(2022-12-07 初版)

nihuBridge の API について

API(Application Programming Interfece) とは，狭義には「各種システム／サービスがそのシステム／サービスを利用するアプリケーションに対して公開するインタフェース(Wikipedia)」を指します．nihuBridge の API には，統合検索を利用・管理するためのものと，蓄積データの利用・管理のためのものがあります．また，一般ユーザが利用できるものと，管理者ユーザのみが利用できるものとに分かれます．

ここでは，一般ユーザが利用できる nihuBridge API のうち，統合検索機能の利用に関するものについて，実例を挙げて紹介したいと思います．

この資料は Python を用いた事例集となっています．この資料自体は， Google Colaboratoryを用いて作成されたJupyter Notebook ファイルになっています．

基本編

応用編

付録

API「メタデータ検索(HIT数)」について

レコードの検索のための API として，ヒット数のみを返す API「メタデータ検索(HIT数)」と，レコード情報まで返す API「メタデータ検索」とがあります．

検索条件に該当するレコードがあるかないかだけ知りたいだけなら，極力 API「メタデータ検索(HIT数)」を使うようにしてください．

一番簡単な検索のサンプルとして，「堀浩一」を含むレコードの件数を調べてみましょう．検索条件を Python のデータ値（辞書型の値）として書き（6～16行目），API を呼び出してみます．検索条件の書き方については付録A：検索条件の書き方をご覧ください．

20行目の「response = request.post(cmd, json=query)」によって，POST メソッドによって API が呼び出されます．検索条件は JSON 形式の文字列に変換されて送信されます(json=queryの部分）．API からの返信は変数 response に格納されます．

21行目の「result = response.json()」によって，API からの返信が JSON 形式の文字列であると解釈され，Python のデータ値（辞書型の値）に変換されて，変数 result に格納されます．

24行目の「print(json.dumps(result, indent=2))」によって，resultの内容が JSON形式の文字列に変換され，かつ人間が見やすい形に整えられて出力されます．

[query_hit.py]

```
import json
import codecs
import requests

# 検索条件
query = {
    'query': {
        'conditions': [
            {
              'query': {
                  'term': "堀浩一"
              }
            }
        ]
    }
}
```

```
17
18 # API「メタデータ検索(HIT数)」を呼び出す
19 cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search-hits"
20 response = requests.post(cmd, json=query)
21 result = response.json()
22
23 # 検索結果を画面に出力する (pretty printing)
24 print(json.dumps(result, indent=2))
```

```
{
  "info": {
    "statusCode": 0,
    "total": 3
  }
}
```

３件あると表示されました.

API「メタデータ検索」について

それでは，この３件のデータを具体的に求めてみましょう．API「メタデータ検索(HIT数)」のかわりに API「メタデータ検索」を呼び出してみます.

検索条件の書き方（6～23行目）が少し追加になっていることに注意してください．取得するレコード数の上限値（最大何件分取得するか）を必ず指定する必要があります．検索条件の書き方については，付録A：検索条件の書き方をご覧ください

ここでは，取得するレコードのフィールドを指定してみることにします．「タイトル(title)」と「作成者(creator)」の２つのフィールドの値を取得してみましょう．取得できるフィールドについては，付録D：メタデータについてをご参照ください.

31行目の「print(json.dumps(result, indent=2, ensure_ascii=False))」のうち，「ensure_ascii=False」は，resultの内容に漢字（マルチバイト文字）を含む場合，エスケープ表記(\u30bfのような表記）に変換されてしまうのを防ぐものです.

[query.py]

```
1 import json
2 import codecs
3 import requests
4
5 # 検索条件
6 query = {
7     'query': {
8         'conditions': [
9             {
10                'query': {
11                    'term': '堀浩一'
12                }
13            }
14        ],
15        'fields': [
16            "title",
17            "creator"
18        ],
19        'paging': {
20            'size': 1000
21        }
22    }
23 }
24
25 # API「メタデータ検索」を呼び出す
26 cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search"
27 response = requests.post(cmd, json=query)
28 result = response.json()
29
30 # 検索結果を画面に出力する (pretty printing)
31 print(json.dumps(result, indent=2, ensure_ascii=False))
```

```
{
  "info": {
    "statusCode": 0,
    "total": 3,
    "databases": {
      "nijl_kokubungakuronbun": 2,
      "ninjal_toshokanmokuroku": 1
    }
  },
  "hits": [
    {
      "database": "ninjal_toshokanmokuroku",
      "id": "13256078",
```

```
      "fields": [
        {
          "field": "title",
          "label": "タイトル",
          "value": [
            "専門用語の自動抽出/シナリオを用いて、構造化されたキーワードをアブストラクトから抽出する一手法/MDSによる日本語入力と編集処理に
          ]
        },
        {
          "field": "creator",
          "label": "作成者",
          "value": [
            "田中康仁著/堀浩一[ほか]著/坂本義行著/大河内正明[ほか]著/中野洋著/吉村賢治[ほか]著/長尾真[ほか]著/長尾真[ほか]著"
          ],
          "highlight": [
            "田中康仁著/<em class=¥"highlight_keyword¥">堀浩一</em>[ほか]著/坂本義行著/大河内正明[ほか]著/中野洋著/吉村賢治[ほか]著/長尾真
          ]
        }
      ]
    },
    {
      "database": "nijl_kokubungakuronbun",
      "id": "14322727",
      "fields": [
        {
          "field": "title",
          "label": "タイトル",
          "value": [
            "[論文題名]「かつはあやしき」",
            "[掲載誌名]平安文学研究"
          ]
        },
        {
          "field": "creator",
          "label": "作成者",
          "value": [
            "[論文執筆者名]堀浩一郎"
          ],
          "highlight": [
            "[論文執筆者名]<em class=¥"highlight_keyword¥">堀浩一</em>郎"
```


3件分のレコードを取得することが出来ました．いずれのレコードも「作成者(creator)」のフィールドに検索文字列「堀浩一」が含まれています．（ただし，２件目のレコードは「堀浩一郎」さんであって「堀浩一」さんではありません．）

検索結果として得られるフィールドの情報は，フィールド名(field)，表示名(label)，フィールド値(value)のほか，検索文字列が含まれるフィールドについては，強調表示(highlight)の情報が追加され，ヒット部分が HTML の em タグで囲まれた形で出力されます．

```
...
{
  "field": "creator"
  "label": "作成者",
  "value": [
    "[論文執筆者名]堀浩一"
  ],
  "highlight": [
    "[論文執筆者名]<em class=¥"highlight_keyword¥">堀浩一</em>"
  ]
}
...
```

API「メタデータ取得」について

nihuBridge に登録されてる統合検索データの各レコード（メタデータ）にはそれぞれ，「研究資源ID」という識別子が割り当てられており，レコードを一意に識別することができます．

研究資源IDを用いてレコードを指定して，メタデータを取り出すには，API「メタデータ取得」を用います．

さきほどの検索で得られた3件のレコードのうち，１件目のメタデータを取り出してみましょう．研究資源IDは13256078とありました．


```
...
{
  "info": { ... },
  "hits": [
    {
      "database": "ninjal_toshokanmokuroku",
```

```
      "id": "13256078",
      "fields": [ ... ]
    }, ...
  ]
}
```

