと、指定した地点を囲むドロネー三角形を構成する住所ノードを検索し、住所ノードを candidate、指定した地点からの距離を dist に持つ dict の list を返します。

```
>>> import jageocoder
>>> jageocoder.init()
>>> triangle = jageocoder.reverse(139.6917, 35.6896)
>>> if len(triangle) > 0:
...     print(triangle[0]['candidate']['fullname'])
...
['東京都', '新宿区', '西新宿', '二丁目']
```

jageocoder.reverse() に level オプションパラメータを指定すると、検索する住所のレベルを指定できます。デフォルトでは字レベル (6) なので、街区・地番レベルで検索したい場合は 7 を、号・枝番レベルまで検索したい場合は 8 を指定してください。

```
>>> triangle = jageocoder.reverse(139.6917, 35.6896, level=7)
>>> if len(triangle) > 0:
...     print(triangle[0]['candidate']['fullname'])
...
['東京都', '新宿区', '西新宿', '二丁目', '8 番']
```

注釈: リバースジオコーディング用のインデックスは、初めてリバースジオコーディングを実行した時に自動的に

作成されます。この処理には辞書データーベースのサイズによっては非常に長い時間がかかる（1 時間以上）ので、辞書データベースのインストール後に `jageocoder reverse 135 34` のように実行して構築しておくことをお勧めします。

インデックスを削除したい場合は、辞書データベースのディレクトリにある `rtree.dat` `rtree.idx` という 2 つのファイルを削除してください。

3.4.4 住所の属性情報を調べる

AddressNode クラスのオブジェクトには、経緯度以外にもさまざまな属性やクラスメソッドがあります。

まず以下のコードで「新宿区西新宿 2-8-1」に対応する住所要素の AddressNode オブジェクトを node 変数に代入しておきます。

```
>>> import jageocoder
>>> jageocoder.init()
>>> results = jageocoder.searchNode('新宿区西新宿 2-8-1')
>>> node = results[0].node
```

GeoJSON 表現

as_geojson() メソッドを利用すると GeoJSON 表現を取得できます。このメソッドが返すのは dict 形式のオブジェクトです。GeoJSON 文字列を取得するには、 `json.dumps()` でエンコードしてください。

```
>>> import json
>>> print(json.dumps(node.as_geojson(), indent=4, ensure_ascii=False))
{
   "type": "Feature",
   "geometry": {
      "type": "Point",
      "coordinates": [
            139.6917724609375,
            35.68962860107422
      ]
   },
   "properties": {
      "id": 50357162,
      "name": "8番",
      "level": 7,
      "priority": 3,
```

(次のページに続く)

(前のページからの続き)

```
      "note": "",
      "fullname": [
            "東京都",
            "新宿区",
            "西新宿",
            "二丁目",
            "8 番"
      ]
   }
}
```

都道府県コード

get_pref_jiscode() メソッドを利用すると JISX0401 で規定されている都道府県コード（2 桁）を取得できます。同様に、 *get_pref_local_authority_code()* メソッドでこの都道府県の団体コード（6 桁）を取得できます。

```
>>> node.get_pref_jiscode()
'13'
>>> node.get_pref_local_authority_code()
'130001'
```

市区町村コード

get_city_jiscode() メソッドを利用すると JISX0402 で規定されている市区町村コード（5 桁）を取得できます。同様に、 *get_city_local_authority_code()* メソッドでこの市区町村の団体コード（6 桁）を取得できます。

```
>>> node.get_city_jiscode()
'13104'
>>> node.get_city_local_authority_code()
'131041'
```

アドレス・ベース・レジストリ

get_aza_code() メソッドで、この住所に対応するアドレス・ベース・レジストリの町字コードを取得できます。*get_aza_names()* メソッドで町字レベルの名称（漢字表記、カナ表記、英字表記）を取得できます。

```
>>> node.get_aza_code()
'131040023002'
>>> node.get_aza_names()
[[1, '東京都', 'トウキョウト', 'Tokyo', '13'], [3, '新宿区', 'シンジュクク', 'Shinjuku-ku',
↪'13104'], [5, '西新宿', 'ニシシンジュク', '', '131040023'], [6, '二 丁目', '２チョウメ',
```

(次のページに続く)

(前のページからの続き)

```
↪'2chome', '131040023002']]
```

get_aza_names() は v1.3 から list オブジェクトを返すように変更されました。

郵便番号

get_postcode() メソッドで郵便番号を取得できます。ただしビルや事業者の郵便番号は登録されていません。

```
>>> node.get_postcode()
'1600023'
```

地図 URL のリンク

get_gsimap_link() メソッドで地理院地図へのリンク URL を、*get_googlemap_link()* メソッドで Google 地図へのリンク URL を生成します。

これらのリンクは座標から生成しています。

```
>>> node.get_gsimap_link()
'https://maps.gsi.go.jp/#16/35.689629/139.691772/'
>>> node.get_googlemap_link()
'https://maps.google.com/maps?q=35.689629,139.691772&z=16'
```

親ノードを辿る

「親ノード」とは、住所の一つ上の階層を表すノードのことです。AddressNode の属性 *parent* で取得できます。

今 node は '8 番' を指しているので、親ノードは ' 二丁目' になります。

```
>>> parent = node.parent
>>> parent.get_fullname()
[' 東京都', ' 新宿区', ' 西新宿', ' 二丁目']
>>> parent.x, parent.y
(139.6917724609375, 35.689449310302734)
```

