記事
Mihoko Iijima · 2022年2月28日 7m read

Embedded Python を使ってレシート（JPG）の中身を IRIS に登録してみました

開発者のみなさん、こんにちは。

今回は、スーパーやコンビニでもらうレシートを写真で撮り、OCR
を使ってレシートの画像から文字列を切り出して IRIS に登録する流れを試してみました。

サンプルでは、Google の Vision API を利用してレシートの JPG
画像から購入物品をテキストで抽出しています。

サンプルコード一式　https://github.com/Intersystems-jp/iris-embeddedpython-OCR

最初、オンラインラーニングで使っている 「tesseract-OCR」を使ってみようと思ったのですが、レシートには半角カナが混在していたりで、半カナがなかなかうまく切り出せず、あきらめました・・（半角カナがなかったら日本語もばっちり読めていたのですが・・）

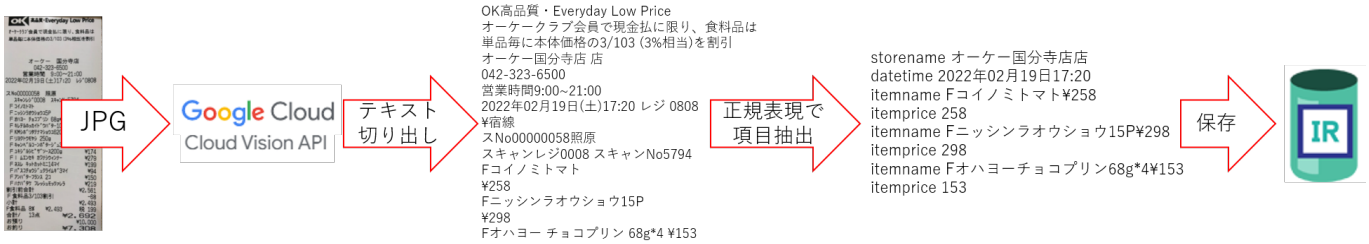
もし、tesseract-OCR で半カナを切り出す良い方法をご存知の方いらっしゃいましたら、ぜひ教えてください！

今回試すにあたり、Vision API の使い方を詳しく書いているページがありましたのでコードなど参考させていただきました。ありがとうございました。

【Google Colab】Vision APIで『レシートOCR』

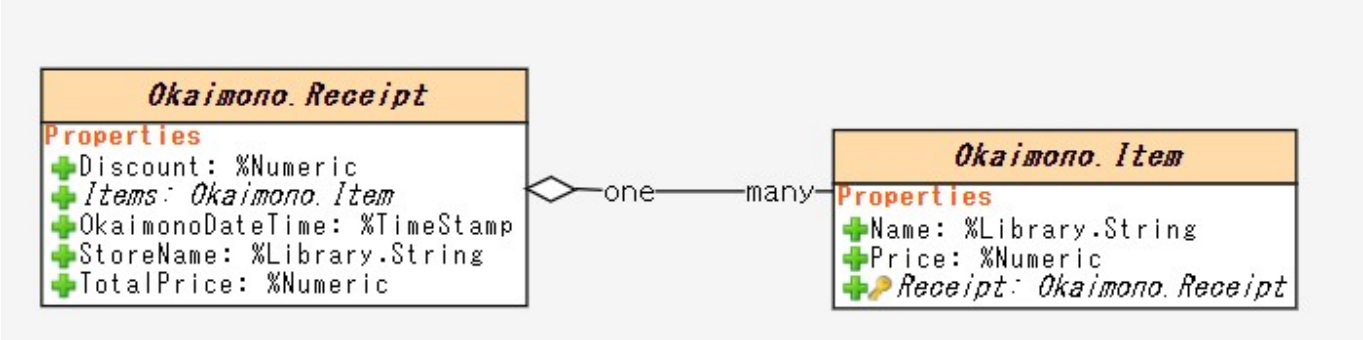
全体の流れですが、レシートの写真を JPG にし、Python の google-cloud-vision モジュールを使用して JPG を Vision API に渡し、テキストを切り出します。