以下のリンクをクリックすると，Webブラウザが開いて，研究資源IDが13256078のレコードのメタデータが表示されます．

https://api.bridge.nihu.jp/v1/integratedsearch/metadata/13256078/

同じことを，pythonを用いて書いてみましょう．10行目のrequest.get関数によって，GETメソッドによってAPIが呼び出されます．

[get.py]

```
1 import json
2 import codecs
3 import requests
4
5 # 研究資源ID
6 id = "13256078"
7
8 # API「メタデータ取得」を呼び出す
9 cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadata/"+id+"/"
10 response = requests.get(cmd)
11 result = response.json()
12
13 # 取得結果を画面に表示する
14 print(json.dumps(result, indent=2, ensure_ascii=False))
```


```
{
  "info": {
    "statusCode": 0,
    "total": 1
  },
  "researchResource": {
    "databaseId": "ninjal_toshokanmokuroku",
    "researchResourceId": "13256078",
    "doi": null,
    "originalId": [
      "ninjal_toshokanmokuroku_1001227147",
      " [資料ID] 1001227147",
      " [請求記号] 801.019/Ke27/25"
    ],
    "status": "1",
    "title": [
      "専門用語の自動抽出/シナリオを用いて、構造化されたキーワードをアブストラクトから抽出する一手法/MDSによる日本語入力と編集処理について
    ],
    "alternativeTitle": null,
    "creator": [
      "田中康仁著/堀浩一[ほか]著/坂本義行著/大河内正明[ほか]著/中野洋著/吉村賢治[ほか]著/長尾真[ほか]著/長尾真[ほか]著"
    ],
    "contributor": null,
    "publisher": [
      " [情報処理学会 : 計算言語学研究会] "
    ],
    "subject": [
      "計量言語学"
    ],
    "keyword": null,
    "description": null,
    "dateCreated": [
      "[1981.2]"
    ],
    "datePublished": "2022-04-13",
    "dateModified": "2022-04-13",
    "futureUpdates": "1",
    "license": null,
    "type": null,
    "archiveFormat": null,
    "contentSize": null,
    "encodingFormat": null,
    "inLanguage": [
      "日本語"
    ],
    "isBasedOn": null,
    "link": [
      {
        "type": "",
        "link": "http://libgw.ninjal.ac.jp/mylimedio/search/book.do?target=local&bibid=53834"
      }
    ],
    "sampleLink": null,
    "metadataFields": null,
    "metadataLanguage": null,
    "relatedLink": [
```

API「メタデータ取得」で取得できるフィールドの詳細は，付録D：メタデータについてをご参照ください．

事例１：正規表現による文字列検索

API「メタデータ検索(HIT数)」「メタデータ検索」では，文字列のより精密な検索方法として，完全一致(is)，前方一致(start)，後方一致(end)，部分一致(contain)，正規表現(regex)の５種類を指定することができます．この場合，検索するフィールドを指定する必要があります．

ここでは，正規表現による検索をご紹介します．（文字列の）正規表現(Wikipedia)は，複雑な検索条件を簡潔に指示できる場合があります．うまく使えば，より精密で漏れの少ない検索を行うことが可能です．

以下の例は，タイトル(title)フィールドから「三十三間堂」を部分一致で検索してみた例です．135件ヒットします．

[regex_1.py]

```
import json
import codecs
import requests

# 検索条件
query = {
    'query': {
        'conditions': [
            {
                'query': {
                    'field': "title",
                    'term': "三十三間堂",
                    'match': "contain"
                }
            }
        ]
    }
}

# API「メタデータ検索(HIT数)」を呼び出す
cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search-hits"
response = requests.post(cmd, json=query)
result = response.json()

# 検索結果を画面に出力する (pretty printing)
print(json.dumps(result, indent=2))
```

```
{
  "info": {
    "statusCode": 0,
    "total": 135
  }
}
```

昔の文献では，「三十」を表す漢字として「卅」や「丗」が用いられる場合があります．検索文字列を「(三十|卅|丗)三間堂」として正規表現で検索してみましょう．これで，「三十三間堂」「卅三間堂」「丗三間堂」の３つの文字列を一度に探してくれることになります．

[regex_2.py]

```
import json
import codecs
import requests

# 検索条件
query = {
    'query': {
        'conditions': [
            {
                'query': {
                    'field': "title",
                    'term': "(三十|卅|丗)三間堂",
                    'match': "regex"
                }
            }
        ]
    }
}

# API「メタデータ検索(HIT数)」を呼び出す
cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search-hits"
response = requests.post(cmd, json=query)
result = response.json()
```

```
24
25 # 検索結果を画面に出力する（pretty printing）
26 print(json.dumps(result, indent=2, ensure_ascii=False))
```


```
{
  "info": {
    "statusCode": 0,
    "total": 166
  }
}
```


166件ヒットしました．31件分の検索漏れをすくいあげることができました．

正規表現は，いくつかの文字に特別の意味を持たせて，複数種類の文字列を１つの「パターン」として書き表すことで，検索したい文字列を指定する方法と言えます．詳しくは付録B：正規表現の書き方をご覧ください．

正規表現以外の文字列検索（前方一致，後方一致，部分一致，完全一致）では，異体字同定（異体字を同一視した検索)を自動的に行います．詳しくは付録C：異体字同定についてをご覧ください．

事例２：時間検索

nihuBridge APIをもちいた時間検索の例を紹介します．ここでは，文政年間(文政元～13，1818～1831年)に製作された錦絵の情報を，国立歴史民俗博物館の館蔵錦絵データベース(nmjh_kanzounisikie)および国際日本文化研究センターの米国議会図書館所蔵浮世絵データベース(ircjs_ukiyoe)から検索してみましょう．

和暦で指定した年代から，時間検索に必要なグレゴリオ暦による日付範囲の情報を得るために，Hutimeが提供するHutime Web APIを利用します．

はじめに「文政年間」から，始まりの日と終わりの日を求めます(17～39行目)．これは和暦の書式で返されるので，つぎにそれぞれをグレゴリオ暦に変換します(41～65行目)．都合２回 Hutime Web API を呼び出します．

時間情報は対象時間範囲(temporal)のフィールドに入っていますので，そのフィールドを指定して，時間の範囲を検索文字列として与え，範囲検索(BETWEEN)を指定して，検索を実行します(69～102行目)．

それぞれのデータベースごとのヒット件数を求めたいので，合計のヒット数のみ表示される「メタデータ検索(HIT数)」ではなく，「メタデータ検索」のAPIを用います(101行目)．ヒットしたレコードのデータはいまは不要なので，データ取得の最大件数を1にして最小に留めるようにしています(93～95行目)．

