記事
Minoru Horita · 2020年8月27日 7m read

# Python Native APIでNoSQLデータベースにアクセス

NoSQLデータベースという言葉を聞かれたことがあると思います。色々な定義がありますが、簡単に言えば、文字通りSQLを使わない、つまりリレーショナルデータベース(RDB)以外のデータベースのことを指すのが一般的です。

InterSystems IRIS Data Platformでは、テーブルを定義してSQLでデータにアクセスできます。ですから、InterSystems IRIS Data Platformは厳密にNoSQLデータベースというわけではありません。しかし、InterSystems IRISの高パフォーマンスを支える「グローバル」は、40年も前からInterSystemsのコア技術として、現代で言うNoSQLデータベースを提供してきました。本稿では、InterSystems IRISの「グローバル」でグラフ構造を作り、それをPythonでアクセスする方法を紹介します。

本稿で説明する内容は動画でも公開しています。ぜひご覧ください。

## NoSQL

NoSQLに分類されるデータベースには様々なデータモデルを扱うものがあります。以下に代表的なものを挙げます。

- **Key-Value**: キーと値の対応関係を保持する。代表的なデータベース: Redis
- **Document**: JSONやXMLをデータモデルとするもの。代表的なデータベース: MongoDB
- **Graph**: 辺とノードからなるグラフ構造をモデルとするもの。代表的なデータベース: Neo4j

その他、数え切れないほどのデータモデルと製品が存在します。

### NoSQL登場の背景

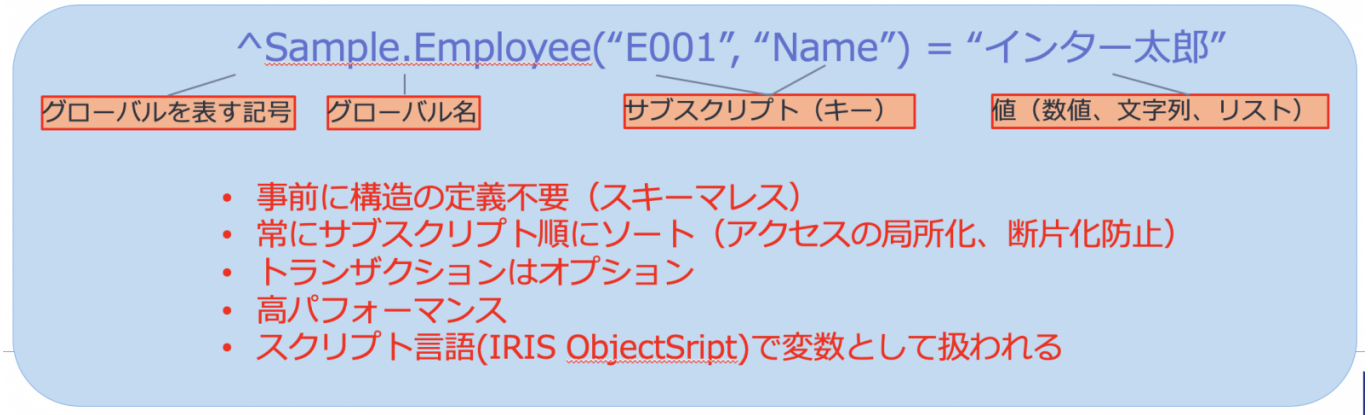
なぜNoSQLというものが登場し、普及に至っているのでしょうか？本稿では、次のような理由を挙げたいと思います。

- **テーブルで表すのが自然ではないデータ構造がある**： 理論的にはどんなデータ構造もテーブルで表し、SQLでアクセスできますが、SQLが複雑になったり、パフォーマンスが出ないなどの問題が発生するケースがあります。そのような場合、テーブルではなく、その構造に合ったデータモデルを使用するほうが良い結果を生みます。
- **分散環境への最適化**
  ：NoSQLの活用が活発になった一つの分野はFacebookなどのSNSのインフラです。大量のデータ、多数のユーザをサポートし、高可用性を得るため、データベースを分散させるのが必須です。RDBは、必ずしも分散環境で最適な能力を発揮できません。また、次に挙げるRDBのトランザクションに対する考え方も、分散環境では実装が難しくなる要因です。
- **トランザクション処理に対する要求**
  ：トランザクション処理とデータの整合性は、RDBが得意とする機能です。銀行口座を管理するシステムなどでは、トランザクション処理とデータの整合性の厳密なサポートが必須なのは言うまでもありません

