

記事

[Toshihiko Minamoto](#) · 2022年1月27日 11m read

Python用IRISネイティブAPI

はじめに

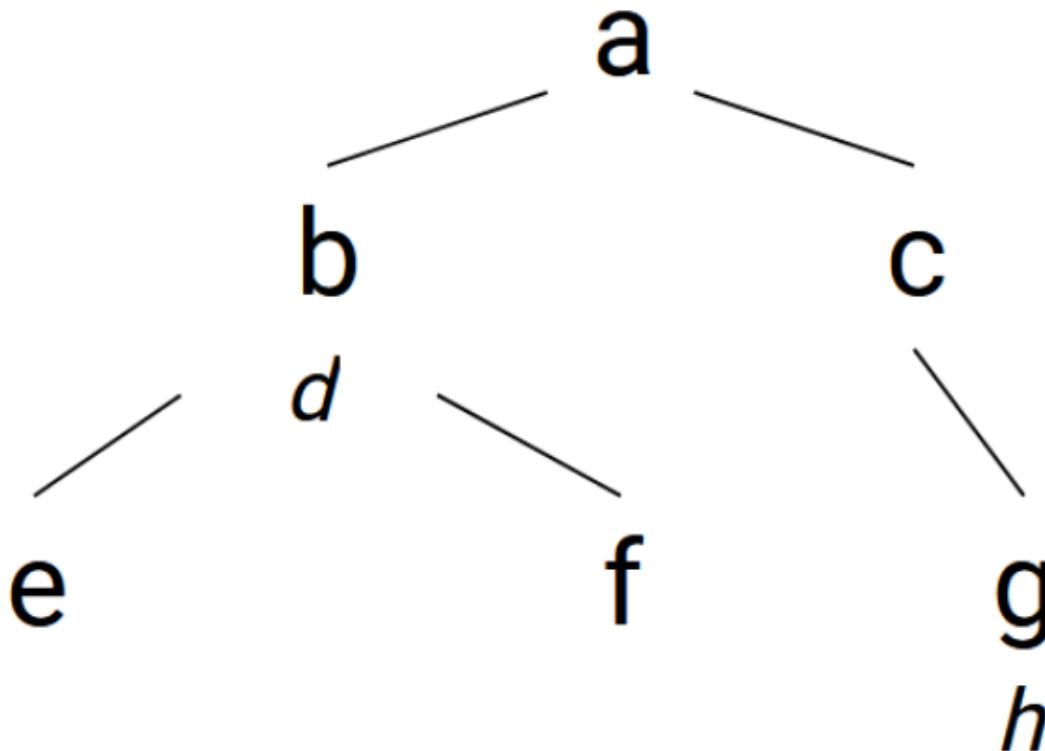
バージョン2019.2より、InterSystems

IRISは、高性能データアクセス手法としてPython用のネイティブAPIを提供してきました。ネイティブAPIを使用すると、ネイティブのIRISデータ構造と直接対話することができます。

グローバル

InterSystems開発者であれば、おそらくすでにグローバルを理解していることでしょう。ここでは、復習目的でその基本を確認しますが、次のセクションに進んでも構いません。

InterSystems IRISはグローバルを使用してデータを格納しています。グローバルは、値の有無に関係なくノードとサブノードで構成されるスパース配列です。以下に、グローバルの概念的な例を示します。



この例では、aはルートノードであり、グローバル名と呼ばれています。

各ノードには、グローバル名と1つまたは複数のサブスクリプト

(サブノードの名前)で構成されるノードアドレスがあります。

aにはサブスクリプトbとcがあるため、それらのノードアドレスは、a->bおよびa->cとなります。

ノードa->bとa->c->gには値 (dとh) があり、ノードa->b->eとa->b->fには値がありません。

ノードa->bにはサブスクリプトeとfがあります。

この構造の詳細な説明は、[InterSystems dookbook『グローバルの使用法』](#)に記載されています。

グローバルの読み取りと書き込み

ネイティブのPython APIでは、IRISグローバルからデータを読み取りとそれへの書き込みを直接行えます。irisnativeパッケージは[GitHub](#)で提供されています。または、InterSystems IRISがローカルマシンにインストールされている場合は、インストールディレクトリのdev/pythonサブディレクトリにあります。