[temporal.py]

```
 1 import json
 2 import codecs
 3 import requests
 4
 5 # 和暦検索
 6 # 和暦を指定
 7 wadate_keyword = "文政"
 8 wadate_type = "年間"
 9
10 wadate_type_str = {
11     '日': 'day',
12     '月': 'month',
13     '年': "year",
14     '年間': "era"
15 }
16
17 # 区間の最初の日と最後の日を求める
18 date_query_1 = {
19     "jsonrpc": "2.0",
20     "method": "info",
21     "params": {
22         "ical": "1001.1", # 和暦（南朝）
23         "itype": wadate_type_str[wadate_type],
24         "ival": wadate_keyword,
25         "oprop": "begin;end"
26     },
27     "id": "2022120901"
28 }
29
30 # Hutime Web API を呼び出す
31 cmd = "http://ap.hutime.org/cal/"
32 response = requests.post(cmd, json=date_query_1)
33 date_result_1 = response.json()
34 #print(json.dumps(date_result_1, indent=2, ensure_ascii=False))
35
36 # 開始日と終了日（和暦表記）
37 wadate_start = date_result_1['result'][0]['begin']
```

```
wadate_end = date_result_1['result'][0]['end']
#print(wadate_start, wadate_end)

# グレゴリオ暦に変換する：ISO 8601の規定に従う
date_query_2 = {
    "jsonrpc": "2.0",
    "method": "conv",
    "params": {
        "ical": "1001.1", # 和暦（南朝）
        "itype": "date",
        "ival": wadate_start + f'¥r¥n' + wadate_end,
        "ocal": "2.1", # グレゴリオ暦
        "otype": "date",
        "oprop": "text",
        "oform": "yyyy-MM-dd"
    },
    "id": "2022120902"
}
#print(date_query_2['params']['ival'])

# Hutime Web API を呼び出す
response = requests.post(cmd, json=date_query_2)
date_result_2 = response.json()
#print(json.dumps(date_result_2, indent=2, ensure_ascii=False))

# 開始日と終了日（グレゴリオ暦表記）
date_start = date_result_2['result'][0]['text']
date_end = date_result_2['result'][1]['text']

print(f'{wadate_keyword}[{wadate_type}]は{date_start}({wadate_start})から{date_end}({wadate_end})まで')

# ISO 8601形式による時間表記で「開始時間, 終了時間」を与える
date_range = f'{date_start}T00:00:00+09:00,{date_end}T23:59:59+09:00'

# データベースの検索条件
query = {
    'database': [
        "nmjh_kanzounisikie",
        "ircjs_ukiyoe"
    ],
    'query': {
        'conditions': [
            {
                'query': {
                    'field': "temporal",
                    'term': date_range,
                    'operator': "BETWEEN"
                }
            }
        ],
        'fields': [
            "title",
            "creator",
            "temporal"
        ],
        'paging': {
            'size': 1
        }
    }
}

# API「メタデータ検索(HIT数)」を呼び出す
cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search"
response = requests.post(cmd, json=query)
result = response.json()

# 検索結果を画面に出力する（pretty printing）
print(json.dumps(result, indent=2, ensure_ascii=False))
```

```
文政[年間]は1818-05-26(文政1年4月22日)から1831-01-22(文政13年12月9日)まで
{
  "info": {
    "statusCode": 0,
    "total": 391,
    "databases": {
      "ircjs_ukiyoe": 323,
      "nmjh_kanzounisikie": 68
    }
  },
  "hits": [
    {
      "database": "ircjs_ukiyoe",
      "id": "10097993",
      "fields": [
```

```
            {
              "field": "title",
              "label": "タイトル",
              "value": [
                "[題名（日本語）/題名（ローマ字）]娘道成寺/musume doujyouji"
              ]
            },
            {
              "field": "creator",
              "label": "作成者",
              "value": [
                "[絵師名（日本語）/絵師名（ローマ字）]歌川国貞/Utagawa Kunisada"
              ]
            },
            {
              "field": "temporal",
              "label": "対象時間範囲 表示",
              "value": [
                {
                  "description": [
                    "1830-1844"
                  ],
                  "date": "1830-01-01T00:00:00+09:00,1844-12-31T00:00:00+09:00"
                }
              ]
            }
          ]
        }
      ]
    }
```

事例３：空間検索

最後に，nihuBridge APIによる空間検索の例を紹介します．国文学研究資料館の史料所在情報・検索データベース(nijl_siryousyozaijyouhou_kensaku)から，都道府県（ここでは「京都府」）を指定して(9行目)，「都道府県の境界を囲む長方形」の範囲で検索してみます(79～192行目)．

指定した都道府県の範囲を含む地図を表示(194～213行目)し，検索結果の位置情報を地図上にマークしてみました(215～219行目)．検索の結果得られる位置情報は複数のレコードで重複しているので，重複チェックの処理を行っています(169～187行目)．

地図の表示には folium を利用しています． 地図画像としては国土交通省国土地理院が配信する地理院地図（地理院タイル）を用いています． また，「都道府県の境界を囲む長方形」の情報は，国土交通省が公開する国土数値情報 行政区域 第1.0版を参考に作成しました．