子ノードを辿る

「子ノード」とは、住所の一つ下の階層を表すノードのことです。AddressNode の属性 *children* で取得します。

親ノードは一つですが、子ノードは複数あります。今 parent は ' 二丁目' を指しているので、子ノードはそこに含まれる街区レベル（〇番）を持つノードのリストになります。

```
>>> parent.children
[{'id': 50357153, 'name': '1 番', 'x': 139.6939239501953, 'y': 35.6916618347168, 'level':␣
```

(次のページに続く)

(前のページからの続き)

```
↪7, 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目', '1番
↪']}, {'id': 50357154, 'name': '10番', 'x': 139.689697265625, 'y': 35.687679290771484,
 'level': 7, 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目
↪', '10番']}, {'id': 50357155, 'name': '11番', 'x': 139.6876983642578, 'y': 35.
↪691104888916016, 'level': 7, 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区',
 '西新宿', '二丁目', '11番']}, {'id': 50357156, 'name': '2番', 'x': 139.6943359375, 'y':␣
↪35.68998718261719, 'level': 7, 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区
', '西新宿', '二丁目', '2番']}, {'id': 50357157, 'name': '3番', 'x': 139.6947784423828,
↪'y': 35.68826675415039, 'level': 7, 'priority': 3, 'note': '', 'fullname': ['東京都', '
新宿区', '西新宿', '二丁目', '3番']}, {'id': 50357158, 'name': '4番', 'x': 139.
↪69332885742188, 'y': 35.688148498535156, 'level': 7, 'priority': 3, 'note': '',
 'fullname': ['東京都', '新宿区', '西新宿', '二丁目', '4番']}, {'id': 50357159, 'name': '5
番', 'x': 139.69297790527344, 'y': 35.68976593017578, 'level': 7, 'priority': 3, 'note': '
↪', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目', '5番']}, {'id': 50357160, 'name
↪': '6番', 'x': 139.6924591064453, 'y': 35.6920166015625, 'level': 7, 'priority': 3,
↪'note': '', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目', '6番']}, {'id':␣
↪50357161, 'name': '7番', 'x': 139.69137573242188, 'y': 35.691253662109375, 'level': 7,
 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目', '7番']},
↪{'id': 50357162, 'name': '8番', 'x': 139.6917724609375, 'y': 35.68962860107422, 'level
↪': 7, 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目', '8
番']}, {'id': 50357163, 'name': '9番', 'x': 139.692138671875, 'y': 35.688079833984375,
 'level': 7, 'priority': 3, 'note': '', 'fullname': ['東京都', '新宿区', '西新宿', '二丁目
↪', '9番']}]
>>> [child.name for child in parent.children]
['1番', '10番', '11番', '2番', '3番', '4番', '5番', '6番', '7番', '8番', '9番']
```

AddressNode のメソッドのより詳しい説明は API リファレンスの *AddressNode* **クラス** を参照してください。

3.5 API リファレンス

jageocoder を Python コード内で利用するには、ほとんどの場合 **コードサンプル** に紹介したモジュールメソッドの *searchNode()*, *reverse()* および *jageocoder.node.AddressNode* クラスのメソッドで十分です。

ここではより詳細な API のリファレンスを提供します。

3.5.1 モジュールメソッド

jageocoder モジュールのメソッドは、以下の 3 つのグループに分類できます。

住所辞書データベースのインストールや削除などの管理を行なう

jageocoder.download_dictionary(), *jageocoder.install_dictionary()*, *jageocoder.uninstall_dictionary()*, jageocoder.migrate_dictionary(), *jageocoder.create_trie_index()*, jageocoder.dictionary_version()

モジュールの初期化や状態確認などの管理を行なう

jageocoder.init(), *jageocoder.free()*, *jageocoder.is_initialized()*, *jageocoder.get_db_dir()*, *jageocoder.get_module_tree()*, jageocoder.version()

検索機能を提供する

jageocoder.set_search_config(), *jageocoder.get_search_config()*, *jageocoder.search()*, *jageocoder.searchNode()*, *jageocoder.reverse()*

A Python module for Japanese-address geocoding.

注釈: Before using this module, install address-dictionary from the Web as follows:

$ python -m jageocoder install-dictionary

サンプル

You can get the latitude and longitude from a Japanese address by running the following steps.

```
>>> import jageocoder
>>> jageocoder.init()
>>> jageocoder.searchNode('<Japanese-address>')
```

jageocoder.**create_trie_index**() → None

Create the TRIE index from the database file.

This function is a shortcut for AddressTree.create_trie_index().

jageocoder.**download_dictionary**(*url: str*) → None

Download address-dictionary from the specified url into the current directory.

パラメータ

url (*str*) -- The URL where the zipped address-dictionary file is available.

jageocoder.**free**()

Frees all objects created by 'init()'.

jageocoder.**get_db_dir**(*mode: str = 'r'*) → Path | None

Get the database directory.

パラメータ

mode (*str, optional(default='r')*) -- Specifies the mode for searching the database directory. If 'a' or 'w' is set, search a writable directory. If 'r' is set, search a database file that already exists.

戻り値

The path to the database directory. If no suitable directory is found, raise an AddressTreeException.