。しかし、トランザクション処理におけるデータの整合性の維持は、分散環境では実装が大変困難で、特にパフォーマンス性能との両立が難しいと言われています。したがって、厳密な整合性の維持よりも、分散環境におけるパフォーマンスや可用性の向上を優先したい場合、RDBよりもNoSQLの方がニーズにマッチする場合があります。

## グローバル

InterSystems IRISでは、グローバルと呼ばれるデータ構造がデータベースのコアに採用されています。グローバルは、次の図のように、配列変数のようなモデルです。




グローバルは事前に定義が必要ありません（スキーマレス）ので、データ構造を非常に柔軟に設計できます。また、ディスク上、メモリキャッシュ上、どちらにおいても、データは常にサブスクリプト（キー）の順にソートされていますので、データアクセスの局所化・高速化、断片化の防止を図れます。

グローバルは配列変数のような形をしていますが、実際、IRIS ObjectScriptというIRISのスクリプト言語では、メモリ上の変数とほぼ同じシンタックスで使用でき、プログラム開発の見通しが良くなります。グローバルの詳細については、グローバル使用法のドキュメントを参照してください。

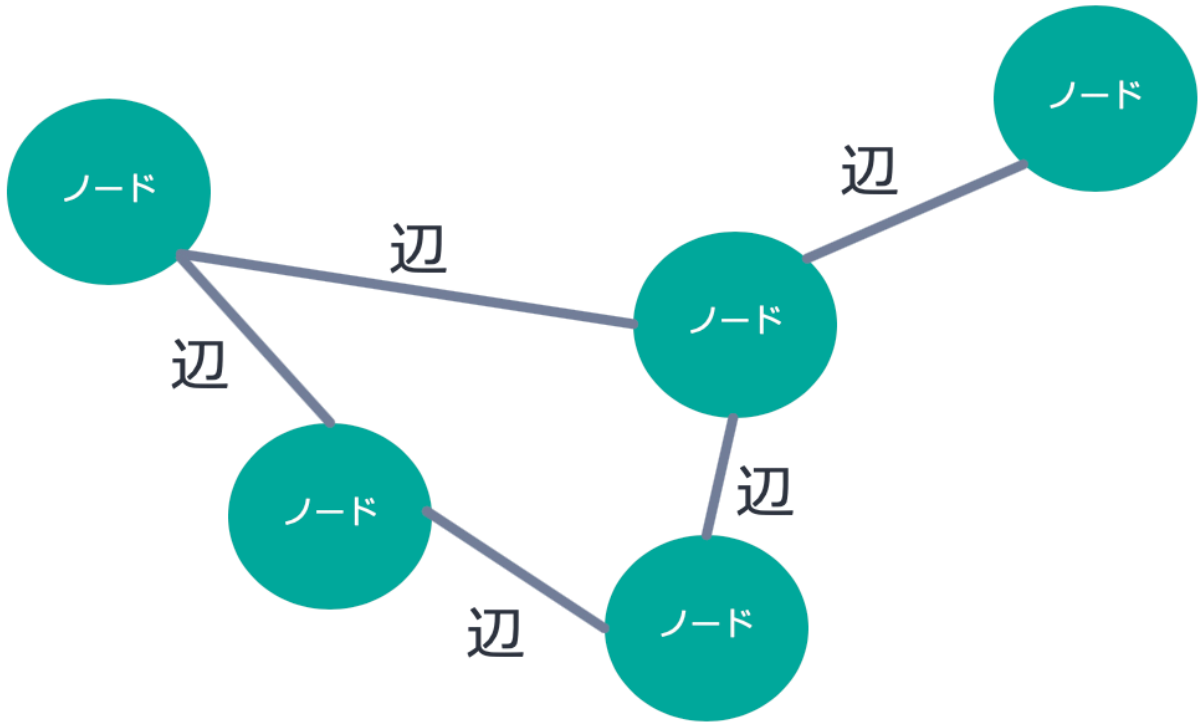
## Pythonからアクセス

Python Native APIを使って、Pythonからグローバルを操作する例を紹介します。

題材としては、Standford大学のSNAP
にある"WikiSpeedia"のデータを利用します。WikiSpeediaとは、与えられた言葉から目標とする言葉に、Wikipediaのリンクだけを辿ってたどり着く速さを競うゲームです。SNAPには、実際のユーザが辿ったリンクの情報が約11万件ありますので、それをInterSystems IRISのグローバルに格納しました。