[spatial.py]

```python
import sys
import json
import codecs
import requests
import folium
import re

# 検索する都道府県
pref = "京都府"
print(f'{pref}を囲む長方形範囲を検索')

# 都道府県コード(JIS X 0401より)
PrefId = {
    '北海道': 1, '青森県': 2, '岩手県': 3, '宮城県': 4, '秋田県': 5,
    '山形県': 6, '福島県': 7, '茨城県': 8, '栃木県': 9, '群馬県': 10,
    '埼玉県': 11, '千葉県': 12, '東京都': 13, '神奈川県': 14, '新潟県': 15,
    '富山県': 16, '石川県': 17, '福井県': 18, '山梨県': 19, '長野県': 20,
    '岐阜県': 21, '静岡県': 22, '愛知県': 23, '三重県': 24, '滋賀県': 25,
    '京都府': 26, '大阪府': 27, '兵庫県': 28, '奈良県': 29, '和歌山県': 30,
    '鳥取県': 31, '島根県': 32, '岡山県': 33, '広島県': 34, '山口県': 35,
    '徳島県': 36, '香川県': 37, '愛媛県': 38, '高知県': 39, '福岡県': 40,
    '佐賀県': 41, '長崎県': 42, '熊本県': 43, '大分県': 44, '宮崎県': 45,
    '鹿児島県': 46, '沖縄県': 47
}

# 都道府県境界を囲む長方形の緯度経度座標
# 国土交通省　国土数値情報　行政区域第1.0版を参考に作成
# https://nlftp.mlit.go.jp/ksj/jpgis/datalist/KsjTmplt-N03.html
PrefRect_LatLon = [ [(0.0,0.0),(0.0,0.0)], # 0:dummy
    [(41.333333,139.25),(45.6,149.05)], #  1:北海道
    [(40.166667,139.375),(41.583333,141.75)], #  2:青森県
    [(38.666667,140.625),(40.5,142.125)], #  3:岩手県
    [(37.75,140.25),(39.083333,141.75)], #  4:宮城県
    [(38.833333,139.625),(40.583333,141.0)], #  5:秋田県
    [(37.666667,139.5),(39.25,140.75)], #  6:山形県
    [(36.75,139.125),(38.0,141.125)], #  7:福島県
```

```
    [(35.666667,139.625),(37.0,140.875)], #  8:茨城県
    [(36.166667,139.25),(37.166667,140.375)], #  9:栃木県
    [(35.916667,138.375),(37.083333,139.75)], # 10:群馬県
    [(35.75,138.625),(36.333333,140.0)], # 11:埼玉県
    [(34.833333,139.625),(36.166667,141.0)], # 12:千葉県
    [(20.416667,136.0),(35.916667,154.0)], # 13:東京都
    [(35.083333,138.875),(35.75,139.875)], # 14:神奈川県
    [(36.666667,137.625),(38.583333,140.0)], # 15:新潟県
    [(36.25,136.75),(37.0,137.875)], # 16:富山県
    [(36.0,136.125),(37.916667,137.375)], # 17:石川県
    [(35.333333,135.375),(36.333333,136.875)], # 18:福井県
    [(35.166667,138.125),(36.0,139.25)], # 19:山梨県
    [(35.166667,137.25),(37.083333,138.75)], # 20:長野県
    [(35.083333,136.25),(36.5,137.75)], # 21:岐阜県
    [(34.5,137.375),(35.666667,139.25)], # 22:静岡県
    [(34.5,136.625),(35.5,137.875)], # 23:愛知県
    [(33.666667,135.75),(35.333333,137.0)], # 24:三重県
    [(34.75,135.75),(35.75,136.5)], # 25:滋賀県
    [(34.666667,134.75),(35.833333,136.125)], # 26:京都府
    [(34.25,135.0),(35.083333,135.75)], # 27:大阪府
    [(34.083333,134.25),(35.75,135.5)], # 28:兵庫県
    [(33.833333,135.5),(34.833333,136.25)], # 29:奈良県
    [(33.416667,134.875),(34.416667,136.125)], # 30:和歌山県
    [(35.0,133.125),(35.666667,134.625)], # 31:鳥取県
    [(34.25,131.625),(36.416667,133.5)], # 32:島根県
    [(34.25,133.25),(35.416667,134.5)], # 33:岡山県
    [(34.0,132.0),(35.166667,133.5)], # 34:広島県
    [(33.666667,130.75),(34.833333,132.5)], # 35:山口県
    [(33.5,133.625),(34.333333,134.875)], # 36:徳島県
    [(34.0,133.375),(34.583333,134.5)], # 37:香川県
    [(32.833333,132.0),(34.333333,133.75)], # 38:愛媛県
    [(32.666667,132.375),(33.916667,134.375)], # 39:高知県
    [(32.916667,130.0),(34.25,131.25)], # 40:福岡県
    [(32.916667,129.625),(33.666667,130.625)], # 41:佐賀県
    [(31.916667,128.0),(34.75,130.5)], # 42:長崎県
    [(32.083333,129.875),(33.25,131.375)], # 43:熊本県
    [(32.666667,130.75),(33.75,132.25)], # 44:大分県
    [(31.333333,130.625),(32.916667,132.0)], # 45:宮崎県
    [(27.0,128.375),(32.333333,131.25)], # 46:鹿児島県
    [(24.0,122.875),(27.916667,131.375)] # 47:沖縄県
]

# 検索する緯度経度範囲
range_latlon = PrefRect_LatLon[PrefId[pref]]
query_range_latlon = f'({range_latlon[0][0]},{range_latlon[0][1]}),({range_latlon[1][0]},{range_latlon[1][1]})'
print(f'範囲: {query_range_latlon}')

# 検索条件
query = {
    'database': [ "nijl_siryousyozaijyouhou_kensaku" ],
    'query': {
        'conditions': [
            {
                'query': {
                    'field': "spatial",
                    'term': query_range_latlon
                }
            }
        ]
    }
}

# API「メタデータ検索(HIT数)」を呼び出す
cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search-hits"
response = requests.post(cmd, json=query)
result = response.json()

# 検索結果を画面に出力する (pretty printing)
#print(json.dumps(result, indent=2, ensure_ascii=False))

# ヒット件数
n_hits = result['info']['total']
print(f'{n_hits}件ヒットしました')

# 「メタデータ検索」のレコード取得数の上限値
max_size = 1000

# 文字列sから緯度経度座標を読み取りリストを返す関数
def scanLatLonStr(s):
    s1 = re.sub(r'[()]', "", s) # かっこをとる
    s2 = s1.split(',') # コンマを区切りにして分割 (文字列のリスト)
    # 浮動小数点数のリストに変換
    return list(map(float, s2))
```

```
    return tuple(map(float, s))

# 集合 -> コンマ区切り文字列
def SetToStr(s):
    t = ""
    for u in list(s):
        if len(t) > 0:
            t = t + ", "
        t = t + u
    return t

# レコードから取得する情報を格納する変数
# 座標 -> ヒット数
p_hits = {}
# 座標 -> 地名表記集合（現地名）
p_names = {}
# 座標 -> 地名表記集合（旧地名）
p_oldnames = {}

# ヒット数が１以上ならレコードを取得する
if n_hits > 0:

    # 取得するレコード位置の先頭
    start = 0

    # 検索条件の追加：取得するフィールド
    query['query']['fields'] = [ "spatial" ]

    # レコード群を max_size 件づつ取得する
    while start < n_hits:
        # 検索条件の追加：取得するレコード範囲
        remains = n_hits - start # 残りレコード数
        query['query']['paging'] = {
            'start': start,
            'size': max_size if remains > max_size else remains
        }
        #print(query)

        # API「メタデータ検索」を呼び出す
        cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search"
        response = requests.post(cmd, json=query)
        result = response.json()

        # 検索結果を画面に出力する（pretty printing）
        #print(json.dumps(result, indent=2, ensure_ascii=False))

        # 取得データを変数に読み込む
        for r in result['hits']:
            # 座標の取得
            p = scanLatLonStr(r['fields'][0]['value'][0]['place'][0])
            k = (p[0], p[1])
            # 件数のカウント
            if not k in p_hits:
                p_hits[k] = 0
            p_hits[k] = p_hits[k] + 1
            # 地名情報の取得
            if not k in p_names:
                p_names[k] = set()
                p_oldnames[k] = set()
            d = r['fields'][0]['value'][0]['description']
            # 地名は要素数2の配列[現地名, 旧地名]
            p_names[k].add(d[0])
            p_oldnames[k].add(d[1])
            # 次のレコード範囲へ
            start = start + max_size

    # データの表示
    print(f'マークの数: {len(p_hits)}')
    for k, v in p_hits.items():
        print(f'{k}: {v:4}: {SetToStr(p_oldnames[k])}({SetToStr(p_names[k])})')

# 都道府県の地図を表示
print('***地図の表示***')

# 地図の表示中心を求める
# range_latlon = PrefRect_LatLon[PrefId[pref]]
center_lat = (range_latlon[0][0] + range_latlon[1][0]) / 2
center_lon = (range_latlon[0][1] + range_latlon[1][1]) / 2
center_latlon = (center_lat,center_lon)
print(center_latlon, range_latlon)

# 地理院地図を用いて地図を作成する
prefmap = folium.Map(location=center_latlon,
```

```
                    zoom_start=6,
                    tiles="https://cyberjapandata.gsi.go.jp/xyz/std/{z}/{x}/{y}.png",
                    attr="地理院地図",
                    crs="EPSG3857",
                    width="100%", height="100%")

# 都道府県境界を囲む長方形を描画する
folium.Rectangle(bounds=range_latlon).add_to(prefmap)

# マーカーを描画する　ポップアップに情報を登録する
for k, v in p_hits.items():
    f = folium.IFrame(f'{SetToStr(p_oldnames[k])}<br/>{SetToStr(p_names[k])}<br/>{v}件')
    p = folium.Popup(f, min_width=200, max_width=200)
    folium.Marker(location=k,popup=p).add_to(prefmap)

# 地図表示の大きさを最適化する
prefmap.fit_bounds(bounds=range_latlon)

# 地図を表示する
prefmap
# prefmap.save("spatial_out.html")
```

```
京都府を囲む長方形範囲を検索
範囲: (34.666667, 134.75), (35.833333, 136.125)
1675件ヒットしました
マークの数: 50
(35.016667, 135.959722):  57: 近江国栗太郡(滋賀県栗太郡栗東町, 滋賀県草津市)
(35.582778, 135.152778):  12: 丹後国与謝郡(京都府与謝郡伊根町, 京都府与謝郡加悦町, 京都府宮津市, 京都府福知山市)
(35.405278, 134.767222):  10: 但馬国養父郡(兵庫県養父郡八鹿町, 兵庫県養父郡養父町)
(34.887222, 135.22):   4: 摂津国有馬郡(兵庫県西宮市, 兵庫県三田市)
(35.076667, 135.216667):   4: 丹波国多紀郡(兵庫県多紀郡, 兵庫県多紀郡西紀町, 兵庫県多紀郡篠山町)
(34.797222, 134.9825):   2: 播磨国美嚢郡(兵庫県三木市, 兵庫県不明)
(34.916667, 134.965833):   6: 播磨国加東郡, 播磨加東郡(兵庫県小野市, 兵庫県加東郡社町, 兵庫県加東郡東条町)
(34.9325, 134.828056):   3: 播磨国加西郡(兵庫県加西市)
(35.048056, 134.924167):   7: 播磨国多可郡(兵庫県多可郡黒田庄町, 兵庫県多可郡中町, 兵庫県多可郡加美町)
(35.659167, 134.762778):   6: 但馬国美含郡(兵庫県城崎郡竹野町, 兵庫県城崎郡香住町)
(35.4625, 134.873611):   5: 但馬国出石郡(兵庫県出石郡出石町, 兵庫県出石郡但東町)
(34.954444, 134.760833):  19: 播磨国神西郡, 播磨国神東郡(兵庫県神崎郡市川町, 兵庫県神崎郡香寺町, 兵庫県姫路市)
(34.791667, 134.796944):  24: 播磨国印南郡(兵庫県加古川市, 兵庫県高砂市, 兵庫県姫路市)
(34.766111, 134.836667):  23: 播磨国加古郡(兵庫県加古川市, 兵庫県高砂市)
(34.8525, 135.603611):  26: 摂津国島上郡(大阪府高槻市, 大阪府枚方市, 福岡県福岡市東区, 大阪府三島郡島本町)
(35.014167, 135.707222):  133: 山城国葛野, 山城国葛野郡(京都府京都市中京区, 京都府京都市右京区, 京都府京都市北区, 京都府京都市南区, 京都府京都市
(35.645833, 136.055278):  11: 越前国敦賀郡(福井県南条郡河野村, 福井県敦賀市, 福井県小浜市)
(34.696667, 135.486944):   2: 摂津国西成郡(大阪府大阪市西区, 大阪府大阪市東区)
(34.9725, 135.814167):  30: 山城国宇治郡(京都府京都市伏見区, 京都府京都市山科区)
(35.025, 135.761389):  103: 山城国京都(京都府京都市, 京都府京都市左京区, 京都府京都市上京区, 京都府京都市中京区, 京都府京都市東山区)
(34.690278, 135.183056):  38: 摂津国武庫郡(兵庫県西宮市, 不明不明, 兵庫県宝塚市)
(35.544444, 134.820556):  12: 但馬国城崎郡(兵庫県城崎郡城崎町, 兵庫県豊岡市)
(35.483611, 135.746389):  40: 若狭国遠敷郡(福井県福井市, 福井県遠敷郡上中町, 福井県遠敷郡名田庄村, 福井県小浜市)
(34.935278, 135.761389):   6: 山城国紀伊郡(京都府京都市伏見区, 京都府京都市南区)
(35.043611, 135.770556):  74: 山城国愛宕郡(京都府京都市東山区, 京都府京都市北区, 京都府京都市左京区, 京都府京都市上京区)
(35.301111, 135.118889):  72: 丹波国天田郡(京都府天田郡夜久野町, 京都府天田郡夜久里町, 京都府福知山市, 京都府天田市)
(35.2975, 135.256667):   7: 丹波国何鹿郡(京都府綾部市, 京都府福知山市)
(34.736111, 135.818333):  26: 山城国相楽郡(京都府相楽郡加茂町, 京都府相楽郡精華町)
(35.110556, 135.473611):  58: 丹波国船井郡(京都府船井郡園部町, 京都府船井郡和知町, 京都府船井郡丹波町)
(35.605833, 134.888889):   1: 丹後国熊野郡(京都府熊野郡久美町)
(34.947778, 135.700278):  29: 山城国乙訓郡(京都府京都市西京区, 京都府向日市, 京都府京都市南区)
(35.487222, 135.553611):   1: 若狭国大飯郡(福井県小浜市)
(35.1275, 136.098056):  20: 近江国蒲生郡(滋賀県近江八幡市, 滋賀県蒲生郡竜王町, 滋賀県蒲生郡蒲生町, 滋賀県蒲生郡日野町, 滋賀県蒲生郡安土町, 滋賀
(35.059167, 135.994167):  25: 近江国野洲郡(滋賀県野洲郡中主町, 滋賀県守山市, 滋賀県野洲郡野洲町)
```