戻り値の型

Path or None

メモ

This method searches a directory in the following order of priority. - 'JAGEOCODER_DB_DIR' environment variable - '(sys.prefix)/jageocoder/db2/' - '(site.USER_BASE)/jageocoder/db2/'

jageocoder.**get_module_tree**() → *AddressTree* | None

Get the module-level AddressTree singleton object.

戻り値

The singleton object.

戻り値の型

AddressTree

jageocoder.**get_search_config**(*keys: str | List[str] | None = None*) → dict

Get current configurable search parameters.

パラメータ

keys (*str, List[str], optional*) -- If a name of parameter is specified, return its value. Otherwise, a dict of specified key and its value pairs will be returned.

戻り値の型

Any, or dict.

jageocoder.**init**(*db_dir: PathLike | None = None*, *mode: str | None = 'r'*, *debug: bool | None = False*, ***kwargs*) → None

Initialize the module-level AddressTree object *jageocoder.tree* ready for use.

パラメータ

- **db_dir** (*os.PathLike, optional*) -- The database directory. 'address.db' and 'address.trie' are stored in this directory.
- **mode** (*str, optional(default='r')*) -- Specifies the mode for opening the database.
  - In the case of 'a', if the database already exists, it will be used. If it does not exist, create a new one.
  - In the case of 'w', if the database already exists, delete it first. Then create a new one.
  - In the case of 'r', if the database already exists, it will be used. Otherwise raise a JageocoderError exception.
- **debug** (*bool, Optional(default=False)*) -- Debugging flag.

jageocoder.**install_dictionary**(*path: PathLike*, *db_dir: PathLike | None = None*, *skip_confirmation: bool = False*) → None

Install address-dictionary from the specified path.

パラメータ

- **path** (*os.PathLike*) -- The file path where the zipped address-dictionary file exists.
- **db_dir** (*os.PathLike, optional*) -- The directory directory where the database files will be installed.

  If omitted, use *get_db_dir()* to decide the directory.

jageocoder.**installed_dictionary_readme**(*db_dir: PathLike | None = None*) → str

Get the content of README.txt attached to the installed dictionary.

パラメータ

db_dir (*os.PathLike, optional*) -- The directory where the database files has been installed. If omitted, it will be determined by *get_db_dir()*.

戻り値

The content of the text.

戻り値の型

str

jageocoder.**installed_dictionary_version**(*db_dir: PathLike | None = None*) → str

Get the installed dictionary version.

パラメータ

db_dir (*os.PathLike, optional*) -- The directory where the database files has been installed. If omitted, it will be determined by *get_db_dir()*.

戻り値

The version string of the installed dicitionary.

戻り値の型

str

jageocoder.**is_initialized**() → bool

Checks if the module has been initialized with *init()*.

戻り値

True if the module is initialized, otherwise False.

戻り値の型

bool

jageocoder.**reverse**(*x: float*, *y: float*, *level: int | None = None*, *as_dict: bool | None = True*) → list

Reverse geocoding.

パラメータ

- **x** (*float*) -- Longitude of the point.
- **y** (*float*) -- Latitude of the point.
- **level** (*int, optional*) -- Target node level.
- **as_dict** (*bool, default=True*) -- If True, returns candidates as dict objects.

戻り値の型

list

メモ

- The result list contains up to 3 nodes.

Each element is a dict type with the following structure:

{"candidate":AddressNode, "dist":float}

jageocoder.**search**(*query: str*) → dict

Search node from the tree by the query.

パラメータ

query (*str*) -- An address notation to be searched.

戻り値

- *A dict containing the following elements.*

- **matched** (*str*) -- The matching substring.
- **candidates** (*list of dict*) -- List of dict representation of nodes with the longest match to the query string.

jageocoder.**searchNode**(*query: str*) → List[*Result*]

Searches for address nodes corresponding to an address notation and returns the matching substring and a list of nodes.

パラメータ

query (*str*) -- An address notation to be searched.

戻り値

A list of AddressNode and matched substring pairs.

戻り値の型

list

注釈: The *search_by_trie* function returns the standardized string as the match string. In contrast, the *searchNode* function returns the de-starndized string.

サンプル

```
>>> import jageocoder
>>> jageocoder.init()
>>> jageocoder.searchNode('多摩市落合1-15-2')
[[[11460207:東京都(139.69178,35.68963)1(lasdec:130001/jisx0401:13)]>[12063502:多摩市
(139.446366,35.636959)3(jisx0402:13224)]>[12065383:落合(139.427097,35.
↪624877)5(None)]>[12065384:一丁目(139.427097,35.624877)6(None)]>[12065390:15番地
(139.428969,35.625779)7(None)], '多摩市落合1-15-']]
```

jageocoder.**set_search_config**(***kwargs*)

Set configurable search parameters.

注釈: The possible keywords and their meanings are as follows.

best_only: bool (default = True)

If set to False, returns all search result candidates whose prefix matches.

aza_skip: bool, None (default = False)

Specifies how to skip aza-names while searching nodes. - If None, make the decision automatically -

If False, do not skip - If True, always skip

require_coordinates: bool (default = True) If set to False, nodes without coordinates are also included in the search.

target_area: List[str] (Default = [])
Specify the areas to be searched. The area can be specified by the list of name of the node (such as prefecture name or city name), or JIS code.

auto_redirect: bool (default = True)
When this option is set and the retrieved node has a new address recorded in the "ref" attribute, the new address is retrieved automatically.