irisnative.createConnection関数は、IRISへの接続を作成し、irisnative.createIris関数は、この接続から、グローバルを操作するために使用するオブジェクトを取得します。このオブジェクトには、グローバルの読み書きに使用するgetメソッドとsetメソッド、そしてノードとそのサブノードの削除に使用するkillメソッドがあります。また、isDefinedメソッドもあり、このメソッドによって、リクエストされたノードが存在しない場合は「0」、値があり子孫がない場合は「1」、値がなく子孫がある場合は「10」、値と子孫がある場合は「11」を返します。

```
import irisnative
conn = irisnative.createConnection("127.0.0.1", 51773, "USER", "<user>", "<password>")
iris = irisnative.createIris(conn)
iris.set("value", "root", "sub1", "sub2") # sets "value" to root->sub1->sub2
print(iris.get("root", "sub1", "sub2"))
print(iris.isDefined("root", "sub1"))
iris.kill("root")
conn.close()
```

また、特定のノードのサブノードをループするiteratorメソッドもあります。
(使用方法は次のセクションで説明します。)

各メソッドの詳細な説明については、[APIドキュメント](#)をご覧ください。

サンフランシスコのGTFSトランジットデータファイル

グローバルにデータを格納する

General Transit Feed Specification (GTFS) は、公共交通機関の時刻表と経路に関するフォーマットです。IRISネイティブAPIを使用して、2019年6月10日の[サンフランシスコのGTFSデータ](#)を処理する方法を見てみましょう。

まず、データファイルの情報をIRISグローバルに格納します。
(このデモでは、すべてのファイルと列を使用するわけではありません。)
ファイルはCSV形式で、最初の行に列名、残りの行にデータが含まれます。
Pythonで、必要なインポートを開始し、IRISへの接続を確立します。

```
import csv
import irisnative

conn = irisnative.createConnection("127.0.0.1", 51773, "USER", "<user>", "<password>")
iris = irisnative.createIris(conn)
```

列名とデータに基づき、各ファイルの実用的なツリー構造を構築し、iris.setを使用してグローバルにデータを格納できます。

stops.txtファイルから始めましょう。このファイルにはサンフランシスコのすべての公共交通機関の停留所・停車駅が含まれます。このファイルのstopid列とstopname列のみを使用します。これらを、stop

IDをサブスクリプト、stop nameをノードの値として使用し、1つのレイヤーのノードを持つツリー構造内のstopsというグローバルに格納します。したがって、この構造は「stops → [stopid]=[stopname]」のようになります。（この記事では、サブスクリプトがリテラルではなくデータファイルから読み取った値である場合に、角括弧を使用しています。）

```
with open("stops.txt", "r") as csvfile:
    reader = csv.reader(csvfile)
    next(reader) # Ignore column names

    # stops -> [stop_id]=[stop_name]
    for row in reader:
        iris.set(row[6], "stops", row[4])
```

csv.readerは、カンマ区切りの値を保持するリストのイテレータを返します。最初の行には列名が含まれているため、next(reader)を使ってその行をスキップします。stop nameをstops -> [stopid]の値として設定するために、iris.setを使用します。

次はroutes.txtファイルです。このファイルのroute_type、route_id、route_short_name、およびroute_long_name列を使用します。実用的なグローバル構造は、routes -> [route_type] -> [route_id] -> [route_short_name]=[route_long_name]です。（route type（経路タイプ）は、路面電車の場合は「0」、バスの場合は「3」、ケーブルカーの場合は「5」です。）このCSVファイルからデータを読み取って、まったく同じようにグローバルに格納します。

```
with open("routes.txt", "r") as csvfile:
    reader = csv.reader(csvfile)
    next(reader) # Ignore column names

    # routes -> [route_type] -> [route_id] -> [route_short_name]=[route_long_name]
    for row in reader:
        iris.set(row[0], "routes", row[1], row[5], row[8])
```