付録A：検索条件の書き方

※ここで説明しているのは Python から nihuBridge API を利用するときの検索条件の書き方になります.

API「メタデータ検索(HIT数)」における検索条件の書き方は以下のようになります.

```
{
  "query": {
    "conditions": [ 《検索条件項》, ... ]
  }
}
```

"conditions"に複数の《検索条件項》を並べて与えると，絞り込み検索を指示できます．すなわち，ひとつ前の検索条件で検索した結果に対して，AND/ORで検索条件を追加することができます.

API「メタデータ検索」における検索条件の書き方は以下のようになります.

```
{
  "query": {
    "conditions": [ 《検索条件項》, ... ],
    "fields": [ "《取得フィールド名》", ... ], # 省略可
    "paging": {
      "start": 《整数値》, # 何件目から（先頭は0）：省略可
      "size":  《整数値》 # 何件分：必須
    }
  }
}
```

API「メタデータ検索」では，取得するレコード数の上限を検索条件中に"size"として指定しなければなりません．現在"size"に与えられる数の上限は1000です.

何件目から，を表す"start"は，先頭を0件目として指定します．省略すると0と解釈されます.

また，データを取得するフィールドを"fields"で指定することが可能です．《取得フィールド名》として指定できるフィールド名は，付録D：メタデータについてをご参照ください.