`jageocoder.uninstall_dictionary`(*db_dir: PathLike | None = None*) → None

Uninstall address-dictionary.

パラメータ

db_dir (*os.PathLike, optional*) -- The directory where the database files has been installed. If omitted, it will be determined by *get_db_dir()*.

3.5.2 AddressTree クラス

住所階層木構造を表すクラスです。

jageocoder では、住所は表形式ではなく、id=0 を持つ根 (root) ノードの下に都道府県を表すノードがあり、そのさらに下に市区町村を表すノードがあり、という階層木構造を利用して管理しています。

それぞれのノードは *jageocoder.node.AddressNode* クラスのオブジェクトです。

また、このクラスは利用する住所データベースの情報も管理しています。言い換えれば、複数の AddressTree オブジェクトを生成し、それぞれ別の住所データベースを開けば、同時に複数の異なる住所データベースを利用することもできます。

class `jageocoder.tree.`**AddressTree**(*db_dir: PathLike | None = None, mode: str = 'a', debug: bool | None = None*)

The address-tree structure.

db_path

Path to the sqlite3 database file.

Type
str

dsn

RFC-1738 based database-url, so called "data source name".

Type

str

trie_path

Path to the TRIE index file.

Type

str

engine

The database engine which is used to connect to the database.

Type

sqlalchemy.engine.Engine

conn

The connection object which is used to communicate witht the database.

Type

sqlalchemy.engine.Connection

session

The session object used for a series of database operations.

Type

sqlalchemy.orm.Session

root

The root node of the tree.

Type

AddressNode

trie

The TRIE index of the tree.

Type

AddressTrie

mode

The mode in which this tree was opened.

Type
str

config

Settings the search method in this tree.

Type
dict

__init__(*db_dir: PathLike | None = None*, *mode: str = 'a'*, *debug: bool | None = None*)

The initializer

パラメータ

- **db_dir** (*os.PathLike, optional*) -- The database directory. If omitted, the directory returned by get_db_dir() is used. 'address.db' and 'address.trie' are stored under this directory.
- **mode** (*str, optional (default='a')*) -- Specifies the mode for opening the database.
  - In the case of 'a', if the database already exists, use it. Otherwize create a new one.
  - In the case of 'w', if the database already exists, delete it first. Then create a new one.
  - In the case of 'r', if the database already exists, use it. Otherwise raise a JageocoderError exception.
- **debug** (*bool, optional (default=False)*) -- Debugging flag. If set to True, write debugging messages. If omitted, refer 'JAGEOCODER_DEBUG' environment variable, or False if the environment variable is also undefined.

add_address(*address_names: List[str]*, *do_update: bool = False*, *cache: LRU | None = None*, ***kwargs*) → *AddressNode*

Create a new AddressNode and add to the tree.

パラメータ

- **address_names** (*list of str*) -- A list of the address element names. For example, ["東京都","新宿区","西新宿", "２丁目"]
- **do_update** (*bool*) -- When an address with the same name already exists, update it with the value of kwargs if 'do_update' is true, otherwise do nothing.
- **cache** (*LRU, optional*) -- A dict object to use as a cache for improving performance, whose keys are the address notation from the prefecture level and whose values are the corresponding nodes. If not specified or None is given, do not use the cache.
- ****kwargs** (*properties of the new address node.*) -- x : float. X coordinate or longitude in decimal degree y : float. Y coordinate or latitude in decimal degree level: int. Level of

the node note : str. Note

戻り値

The added node.

戻り値の型

AddressNode

check_line_format(*args: List[str]*) → int

Receives split args from a line of comma-separated text representing a single address element, and returns the format ID.

パラメータ

args (*list[str]*)

戻り値

The id of the identified format. 1. Address names without level, lon, lat 2. Address names without level, lon, lat, note 3. Address names without level, lon, lat, level without note 4. Address names without level, lon, lat, level, note

戻り値の型

int

サンプル

```
>>> from jageocoder_converter import BaseConverter
>>> base = BaseConverter()
>>> base.check_line_format(['1;北海道','3;札幌市','4;中央区','141.34103','43.
↪05513'])
1
>>> base.check_line_format(['1;北海道','3;札幌市','4;中央区','5;大通','6;西二十丁
目','141.326249','43.057218','01101/ODN-20/'])
2
>>> base.check_line_format(['北海道','札幌市','中央区','大通','西二十丁目','141.
↪326249','43.057218',6])
3
>>> base.check_line_format(['北海道','札幌市','中央区','大通','西二十丁目','141.
↪326249','43.057218',6,'01101/ODN-20/'])
4
```

close() → NoReturn

create_note_index_table() → None

Collect notes from all address elements and create search table with index.

create_trie_index() → None

Create the TRIE index from the tree.

get_address_node(*id: int*) → *AddressNode*

Get address node from the tree by its id.

パラメータ

id (*int*) -- The node id.

戻り値

Node with the specified ID.

戻り値の型

AddressNode

get_cache_info() → dict

get_config(*keys: str | List[str] | None = None*)

Get configurable parameter(s).

パラメータ

keys (*str, List[str], optional*) -- If a name of parameter is specified, return its value. Otherwise, a dict of specified key and its value pairs will be returned.

戻り値の型

Any, or dict.