切り出されたテキストを画像の位置に合わせて一旦ソートし、レシートに記載されているものが店名なのか、商品名なのか、金額なのか、など、IRIS に登録したい項目を Python の正規表現モジュール re
を使用してチェックし IRIS に登録しています。



IRIS にデータを登録する部分はSQLでもできますが、今回のサンプルでは1対多のリレーションシップを使った永続クラスを利用しています。

（永続クラスに対してSQLでも操作ができるので、お好みの文法で操作できます）



- 1側クラス：Okaimono.Receipt

レシートの基本情報を格納します。（店名、買い物時間、購入合計、割引金額など）

- 多側クラス：Okaimono.Item

レシートに記載された詳細項目の項目名と金額を格納します。

サンプルの実行方法については、こちらをご参照ください。

以下、コードについて少しご紹介します。

Python 側のコード：receipt.py は、ほぼこちらの素晴らしい記事　【Google Colab】Vision APIで『レシートOCR』 を参考にさせていただきましたので、詳しくは記事をご参照ください。

Vision API で切り出したテキストですが、IRISへ登録するために必要な情報を正規表現で取り出して Python の list に設定し、receiptIRIS.py に渡して IRIS に保存しています。

レシートの項目ですが、お店によって表記が様々だったため（半カナ混在だったり全角だけだったり、記号が付いたりつかなかったり、など）、ご紹介するサンプルでは「オーケー」のレシートに合った正規表現を利用しています。
コンビニや他スーパーの場合、サンプルの正規表現に合わないと思いますので、receipt.py の 72 行目～ 80 行目辺りをご変更いただくことで、お好みのレシートを読めるようになります。ぜひお試しください。

今回、IRIS に登録するデータは、1対多のリレーションシップクラスを利用しているので、データの登録には SQL ではなくオブジェクトの文法を利用しています（もちろんSQLでも登録できます）。

具体的にどんなコードを書いているかというと、前回の記事「Embedded Python 試してみました」と同様に、Python から IRIS 内のクラスを操作するための iris モジュールのインポートと、クラスの操作に使用する iris.cls() を利用しています。

Okaimono.Receipt のインスタンス生成は以下の通りです。

```
import iris
receipt=iris.cls("Okaimono.Receipt")._New()
receipt.StoreName="?×??"
```

そして、作成された Python の list を読みながら、レシートの詳細項目を登録します。

コードはこんな感じです。

Okaimono.Item のインスタンスを生成し、Name プロパティとPrice プロパティに値を登録し、

```
item=iris.cls("Okaimono.Item")._New()
item.Name="???"
item.Price="100"
```

Okaimono.Item のインスタンス と Okaimono.Receipt のインスタンスを関連付けます。（Okaimono.Item
クラスの Receiptリレーションシッププロパティを利用して Okaimono.Receipt
クラスのインスタンスを代入しています）

```
item.Receipt=receipt
```

この方法の他に、1側の Okaimono.Receipt の Items プロパティ（リレーションシッププロパティ）に提供される Insert()
メソッドを使用して Okaimono.Item のインスタンスを Okaimono.Receipt
のインスタンスに割り当てる方法もあります。（どちらか一方の割り当て方法で大丈夫です）

```
receipt.Items.Insert(item)
```

上記操作をレシートに記載された購入品目分実行し、最後にレシートを IRIS に保存します。

```
st=receipt._Save()
```

これで、Okaimono.Receipt に紐づく Okaimono.Item が一括で保存されます。

以下の文例は、2つの商品を購入したときのレシート登録例です。

```
USER>do ##class(%SYS.Python).Shell()

Python 3.9.5 (default, Jan 25 2022, 13:57:42) [MSC v.1927 64 bit (AMD64)] on win32
Type quit() or Ctrl-D to exit this shell.
>>> import iris
>>> receipt=iris.cls("Okaimono.Receipt")._New()
>>> receipt.StoreName="??????"
>>> receipt.TotalPrice=778
>>> it1=iris.cls("Okaimono.Item")._New()
>>> it1.Name="???"
>>> it1.Price=598
>>> receipt.Items.Insert(it1)
1
>>> it2=iris.cls("Okaimono.Item")._New()
>>> it2.Name="????"
>>> it2.Price=180
>>> it2.Receipt=receipt
>>> receipt._Save()
1
```