各経路には区間

があります。これはtrips.txtファイルに格納されているもので、このファイルのroute_id、direction_id、trip_headsign、およびtrip_id列を使用します。区間は、trip ID（後でstop timesファイルで確認します）で一意に識別されています。ある経路の区間は、方向を基準とした2つのグループに分けられ、方向にはヘッドサインが関連付けられています。これを踏まえると、ツリー構造はtrips -> [route_id] -> [direction_id]=[trip_headsign] -> [trip_id]となります。

ここでは2つのiris.set呼び出しが必要です。direction IDノードに値を設定する呼び出しと、trip IDの値のないノードを作成する呼び出しです。

```
with open("trips.txt", "r") as csvfile:
    reader = csv.reader(csvfile)
    next(reader) # Ignore column names

    # trips -> [route_id] -> [direction_id]=[trip_headsign] -> [trip_id]
    for row in reader:
        iris.set(row[3], "trips", row[1], row[2])
        iris.set(None, "trips", row[1], row[2], row[6])
```

最後に、stop times（停止時刻）を読み取って格納します。これらは、stop_times.txtファイルに格納されており、このファイルのstop_id、trip_id、stop_sequence、departure_time列を使用します。最初のオプションにはstop_times -> [stop_id] -> [trip_id] -> [departure_time]を使用することができます。またはstop sequence（停止順）を維持する場合は、stop_times -> [stop_id] -> [trip_id] -> [stop_sequence]=[departure_time]となります。

```
with open("stop_times.txt", "r") as csvfile:
    reader = csv.reader(csvfile)
    next(reader) # Ignore column names

    # stoptimes -> [stop_id] -> [trip_id] -> [stop_sequence]=[departure_time]
    for row in reader:
        iris.set(row[2], "stoptimes", row[3], row[0], row[4])
```

ネイティブAPIを使用してデータをクエリする

次の目標は、**特定の名前の停留所・停車駅のすべての出発時刻を見つけること**です。

まず、特定のstop nameからstop IDを取得し、そのIDを使ってstoptimesから関連する時刻を見つけます。

```
iris.iterator("stops")
```

呼び出しを使うと、stopsルートノードのサブノードを反復できます。サブスクリプトと値のペアを反復したいため（指定された名前を持つ値を比較し、一致すればスフにサブスクリプトを知ることができるため）、イテレータで.items()を呼び出し、戻り値の型を (subscript, value) のタプルに設定します。次に、これらすべてのタプルを反復して、正しいstopを見つけ出します。

```
stop_name = "Silver Ave & Holyoke St"
```

```
iter = iris.iterator("stops").items()
```

```
stop_id = None
```

```
for item in iter:
    if item[1] == stop_name:
        stop_id = item[0]
        break
```

```
if stop_id is None:
    print("Stop not found.")
    import sys
    sys.exit()
```

ノードが多数ある場合、反復によってキーの値でキーでルックアップするのはあまり効率的ではありません。これを避けるには、サブスクリプトがstop nameで値がIDである別の配列を用意するとよいでしょう。すると、value --> key ルックアップは、この新しい配列への1つのクエリで構成されます。

または、stop nameも一意であるため、stop IDの代わりに、コードのあらゆる箇所でstop nameを識別子として使用することもできます。

お分かりのとおり、stopsの数が膨大にある場合は、この検索に時間が掛かります。これは「フルスキャン」とも呼ばれています。

ただし、グローバルを利用して、値のIDを持つキーが名前となる逆配列を作成することができます。

```
iter = iris.iterator("stops").items()
```

```
stop_id = None
```

```
for item in iter:
    iris.set(item[0], "stopnames", item[1])
```

インデックスが名前でも値がIDのstopnamesグローバルがあることで、名前でもstopidを見つける上記のコードは、フルスキャン検索を行わずに実行する以下のコードに変更されます。

```
stop_name = "Silver Ave & Holyoke St"
stop_id=iris.get("stopnames", stop_name)
if stop_id is None:
    print("Stop not found.")
    import sys
    sys.exit()
```