サンプル

```
>>> import jageocoder
>>> jageocoder.init()
>>> jageocoder.get_module_tree().get_config('aza_skip')
'off'
>>> jageocoder.get_module_tree().get_config(['best_only', 'target_area'])
{'best_only': True, 'target_area': []}
>>> jageocoder.get_module_tree().get_config()
{'debug': False, 'aza_skip': 'off', 'best_only': True, 'target_area': [],
↪'require_coordinates': False}
```

get_node_by_id(*node_id: int*) → *AddressNode*

Get the full node information by its id.

パラメータ

node_id (*int*) -- The target node id.

戻り値の型

AddressNode

get_node_fullname(*node:* AddressNode | *int*) → List[str]

get_root() → *AddressNode*

Get the root-node of the tree. If not set yet, create and get the node from the database.

戻り値

The root node object.

戻り値の型

AddressNode

get_trie_nodes() → TrieNode

Get the TRIE node table.

メモ

- Todo: If the trie index is not created, create.

get_version() → str

Get the version of the tree file.

戻り値

The version string.

戻り値の型

str

is_version_compatible() → bool

Check if the dictionary version is compatible with the package.

戻り値

True if compatible, otherwize False.

戻り値の型

bool

parse_line_args(*args: List[str]*, *format_id: int*) → list

Receives split args from a line of comma-separated text representing a single address element, and returns a list of parsed attributes.

パラメータ

- **args** (*list[str]*) -- List of split args in a line
- **format_id** (*int*) -- The id of the line format identfied by *check_line_format*

戻り値 A list containing the following attributes. - Address names: list[str] - Longitude: float - Latitude: float - Level: int or None - note: str or None

戻り値の型

list

サンプル

```
>>> from jageocoder_converter import BaseConverter
>>> base = BaseConverter()
>>> base.parse_line_args(['1;北海道','3;札幌市','4;中央区','141.34103','43.05513
↪'], 1)
[['1;北海道','3;札幌市','4;中央区'], 141.34103, 43.05513, None, None]
>>> base.parse_line_args(['1;北海道','3;札幌市','4;中央区','5;大通','6;西二十丁目
↪','141.326249','43.057218','01101/ODN-20/'], 2)
[['1;北海道','3;札幌市','4;中央区','5;大通','6;西二十丁目'],141.326249,43.057218,
↪None,'01101/ODN-20/']
>>> base.parse_line_args([' 北海道',' 札幌市',' 中央区',' 大通',' 西二十丁目','141.
↪326249','43.057218',6,'01101/ODN-20/'], 4)
[[' 北海道',' 札幌市',' 中央区',' 大通',' 西二十丁目'],141.326249,43.057218,6,'01101/
↪ODN-20/']
```

read_file(*path: PathLike*, *do_update: bool = False*) → None

Add AddressNodes from a text file. See 'data/test.txt' for the format of the text file.

パラメータ

- **path** (*os.PathLike*) -- Text file path.
- **do_update** (*bool (default=False)*) -- When an address with the same name already exists, update it with the value of the new data if 'do_update' is true, otherwise do nothing.

read_stream(*fp: TextIO*, *do_update: bool = False*) → None

Add AddressNodes to the tree from a stream.

パラメータ

- **fp** (*io.TextIO*) -- Input text stream.

- **do_update** (*bool (default=False)*) -- When an address with the same name already exists, update it with the value of the new data if 'do_update' is true, otherwise do nothing.

search(*query: str*, ***kwargs*) → list

searchNode(*query: str*) → List[*Result*]

Searches for address nodes corresponding to an address notation and returns the matching substring and a list of nodes.

パラメータ

query (*str*) -- An address notation to be searched.

戻り値

A list of AddressNode and matched substring pairs.

戻り値の型

list

注釈: The *search_by_trie* function returns the standardized string as the match string. In contrast, the *searchNode* function returns the de-starndized string.

サンプル

```
>>> import jageocoder
>>> jageocoder.init()
>>> tree = jageocoder.get_module_tree()
>>> tree.searchNode(' 多摩市落合 1-15-2')
[[[11460207: 東京都 (139.69178,35.68963)1(lasdec:130001/jisx0401:13)]>[12063502: 多
摩市 (139.446366,35.636959)3(jisx0402:13224)]>[12065383: 落合 (139.427097,35.
↪624877)5(None)]>[12065384: 一丁目 (139.427097,35.624877)6(None)]>[12065390:15 番
地 (139.428969,35.625779)7(None)], ' 多摩市落合 1-15-']]
```

search_by_tree(*address_names: List[str]*) → *AddressNode*

Get the corresponding node id from the list of address element names, recursively search for child nodes using the tree.

For example, [' 東京都',' 新宿区',' 西新宿',' 二丁目'] will search the ' 東京都' node under the root node, search the ' 新宿区' node from the children of the ' 東京都' node. Repeat this process and return the ' 二丁目' node which is a child of ' 西新宿' node.

パラメータ

address_names (*list of str*) -- A list of address element names to be searched.

戻り値

The node matched last.

戻り値の型

AddressNode

search_by_trie(*query: str*, *processed_nodes: Set[int] | None = None*) → dict

Get the list of corresponding nodes using the TRIE index. Returns a list of address element nodes that match the query string in the longest part from the beginning.

For example, ' 中央区中央 1 丁目' will return the nodes corresponding to ' 千葉県千葉市中央区中央一丁目 ' and ' 神奈川県相模原市中央区中央一丁目'.