《検索条件項》について

《検索条件項》として書けるのは

```
{
  "connect": "AND" もしくは "OR", # ２個目の条件から必須
  "negation": True もしくは False, # 任意
  《単一条件項》 もしくは 《複合条件項》 # 必須
}
```

です．

"connect"は，"conditions"に複数の検索条件を与えた場合，２番目の《検索条件項》から必須の項目となります．"AND"もしくは"OR"を与えてAND/ORを指示します．

"negation"は，《単一条件項》 もしくは 《複合条件項》で与えた検索条件のNOTを取りたい，すなわち指定した条件に“合わない”ものを検索したいときに True を与えます．省略すると False を指定したのと同じ，すなわち条件に“合う”ものを検索します．

《単一条件項》は

```
"query": 《単一条件》
```

と与えます．《複合条件項》は

```
"queries" :[ 《検索条件項》, ... ]
```

と与えます．

《複合検索項》を用いて，《検索条件項》が入れ子のように書けることに注意してください．"queries"は，数学でいうカッコの役割を果たすもので，"conditions"と同じ書き方を許します．これにより複数の検索を１つにまとめることができ，複雑な検索条件を書くことを可能にしています．

《単一条件》について

《単一条件》として書けるのは

```
{
  "field": "《検索フィールド名》", # 省略可
  《単一検索値》 もしくは 《複数検索値》, # 必須
  "operator": "LE" もしくは "GE" もしくは "BETWEEN", # 時間検索では必須
  "match": "start" もしくは "contain" もしくは "end" もしくは "is" もしくは"regex", # 省略可
  "normalize": True もしくは False, # 省略可
  "nagation": True もしくは False # 省略可
}
```

です．

《単一条件》内では，「"field": "《検索フィールド名》"」によって，検索するフィールドを１つだけ指定することができます．指定しなければ全フィールドが検索対象になります．《検索フィールド名》として指定できるフィールド名は，付録D：メタデータについてをご参照ください．

《単一検索値》 は

```
"term": 《検索値》
```

であり，《複数検索値》は

```
"terms" :[ 《検索値》, ... ]
```

です．"terms"に複数の《検索値》を与えた場合は，OR検索の意味になります．つまり与えた《検索値》のいずれかに合致すれば，条件にヒットしたとみなされます．

"operator"は《検索値》が時間検索のとき必須の項目です．"match"は文字列検索の方法を細かく指示するものです．詳しくは次項「データ型と《検索値》の書き方」をご覧ください．

"normalize"は，異体字同定（付録C：異体字同定についてを参照）をしたくないときに False を与えます．省略すると True を指定したのと同じ，すなわち異体字同定を行います．

"negation"は，《単一検索値》 もしくは 《複合検索値》で与えた検索条件のNOTを取りたい，すなわち指定した条件に“合わない”ものを検索したいときに True を与えます．省略すると False を指定したのと同じ，すなわち条件に“合う”ものを検索します．

データ型と《検索値》の書き方

フィールドにはデータ型があり，その型に合わせて《検索値》の書き方が変わります．フィールドのデータ型は，付録D：メタデータについてをご参照ください．

- 《検索値》はすべて文字列として書きます．すなわち，「漢字」「"abc"」のように，クオーテーションマークかダブルクオーテーションマークで囲って書きます．
- 文字列の検索では，「"match": "start"」をともなうことで前方一致，「"match": "contain"」をともなうことで中間一致，「"match": "end"」をともなうことで後方一致，「"match": "is"」をともなうことで全体一致，「"match": "regex"」をともなうことで正規表現による検索を，それぞれ指示することができます．省略された場合は中間一致で検索します．
- 日時はISO 8601に従って書きます．たとえば1983年4月1日午後1時20分（日本時間）であれば「"1983-04-01T13:20:00+9:00"」 のように書きます．「"operator": "LE"」もしくは「"operator": "GE"」をともなうことで，指定した時間より前もしくは後を検索することができます．
- 日時範囲は「"始まりの日時,終わりの日時"」のように，２つの日時をコンマで区切った文字列として書きます．「"operator": "BETWEEN"」をともなうことで，指定した時間範囲に（一部分でも）含まれるかどうかを検索することができます．
- 緯度経度範囲は,「"(南端の緯度,西端の経度),(北端の緯度,東端の経度)"」のように書きます．指定した空間範囲に（一部分でも）含まれるかどうかを検索することができます．
  緯度と経度は十進法度単位の小数(*)で書き表します．すなわち，分と秒を十進小数に変換(1分=1/60，1秒=1/3600）して与えます．南緯と西経は負の数で，北緯と東経は正の数で表します．たとえば「"(34.666667,134.75),(35.833333,136.125)"」は，北緯34度40分から北緯35度50分，東経134度45分から東経136度7分30秒の範囲を表わします．
  (*)東経135度のように，分，秒が0の場合でも，「135」ではなく「135.0」のように，必ず小数点を含むように書かなければならないことに注意してください．

付録B：正規表現の書き方

nihuBridge API における正規表現の書き方は，Elasticsearch Guide – Regular expression syntaxに従います．正規表現では，いくつかの文字に特別の意味を持たせて，複数種類の文字列を１つの「パターン」」として書き表します．nihuBridge APIで使えるパターンの書き方は下表のとおりです．

パターン	意味
文字	【[...] 外で使われたとき： ., ?, +, *, {, }, |, (,), [,], \ 以外の文字】その１文字 【[...] 内で使われたとき： [,], ^, -, \ 以外の文字】その１文字
\文字	【[...] 外で使われたとき： ., ?, +, *, {, }, |, (,), [,], \】その１文字（特別扱いしない） 【[...] 内で使われたとき： [,], ^, -, \】その１文字（特別扱いしない）
.	任意の１文字
パターン?	直前のパターンを0回または1回繰り返す
パターン+	直前のパターンを1回以上繰り返す
パターン*	直前のパターンを0回以上繰り返す
パターン{整数}	直前のパターンを整数回繰り返す
パターン{整数1,整数2}	直前のパターンを整数1回から整数2回の範囲で繰り返す
パターン...|パターン...	右側の最長パターン列または左側の最長パターン列のどちらかに一致
(**パターン...**)	カッコ内のパターン列をひとつのパターンとする
[**文字...**]	カッコ内の１文字 ※先頭の文字は [,], - でも特別扱いせずその1文字として扱う ※※文字として ., ?, +, *, {, }, (,) が現れても特別扱いせずその１文字として扱う
[**^文字...**]	カッコ内の文字以外の１文字 ※先頭の文字は [,], ^, - でも特別扱いせずその１文字として扱う ※※文字として ., ?, +, *, {, }, (,) が現れても特別扱いせずその１文字として扱う
文字1-文字2	【[...] 内で使われたとき】文字1から文字2の範囲（文字コード順)の１文字
^パターン...	【^が正規表現の先頭にあるとき】パターン列が文字列の先頭から一致 【^がそれ以外の場所にあるとき】^ をその１文字とみなす（特別扱いしない）
パターン...$	【$が正規表現の末尾にあるとき】パターン列が文字列の末尾まで一致 【$がそれ以外の場所にあるとき】$をその１文字とみなす（特別扱いしない）