この時点で、stop times（停止時刻）を見つけることができます。サブツリーstoptimes -> [stopid]にはtrip IDサブノードがあり、このサブノードにはstop timesサブノードがあります。trip IDではなく、stop timesのみに関心があるため、すべてのtrip IDを反復して、それぞれのすべてのstop timesを収集します。

```
all_stop_times = set()

trips = iris.iterator("stoptimes", stop_id).subscripts()
for trip in trips:
    all_stop_times.update(iris.iterator("stoptimes", stop_id, trip).values())
```

ここでは、イテレータに.items()を使用していませんが、trip IDはサブスクリプトであるため.subscripts()と.values()を使用します。または、値とdeparture times（出発時刻）に関心がある場合には、下位レイヤー（[stopsequence]=[departuretime]）を使用します。.update呼び出しは、イテレータのすべての項目を既存のセットに追加します。すると、セットにはすべての（一意の）stop timesが含まれます。

```
for stop_time in sorted(all_stop_times):
    print(stop_time)
```

では、もう少し複雑にしてみましょう。
ある停車駅・停留所のすべての出発時刻を見つける代わりに、ある停車駅・停留所におけるroute IDが指定された特定の経路（両方向）のみの出発時刻を見つけることにします。stop nameからstop IDを見つけ出すコードは、そのまま使用できます。次に、特定の経路のすべてのtrip IDを取得します。これらのIDは、departure timesを取得する際の追加の制限として使用されます。

trips -> [routeid]のサブセットは2つの方向に分割され、すべてのtrip IDがサブノードとなります。前と同じようにdirectionsを反復し、すべてのdirectionsのサブノードをセットに追加します。

```
route = "14334"

selected_trips = set()

directions = iris.iterator("trips", route).subscripts()
for direction in directions:
    selected_trips.update(iris.iterator("trips", route, direction).subscripts())
```

次のステップとして、取得されたstop IDを[stopid]、選択されるtrip IDを[tripid]とするstoptimes -> [stopid] -> [tripid]のすべてのサブノードの値を見つけ出します。
selectedtripsセットを反復して、すべての関連する値を見つけます。

```
all_stop_times = set()

for trip in selected_trips:
    all_stop_times.update(iris.iterator("stoptimes", stop_id, trip).values())
```

```
for stop_time in sorted(all_stop_times):
    print(stop_time)
```

最後の例では、isDefined関数の使用方法を示します。以前に記述したコードを拡張し、route IDをハードコーディングする代わりに、経路の短縮名を指定すると、route IDはそれに基づいて取得されるようになります。経路名のあるノードは、ツリーの最下位レイヤーにあります。その上のレイヤーにはroute IDが含まれます。すべての経路タイプを反復してからすべてroute IDを反復すると、ノードroutes -> [routetype] -> [routeid] -> [routeshortname]が存在して値がある（isDefinedが1を返す）場合は、探しているIDが[routeid]であることがわかります。

```
route_short_name = "44"
route = None

types = iris.iterator("routes").subscripts()
for type in types:
    route_ids = iris.iterator("routes", type).subscripts()
    for route_id in route_ids:
        if iris.isDefined("routes", type, route_id, route_short_name) == 1:
            route = route_id

if route is None:
    print("No route found.")
    import sys
    sys.exit()
```

このコードは、ハードコーディングされたroute = "14334"の行の代わりになります

すべてのIRIS操作が完了したら、データベースへの接続を閉じることができます。

```
conn.close()
```

今後の内容

PythonのネイティブAPIを使ってInterSystems IRISのデータ構造にアクセスする方法を説明し、それをサンフランシスコの公共交通機関データに適用しました。APIの詳細な説明については、[ドキュメント](#)をご覧ください。ネイティブAPIは、[Java](#)、[.NET](#)、および[Node.js](#)で提供されています。

[#API #Python #InterSystems IRIS](#)

ソースURL:

<https://jp.community.intersystems.com/post/python%E7%94%A8iris%E3%83%8D%E3%82%A4%E3%83%86%E3%82%A3%E3%83%96api>