パラメータ

- **query** (*str*) -- An address notation to be searched.
- **processed_nodes** (*Set of the AddressNode's id, optional*) -- List of node's id that have already been processed.

戻り値

- *A dict object whose key is a node id*
- *and whose value is a list of node and substrings*
- *that match the query.*

search_nodes_by_codes(*category: str*, *value: str*) → List[*AddressNode*]

Search nodes by category and value.

パラメータ

- **category** (*str*) -- Category name such as 'jisx0402' or 'postcode'.
- **value** (*str*) -- Target value.
- **levels** (*List[int], optional*) -- The address levels of target nodes.

戻り値の型

List[*AddressNode*]

set_config(***kwargs*)

Set configuration parameters.

注釈: The possible keywords and their meanings are as follows.

best_only: bool (default = True) If set to False, returns all search result candidates whose prefix matches.

aza_skip: bool, None (default = False)
Specifies how to skip aza-names while searching nodes. - If None, make the decision automatically - If False, do not skip - If True, always skip

require_coordinates: bool (default = True) If set to False, nodes without coordinates are also included in the search.

target_area: List[str] (Default = [])
Specify the areas to be searched. The area can be specified by the list of name of the node (such as prefecture name or city name), or JIS code.

auto_redirect: bool (default = True)
When this option is set and the retrieved node has a new address recorded in the "ref" attribute, the new address is retrieved automatically.

---

update_name_index() → int

Update *name_index* field using the standardizing logic of the current version.

注釈: This method also updates the version information of the dictionary.

戻り値

Number of records updated.

戻り値の型

int

validate_config(*key: str*, *value: Any*) → None

Validate configuration key and parameters.

パラメータ

- **key** (*str*) -- The name of the parameter.
- **value** (*str*, *int*, *bool*, *None*) -- The value to be set to the parameter.

メモ

If the key-value pair is not valid, raise RuntimeError.

3.5.3 AddressNode クラス

住所要素ノードを表すクラスです。

ノードは名称（「東京都」や「新宿区」など）や経緯度などの属性情報と、親ノードや子ノード集合へのリンクを持ちます。

class jageocoder.node.**AddressNode**(*id: int | None = None*, *name: str | None = None*, *name_index: str | None = None*, *x: float | None = None*, *y: float | None = None*, *level: int | None = None*, *priority: int | None = None*, *note: str | None = None*, *parent_id: int | None = None*, *sibling_id: int | None = None*)

The address node stored in 'address_node' table.

パラメータ

- **id** (*int*) -- The key identifier that is automatically sequentially numbered.
- **name** (*str*) -- The name of the address element, such as ' 東京都' or ' 新宿区'
- **x** (*float*) -- X-coordinate value. (Longitude)
- **y** (*float*) -- Y-coordinate value. (Latitude)
- **level** (*int*) -- The level of the address element. The meaning of each value is as follows.
- **priority** (*int*) -- Priority assigned to each source of data. Smaller value indicates higher priority.
- **note** (*string*) -- Note or comment.
- **parent_id** (*int*) -- The id of the parent node.
- **sibling_id** (*int*) -- The id of the next sibling node.

name_index

The standardized string for indexing created from its name.

Type
: str

__init__(*id: int | None = None*, *name: str | None = None*, *name_index: str | None = None*, *x: float | None = None*, *y: float | None = None*, *level: int | None = None*, *priority: int | None = None*, *note: str | None = None*, *parent_id: int | None = None*, *sibling_id: int | None = None*) → None

The initializer of the node.

In addition to the initialization of the record, the name_index is also created.

__repr__()

Return repr(self).

add_dummy_coordinates() → *AddressNode*

Add dummy coordinate values to the node. If the coordinates of the node are not specified, this method selects a child node with valid coordinates and temporarily duplicates it.

メモ

- The duplicated values are not registered in the database.

add_note(*key: str*, *value: str*) → None

Add an attribute with key-value set to the note field.

パラメータ

- **key** (*str*) -- The key of the attribute to be added.
- **value** (*str*) -- The value of the attribute to be added.

as_dict()

Return the dict notation of the node.

as_geojson()

Return the geojson notation of the node.

property children: List[*AddressNode*]

Get child nodes of the node.

戻り値

The list of the child nodes.

戻り値の型

List[*AddressNode*]

property dataset

Get dataset record.

classmethod from_record(*record*) → *AddressNode*

Convert from a record of AddressNodeTable to an AddressNode object.

パラメータ

record (*CapnpRecord*) -- A record stored in PortableTab's table.

戻り値の型

AddressNode

get_aza_code() → str

Returns the 'AZA-code' concatinated with the city-code and the aza-id containing this node.

get_aza_id() → str

Returns the AZA-id defined by JDA address-base-registry containing this node.

get_aza_names(*tree:* AddressTree | *None* = *None*) → list

Returns representation of Aza node containing this node.

パラメータ

tree (AddressTree, *optional*) -- The tree containing this node.

戻り値

A list containing notations from the prefecture level to the Aza level in the following format:

[AddressLevel, Kanji, Kana, English, code]

戻り値の型

list

get_aza_record(*tree:* AddressTree | *None* = *None*) → object | None

Returns ABR's aza record corresponding to this node..

パラメータ

tree (AddressTree, *optional*) -- The tree containing this node.

戻り値

A record object if exists. Otherwise None.

戻り値の型

Record of AzaMaster

get_child(*target_name: str*) → *AddressNode*

Get a child node with the specified name.