### グラフ構造

ここで、グラフ構造について説明します。




図に示したのがグラフ構造の基本です。グラフは、ノードと辺から構成されます。辺は、2つのノードを結びつけるものです。

例えば、Facebookの友達の関係は、ノード=ユーザ、辺=友達関係
とすれば表現できます。また、ノード=駅、辺=ある駅の隣の駅、辺のプロパティ=駅と駅の距離
とすれば、路線検索に使えるデータ構造が実現できます。

本稿の例では、ノード=Wikipediaの記事（見出し語）、辺=WikiSpeediaのユーザがリンクをクリックした元の記事とリンク先の記事の関係 としてグラフ構造を利用します。

## グローバルの構造

グローバルの構造を見てみましょう。

```
^Links("Tokyo", "18th_century")=""
^Links("Tokyo", "London")=""
^Links("Osaka", "Aquarium")=""
```

例えば、一番上の行は、ユーザが"Tokyo"という記事から"18thcentury"（18世紀）という記事へのリンクをクリックしたことを表します。また、3行目は、"Osaka"から"Aquarium"（水族館）へクリックしたことを表します。

グローバルは、サブスクリプト（キー）の左から右の方向でアクセスすることに最適化されています。したがって、

```
^Links("???”,"????????")=""
```

という構造が最適であることが分かると思います。また、今回は値は""ですが、辺に何らかの属性を持たせたい場合は、それをグローバルの値に設定すれば良いと思います。

## Pythonのコード

Pythonのコードについて説明します。まずは、importです。

```
import irisnative
import networkx as nx
```

Python Native APIを使用するためには、irisnativeモジュールをインポートします。このモジュールは、IRISをインストールした際、dev/pythonディレクトリに置かれる.whlファイルをpipコマンドでpythonの環境にインストールしておきます。また、グラフ構造保持のために、NetworkXを使用していますので、それもimportしておきます。

```
connection = irisnative.createConnection("localhost", 9091, "user", "horita","horita"
)
iris_native = irisnative.createIris(connection)
```

この2行では、createConnection()でIRISへの接続を行い、createIris()でグローバル操作のインターフェースオブジェクトを生成しています。

```
def addNodes(key, g, d):
    g.add_node(key)
    if d > 3:
        return

    iter = iris_native.iterator("Links", key)
    nodelist = [k for k,v in iter.items()]

    # ????????????3????
    random.shuffle(nodelist)
    nodelist = nodelist[0:3]

    edgelist = [(key, n) for n in nodelist]
    g.add_nodes_from(nodelist)
    g.add_edges_from(edgelist)

    for subk in nodelist:
        addNodes(subk, g, d + 1)
```

addNodes()関数の定義です。この関数は、与えられたグラフのノードからリンクを辿ってグラフ構造をNetworkXのオブジェクトとして構築する関数です。再帰呼び出しによって、3階層目まで辿るようにしています。一番のポイントは、

```
    iter = iris_native.iterator("Links", key)
    nodelist = [k for k,v in iter.items()]
```