正規表現の書き方のサンプルを下表に示します．

サンプル	意味
ab.	aba, abb, abz, ... などに一致
abc?	ab と abc に一致
abc\?	abc? に一致
ab+	ab, abb, abbb, ... などに一致
ab\+	ab+ に一致
ab*	a, ab, abb, abbb, ... などに一致
ab*	ab* に一致
ab{2}	abb に一致
ab{2,4}	abb, abbb, abbbb に一致
ab{2,}	abb, abbb, abbbb, abbbbb, ... などに一致
ab{,4}	a, ab, abb, abbb, abbbb に一致
ab\{2\}	ab{2} に一致
abc|xyz	abc, xyz に一致

サンプル	意味
abc|xyz	abc|xyz に一致
abc(def)?	abc, abcdef に一致（abcdには一致しない）
a(ijk|uvw)z	aijkz, auvwz に一致
[abc]	a, b, c に一致
[a-c]	a, b, c に一致
[-abc]	-, a, b, c に一致
[abc\-]	a, b, c, - に一致
[^abc]	a, b, c, 以外のすべての１文字に一致
[^-abc]	-, a, b, c 以外のすべての１文字に一致
[^abc\-]	a, b, c, - 以外のすべての１文字に一致
^abc	(先頭)abc に一致
abc$	abc(末尾) に一致
(三十|卅|出)三間堂	三十三間堂, 卅三間堂, 出三間堂に一致
[横横][浜濱濱]	横浜, 横浜, 横濱, 横濱, 横濱, 横濱に一致

付録C：異体字同定について

「横浜」を例に，異体字同定を行う検索と行わない検索を比べてみましょう．

[itaiji_1.py]

```
import json
import codecs
import requests

# 検索条件
query_1 = {
    'query': {
        'conditions': [
            {
                'query': {
                    'field': "title",
                    'term': "横浜",
                    'match': "contain" # 部分一致：異体字同定を行う
                }
            }
        ]
    }
}
query_2 = {
    'query': {
        'conditions': [
            {
                'query': {
                    'field': "title",
                    'term': "横浜",
                    'match': "regex" # 正規表現：異体字同定を行わない
                }
            }
        ]
    }
}

# API「メタデータ検索(HIT数)」を呼び出す
cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search-hits"
result_1 = requests.post(cmd, json=query_1).json()
result_2 = requests.post(cmd, json=query_2).json()

# 検索結果を画面に出力する (pretty printing)
print(json.dumps(result_1, indent=2, ensure_ascii=False))
print(json.dumps(result_2, indent=2, ensure_ascii=False))
```


```
{
  "info": {
    "statusCode": 0,
    "total": 3866
  }
}
{
  "info": {
    "statusCode": 0,
    "total": 3787
  }
}
```

検索数に79件の食い違いが生じました．異体字による「横浜」表記が含まれていることがわかります．（実際にはすべて「横濱」という表記でした．）

異体字同定のためのテーブルはnihuBridgeの「機構本部内蓄積データ」のうち「単漢字異体字データテーブル」として公開しています．
「横」と「浜」はそれぞれ以下の異体字が定義されています．

文字	UCS	文字	UCS	文字	UCS
横	U+6A2A	橫	U+6A6B		
浜	U+6D5C	濱	U+6FF1	濱	U+6EE8

異体字同定を正規表現による検索で表現すると，たとえば以下のようになります．

[itaiji_2.py]

```
import json
import codecs
import requests

# 検索条件
query = {
    'query': {
        'conditions': [
            {
                'query': {
                    'field': "title",
                    'term': "[横橫][浜濱濵]",
                    'match': "regex"
                }
            }
        ]
    }
}

# API「メタデータ検索(HIT数)」を呼び出す
cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/metadatas/search-hits"
response = requests.post(cmd, json=query)
result = response.json()

# 検索結果を画面に出力する (pretty printing)
print(json.dumps(result, indent=2, ensure_ascii=False))
```

```
{
  "info": {
    "statusCode": 0,
    "total": 3866
  }
}
```

付録D：メタデータについて

API「メタデータ検索(HIT数)」およびAPI「メタデータ検索」において， 検索フィールド名および取得フィールド名として指定できるフィールド，ならびにAPI「メタデータ取得」にて取得できるフィールドについての情報を下表に示します．メタデータは，Dublin Core および schema.org との整合性を考慮して設計されています．

4番のフィールド「状態(status)」は，1～5までの値を取り，それぞれ， 1:提供,2:提供準備,3:休止,4:移転,5:廃止の意味を表わします．

16番のフィールド「今後の更新の有無(futureUpdates)」は，0もしくは1の値を取り，それぞれ，0:なし,1:ありの意味を表わします．

30番のフィールド「対象空間範囲(spatial)」は，「(南端の緯度,西端の経度),(北端の緯度,東端の経度)」の形の文字列でデータが格納されています．緯度と経度は十進法度単位で書き表します．すなわち，分と秒を十進小数に変換(1分=1/60，1秒=1/3600）して与えます．南緯と西経は負の数で，北緯と東経は正の数で表します．

31番のフィールド「対象時間範囲(temporal)」は，「始まりの日時,終わりの日時」の形の文字列でデータが格納されています．日時はISO 8601のフォーマットで書かれています．

33～35番のフィールドは，検索フィールド名としては指定できません．

No.	フィールド名	項目名	データ型	Dublin Coreで対応する基本記述要素	Schema.org で対応するタイプ
1	researchResourceId	研究資源ID	文字列	identifier	identifier
2	doi	doi	文字列	identifier	identifier
3	originalId	オリジナルID	文字列	identifier	identifier
4	status	状態	文字列		creativeWorkStatus
5	title	タイトル	文字列	title	name
6	alternativeTitle	別タイトル	文字列	title	name
7	creator	作成者	文字列	creator	creator
8	contributor	編者・監修者	文字列	contributor	contributor

No.	フィールド名	項目名	データ型	Dublin Coreで対応する基本記述要素	Schema.org で対応するタイプ
9	publisher	所蔵者・組織	文字列	publisher	publisher
10	subject	主題	文字列	subject	about
11	keyword	キーワード	文字列	subject	about
12	description	概要・説明	文字列	description	description
13	dateCreated	研究資源が作られた日	文字列	date	dateCreated
14	datePublished	登録日	文字列	date	datePublished
15	dateModified	最終更新日	文字列	date	dateModified
16	futureUpdates	今後の更新の有無	文字列		(creativeWorkStatus)
17	license	利用ライセンス	文字列	rights	license
18	type	研究データのタイプ	文字列	type	genre
19	archiveFormat	研究データのフォーマット（全体）	文字列	format	encodingFormat
20	contentSize	研究データのサイズ	文字列	format	contentSize
21	encodingFormat	研究データのフォーマット（個別）	文字列	format	encodingFormat
22	inLanguage	研究データの言語	文字列	language	inLanguage
23	isBasedOn	元になった資源	文字列	source	isBasedOn
24	link	研究データ	文字列	relation	url
25	sampleLink	サンプルデータ	文字列	relation	
26	metadataFields	メタデータの項目	文字列		
27	metadataLanguage	メタデータの記述言語	文字列		
28	relatedLink	関連データ	文字列		relatedLink
29	resources	上位の研究資源	文字列	relation	relatedLink
30	spatial	対象空間範囲	緯度経度範囲	coverage	spatial
31	temporal	対象時間範囲	日時範囲	coverage	temporal
32	comment	備考	文字列		
33	useCases	利用事例	文字列		
34	operationHistories	作業履歴	文字列		updateAction
35	accessCount	アクセス状況	整数		interactionStatistics

付録E：データベース一覧

検索対象として選べるデータベースとそのデータベースIDは，以下のように，API「データベース一覧取得」によって求めることができます.