パラメータ

target_name (*str*) -- The name (or standardized name) of the target node.

戻り値

If a node with the specified name is found, it is returned; Otherwise return None.

戻り値の型

AddressNode

get_children() → List[*AddressNode*]

Get child nodes of the node.

戻り値

The list of the child nodes.

戻り値の型

List[*AddressNode*]

get_city_jiscode() → str

Returns the jisx0402 code of the city that contains this node.

get_city_local_authority_code() → str

Returns the 地方公共団体コード of the city that contains this node.

get_city_name() → str

Returns the name of city that contains this node.

get_fullname(*delimiter: str | None = None*, *alt: str | None = ''*) → str | List[str]

Returns a complete address notation starting with the name of the prefecture.

パラメータ

- **delimiter** (*str, optional*) -- Specifies the delimiter character for the address element; If None is specified, returns a list of elements.
- **alt** (*str, optional*) -- String to be used instead if the node name is empty.

戻り値

If a delimiter is specified (including ""), returns a string consisting of the name of the address node concatenated with the delimiter. Otherwise, it returns a list of address node names.

戻り値の型

str, List[str]

get_googlemap_link() → str

Returns the URL for GSI Map with parameters. ex. https://maps.google.com/maps?q=24.197611,120.780512&z=18

get_gsimap_link() → str

Returns the URL for GSI Map with parameters. ex. https://maps.gsi.go.jp/#13/35.713556/139.750385/

get_name(*alt: str | None = ''*)

Name of node.

パラメータ

alt (*str, optional*) -- String to be used instead if the node name is empty.

get_nodes_by_level()

The method returns an array of this node and its upper nodes. The Nth node of the array contains the node corresponding to address level N. If there is no element corresponding to level N, None is stored.

サンプル

```
>>> import jageocoder
>>> jageocoder.init()
>>> node = jageocoder.searchNode('多摩市落合 1-15')[0][0]
>>> [str(x) for x in node.get_nodes_by_level()]
['None', "[{'id': 49229505, 'name': '東京都', 'x': 139.6917724609375, 'y': 35.
↪68962860107422, 'level': 1, 'priority': 1, 'note': 'lasdec:130001/jisx0401:13',
 'fullname': ['東京都']}]", 'None', "[{'id': 50026506, 'name': '多摩市', 'x':␣
↪139.4463653564453, 'y': 35.636959075927734, 'level': 3, 'priority': 1, 'note':
↪'geoshape_city_id:13224A1971/jisx0402:13224/postcode:2060000', 'fullname': ['
東京都', '多摩市']}]", 'None', "[{'id': 50042949, 'name': '落合', 'x': 139.
↪42709350585938, 'y': 35.6248779296875, 'level': 5, 'priority': 2, 'note': '',
↪'fullname': ['東京都', '多摩市', '落合']}]", "[{'id': 50042950, 'name': '一丁目
↪', 'x': 139.42709350585938, 'y': 35.6248779296875, 'level': 6, 'priority': 2,
↪'note': 'aza_id:0010001/postcode:2060033', 'fullname': ['東京都', '多摩市', '落
合', '一丁目']}]", "[{'id': 50042979, 'name': '15番地', 'x': 139.42897033691406,
 'y': 35.62577819824219, 'level': 7, 'priority': 3, 'note': '', 'fullname': ['東
京都', '多摩市', '落合', '一丁目', '15番地']}]"]
```

get_notes() → Tuple[Tuple[str, str]]

Get list of key-value set from the note field.

戻り値

Tuple containing tuples consisting of keys and values.

戻り値の型

Tuple[Tuple[str, str]]

get_omissible_children(*tree:* AddressTree) → List[*AddressNode*]

Create a list of omissible child nodes refer to Aza-master.

パラメータ

tree (AddressTree) -- The tree containing this node.

戻り値

List of omissible child nodes.

戻り値の型

List[*AddressNode*]

get_omissible_index(*tree:* AddressTree, *index: str*, *processed_nodes: List[*AddressNode*]*, *strict: bool* = *False*) → str

Obtains an optional leading substring from the search string index.

パラメータ

- **tree** (*AddressTree*) -- The tree containing this node.
- **index** (*str*) -- Target string.
- **processed_nodes** (*List of AddressNode*) -- List of nodes that have already been processed by TRIE search results.
- **strict** (*bool*) -- If true, check the omission more strict.

戻り値

The optional leading substring. If not omissible, an empty string is returned.

戻り値の型

str

メモ

- Retrieve the lower address elements of this node that have start_count_type is 1 from the aza_master.
- If the name of the element is contained in the index, the substring before the name is returned.

get_parent() → *AddressNode* | None

Get the parent node.

戻り値

The parent object.

戻り値の型

AddressNode

メモ

- Returns None if the current node is directly under the root node.

get_parent_list() → List[*AddressNode*]

Returns a complete node list starting with the prefecture.

get_postcode() → str

Returns the 7digit postcode of the oaza that contains this node.

get_pref_jiscode() → str

Returns the jisx0401 code of the prefecture that contains this node.

get_pref_local_authority_code() → str

Returns the 地方公共団体コード of the prefecture that contains this node.

get_pref_name() → str

Returns the name of prefecture that contains this node.

has_valid_coordinate_values() → bool

Check if it has valid coordinate values.

戻り値

True if x and y are both valid coordinate values.