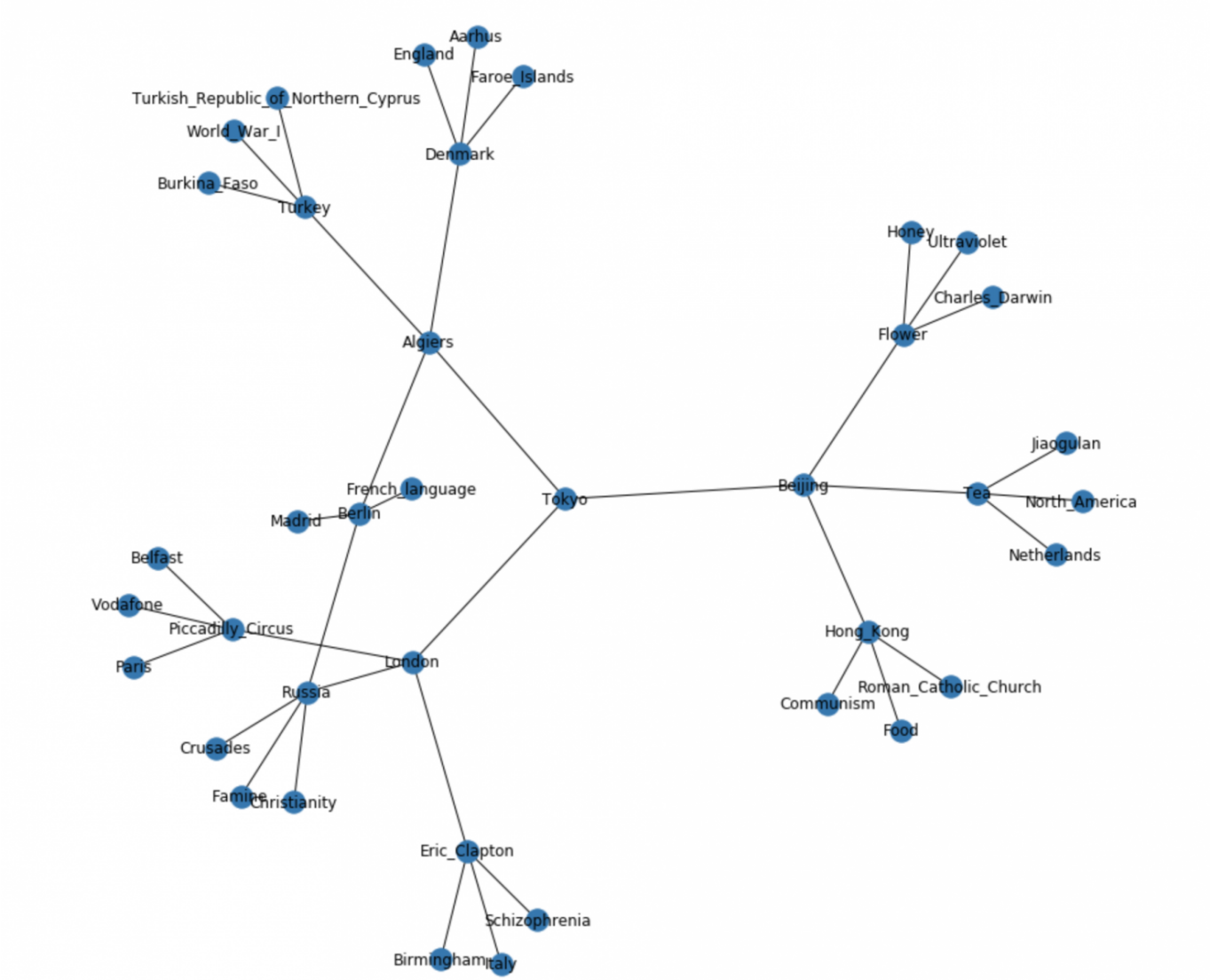
という箇所です。irisnative.iterator("Links", key)では、Python Native APIによって、サブスクリプトのイテレータを作成しています。例えば、key='Tokyo'の場合は、^Links("Tokyo", *)の*の部分にある文字列について繰り返すイテレータになります。

そのイテレータをPythonのforで繰り返し、リストを作ります。このように、グローバルのイテレータがPythonの

繰り返しモデルにうまく適合して、シンプルなコードになっていることが分かります。

```
curkey = 'Tokyo'
G = nx.Graph()
addNodes(curkey, G, 1)
```

そして、"Tokyo"という言葉をキーとしてaddNodes()を呼び出します。これで、"Tokyo"という言葉から辿られたリンクの繋がりが得られます。それが次の図です。




"Tokyo"という言葉からクリックされた言葉のネットワークが表現されています。

## まとめ

InterSystems
IRISのグローバルは、NoSQLデータベースが選択されるようなニーズで力を発揮します。また、Python Native APIによって、Pythonのプログラムから自然な形でグローバルを操作することが可能になります。

ぜひ一度試してみてください。また、質問などをこの記事のコメントに書いて頂ければ大変嬉しいです。

#Python #グラフ #グローバル #ビデオ #Multi-model #InterSystems IRIS #InterSystems IRIS for Health

**ソースURL:**https://jp.community.intersystems.com/post/python-native-api%E3%81%A7nosql%E3%83%87%E3%83%BC%E3%82%BF%E3%83%99%E3%83%BC%E3%82%B9%E3%81%AB%E3%82%A2%E3%82%AF%E3%82%BB%E3%82%B9