[dblist.py]

```
import json
import codecs
import requests

# API「データベース一覧取得」を呼び出す
cmd = "https://api.bridge.nihu.jp/v1/integratedsearch/databases/list"
response = requests.post(cmd, json='')
result = response.json()

# 検索結果を画面に出力する（pretty printing）
#print(json.dumps(result, indent=2, ensure_ascii=False))

# 機関名とデータベース名の一覧を表示する
print('No.,データベースID,データベース名,所蔵機関')
i = 1
for institute in result['institute']:
    for database in institute['database']:
        print(f'{i},{database["databaseSearchId"]},{database["databaseName"]},{institute["instituteName"]}')
        i = i + 1
```

```
No.,データベースID,データベース名,所蔵機関
1,nmjh_kanzousiryou,館蔵資料,国立歴史民俗博物館
2,nmjh_kanzoucyuusei,館蔵中世古文書,国立歴史民俗博物館
3,nmjh_kanzoukinsei_kindai,館蔵近世・近代古文書,国立歴史民俗博物館
4,nmjh_kanzoukisyuutokugawake,館蔵紀州徳川家伝来楽器,国立歴史民俗博物館
5,nmjh_kanzoubukibugu_jitubutu,館蔵武器武具（実物資料）,国立歴史民俗博物館
6,nmjh_kanzoubukibugu_bunken,館蔵武器武具（文献史料）,国立歴史民俗博物館
7,nmjh_kanzounisikie,館蔵錦絵,国立歴史民俗博物館
8,nmjh_kanzou_futokoronitamarumorokuzu,館蔵『懐溜諸屑』,国立歴史民俗博物館
9,nmjh_kanzounomurasyoujirou,館蔵野村正治郎衣裳コレクション,国立歴史民俗博物館
10,nmjh_kanzousensyokuyoukatagami,館蔵染色用型紙,国立歴史民俗博物館
11,nmjh_kanzoujoumonjidaiibutu,館蔵縄文時代遺物,国立歴史民俗博物館
12,nmjh_kanzousousingu,館蔵装身具,国立歴史民俗博物館
13,nmjh_kanzoutakamatukinribon,館蔵高松宮家伝来禁裏本,国立歴史民俗博物館
14,nmjh_kaneakikyouki,兼顕卿記,国立歴史民俗博物館
15,nmjh_tosyomokuroku,歴博図書目録,国立歴史民俗博物館
16,nmjh_nihonsyouen,日本荘園,国立歴史民俗博物館
17,nmjh_syouenkankei,荘園関係文献目録,国立歴史民俗博物館
18,nmjh_jiyuuminkenundou,自由民権運動研究文献目録,国立歴史民俗博物館
```

19,nmjh_munafuda,棟札,国立歴史民俗博物館
20,nmjh_kodai_cyuusei_tosiseikatusi,古代・中世都市生活史,国立歴史民俗博物館
21,nmjh_edosyounin_syokuin,江戸商人・職人,国立歴史民俗博物館
22,nmjh_cyuseiseisatu_seisatu,中世制札（制札）,国立歴史民俗博物館
23,nmjh_cyuseiseisatu_bunken,中世制札（文献）,国立歴史民俗博物館
24,nmjh_chuseitihoutosi_tosi,中世地方都市（都市）,国立歴史民俗博物館
25,nmjh_chuseitihoutosi_bunken,中世地方都市（文献）,国立歴史民俗博物館
26,nmjh_toujikisyutudoiseki_iseki,陶磁器出土遺跡（遺跡）,国立歴史民俗博物館
27,nmjh_toujikisyutudoiseki_bunken,陶磁器出土遺跡（文献）,国立歴史民俗博物館
28,nmjh_doguu,土偶,国立歴史民俗博物館
29,nmjh_kinseiyougyouiseki,近世窯業遺跡,国立歴史民俗博物館
30,nmjh_kinseiyougyoukankei,近世窯業関係主要文献目録,国立歴史民俗博物館
31,nmjh_jyoukanjyoukahakkutu_iseki,城館城下発掘（遺跡）,国立歴史民俗博物館
32,nmjh_jyoukanjyoukahakkutu_bunken,城館城下発掘（文献）,国立歴史民俗博物館
33,nmjh_yayoisekkiiseki_iseki,弥生石器遺跡（遺跡）,国立歴史民俗博物館
34,nmjh_yayoisekkiiseki_zumen,弥生石器遺跡（図面）,国立歴史民俗博物館
35,nmjh_tougokuitabi_syozai,東国板碑（遺跡等）,国立歴史民俗博物館
36,nmjh_tougokuitabi_itabi,東国板碑（板碑）,国立歴史民俗博物館
37,nmjh_tougokuitabi_bunken,東国板碑（文献）,国立歴史民俗博物館
38,nmjh_nihonminzokugakubunken,日本民俗学文献目録,国立歴史民俗博物館
39,nmjh_minzokugoi,民俗語彙,国立歴史民俗博物館
40,nmjh_zokusin,俗信_動植物編,国立歴史民俗博物館
41,nmjh_zokusinsb,俗信_身体・病編,国立歴史民俗博物館
42,nmjh_bunkazai_ronb,文化財材料(色材)知識,国立歴史民俗博物館
43,nmjh_japaneseamerican,日系アメリカ移民,国立歴史民俗博物館
44,nmjh_siebold,シーボルト父子関係資料,国立歴史民俗博物館
45,nmjh_ir,国立歴史民俗博物館学術情報リポジトリ,国立歴史民俗博物館
46,nmjh_jomonyayoi_shurakuiseki,縄文・弥生集落遺跡,国立歴史民俗博物館
47,nijl_syuzou_archive,収蔵歴史アーカイブズ,国文学研究資料館
48,nijl_azumakagami,吾妻鏡,国文学研究資料館
49,nijl_genjimonogatari,絵入源氏物語,国文学研究資料館
50,nijl_21daisyuu,二十一代集,国文学研究資料館
51,nijl_kindaisyosi,近代文献情報（近代書誌・近代画像）,国文学研究資料館
52,nijl_ousyusyozai,コーニツキー版 欧州所在日本古書総合目録,国文学研究資料館
53,nijl_kohitugire,古筆切所収情報,国文学研究資料館
54,nijl_siryousyozaijyouhou_kensaku,史料所在情報・検索,国文学研究資料館
55,nijl_sinnaraehon,新奈良絵本,国文学研究資料館
56,nijl_rekisijinbutu,歴史人物画像,国文学研究資料館
57,nijl_kokubungakuronbun,国文学論文目録,国文学研究資料館

（おわり）