戻り値の型

bool

is_inside(*area: str*) → int

Check if the node is inside the area specified by parent's names or jiscodes.

パラメータ

area (*str*) -- Specify the area by name or jiscode.

戻り値

It returns 1 if the node is inside the region, 0 if it is not inside, and -1 if it cannot be determined by this node.

戻り値の型

int

メモ

If a city code is specified and the node is at the prefecture level, it will return 0 if the first two digits of the code do not match, otherwise it will return -1.

iter_children() → Iterator[*AddressNode*]

Iterate children of the node.

戻り値

An iterator object that returns a child node in sequence.

戻り値の型

Iterator[*AddressNode*]

property parent: *AddressNode* | None

Get the parent node of the node.

戻り値

The parent object. When the parent node is the root-node, None will be returned.

戻り値の型

AddressNode

retrieve_upper_node(*target_levels: List[int]*)

Retrieves the node at the specified level from the this node or one of its upper nodes.

classmethod root() → *AddressNode*

Generate the root AddressNode object.

search_child_with_criteria(*pattern: str*, *min_candidate: str | None = None*, *gt_candidate: str | None = None*, *max_level: int | None = None*) → List[*AddressNode*]

Search for children nodes that satisfy the specified conditions.

パラメータ

- **pattern** (*str*) -- The regular expression that the child node's name must match.
- **min_candidate** (*str, optional*) -- The smallest string that satisfies the condition as the name of a child node.
- **gt_candidate** (*str, optional*) -- The smallest string that exceeds the upper limit that satisfies the condition as the name of a child node.
- **max_level** (*int, optional*) -- Maximum level of child nodes; unlimited if None.

戻り値

A list of all child nodes that satisfy the specified condition.

戻り値の型

List[*AddressNode*]

search_recursive(*tree:* AddressTree, *index: str*, *processed_nodes: Set[int] | None = None*) → List[*Result*]

Search nodes recursively that match the specified address notation.

パラメータ

- **tree** (*AddressTree*) -- The tree containing this node.
- **index** (*str*) -- The standardized address notation.
- **processed_nodes** (*Set of the AddressNode's id, optional*) -- List of node's id that have already been processed by TRIE search results.

戻り値

List of Result objects containing relevant AddressNode.

戻り値の型

List[*Result*]

set_attributes(***kwargs*)

Set attributes of the node by kwargs values.

パラメータ

kwargs (*dict*) -- A dictionary with the name of the attribute to be changed as key and the new value as value.

注釈: The 'name' attribute can't be modified.

---

set_notes(*notes: Tuple[Tuple[str, str]]*) → None

Set the note field from the list of key-value set.

パラメータ

notes (*((str, str))*) -- List of key-value set (as tuples).

to_record()

Convert from the AddressNode object to an object of dict that can be registered to the AddressNodeTable table.

戻り値の型

dict

3.5.4 AddressLevel クラス

住所要素ノードが持つ住所レベルを定義するクラスです。

class jageocoder.address.**AddressLevel**

Address Levels

1 = 都道府県 2 = 郡・支庁・振興局 3 = 市町村および特別区 4 = 政令市の区 5 = 大字 6 = 字 7 = 地番または住居表示実施地域の街区 8 = 枝番または住居表示実施地域の住居番号

classmethod guess(*name*, *parent*, *trigger*)

Guess the level of the address element.

パラメータ

- **name** (*str*) -- The name of the address element
- **parent** (AddressNode) -- The parent node of the target.
- **trigger** (*dict*) -- properties of the new address node who triggered adding the address element.

  name : str. name. ("２丁目") x : float. X coordinate or longitude. (139.69175) y : float. Y coordinate or latitude. (35.689472) level : int. Address level (1: pref, 3: city, 5: oaza, ...) note : str. Note.

3.5.5 Result クラス

ジオコーディング結果を格納するためのクラスです。

node 属性に *jageocoder.node.AddressNode* クラスの住所要素ノードオブジェクトが、*matched* 属性に一致した部分文字列が格納されます。

class jageocoder.result.**Result**(*node:* AddressNode | *None* = *None*, *matched: str* = '', *nchars: int* = *0*)

Representing the result of searchNode().

node

The node matched the query.

Type

AddressNode

matched

The matched substring of the query.

Type

str

nchars

The number of matched characters. It is used only for recursive search.

Type
: int

as_dict() → dict

Convert Result object to dict type for display.

戻り値

A dict object containing the following elements;

"node"
: AddressNode object converted to dict type.

"matched"
: The substring matching the query.

戻り値の型
: dict

as_geojson() → dict

Convert Result to GeoJSON dict type for display.

戻り値

A GeoJSON dict object containing the following elements;

"type"
: Always "Feature".

"geometry"
: A point type geometry containing latitude and longitude.

"properties"
: Include in "matched" the substring that matched the query, in addition to the attributes of the node.

戻り値の型
: dict

get_matched_nchars() → int

Get the the number of matched characters.

戻り値
: Number of characters in the matched substring.

戻り値の型

int

`get_matched_string`() → str

Get the matched string part of the result.

戻り値

The matched substring.

戻り値の型

str

`get_node`() → *AddressNode*

Get the node part of the result.

戻り値

The matched node.

戻り値の型

AddressNode

`set`(*node:* AddressNode, *matched:* *str*, *nchars:* *int* = *0*) → *Result*

Set node and matched string.

Python モジュール索引

索引