>>>

登録されたデータを確認します。

IRIS
では、１対多のリレーションシップ定義を行
った場合、多側クラス（Okaimono.Item）のリレーションシップ用プロパティ：Receipt
が外部キーカラムのようにテーブルに投影されます。

以下、確認例です。

まずは、Okaimono.Receipt データを確認します。

```
SELECT * FROM Okaimono.Receipt
```

ID	Discount	OkaimonoDateTime	StoreName	TotalPrice
1	-68.00	2022-02-19 17:20:00	オーケー国分寺店店	2692.00
2			テストストア	778.00

上記実行例では、「テストストア」のデータは **ID=2** で登録されています。

続いて、Okaimono.Item データを確認します。

```
SELECT * FROM Okaimono.Item
```

ID	Name	Price	Receipt
1	Fコイノミトマト¥258	258.00	1
2	Fニッシンラオウショウ15P¥298	298.00	1
3	Fオハヨーチョコプリン68g*4¥153	153.00	1
4	Fキレテルホッカイドウバター100g¥238	238.00	1
5	FKMシボリタテナマショウ1620ml	328.00	1
6	Fリョクトウモヤシ250g	28.00	1
7	Fキャンベルコーンポタージュ305g	143.00	1
8	Fユキジルシピザソース200g	174.00	1
9	FIムエンセキカワナシウィンナー¥279	279.00	1
10	Fネスレキットカットミニ14マイ	199.00	1
11	Fバスコチョウジュクライムギ3マイ	94.00	1
12	Fアンパターフランス2コ	150.00	1
13	Fハナパタケフレッシュモッツァレラ¥219	219.00	1
14	お弁当	598.00	2
15	ジュース	180.00	2

Receipt 列には、リレーションシッププロパティで定義した1側クラス（Okaimono.Receipt）の ID が登録されていることがわかります。

では、次に、Receipt が 2（Okaimono.Receipt の ID=2）の Okaimono.Item 全情報と Okaimono.Receipt の StoreName と TotalPrice を取得してみます。

```
SELECT *,Receipt->StoreName,Receipt->TotalPrice FROM Okaimono.Item Where Receipt=2
```

ID	Name	Price	Receipt	StoreName	TotalPrice
14	お弁当	598.00	2	テストストア	778.00
15	ジュース	180.00	2	テストストア	778.00

IRIS 独自の**矢印構文（->）**を利用しています。この構文はオブジェクトの定義によって作成された外部キー用カラムを利用して、参照先テーブルのカラムを指定できる文法で、**Receipt->StoreName** は、Okaimono.Item テーブルに紐づく Okaimono.Receipt テーブルの StoreName カラムを指定しています。

これは、1対多のリレーションシップやオブジェクト参照を利用している場合に利用できる方法で、内部的には左外部結合と一緒の処理を行っています。

IRIS 独自の書き方ではありますが、意外とすっきりと記述できるので見やすさ重視の場合に利用いただくケースもあります。

矢印構文について詳細はドキュメントもご参照ください。

ちなみに、Pythonシェル上で 1件の Okaimono.Receipt をオープンし、Okaimono.Receipt に紐づく Okaimono.Item を取得する場合のオブジェクトの文法は以下の通りです。

ID=2 の Okaimono.Receipt をオープンします。

```
>>> r1=iris.cls("Okaimono.Receipt")._OpenId(2)
```

何個の Okaimono.Item と紐づいているか確認するには、Okaimono.Receipt クラスのリレーションシッププロパティ＝Items に対して Count() を使用します。

```
>>> r1.Items.Count()
2
```

1件目の Okaimono.Item を取得する例は以下の通りです。

```
>>> it1=r1.Items.GetAt(1)
>>> it1.Name
'???'
>>> it1.Price
598
```

2件目も確認してみます。

```
>>> it2=r1.Items.GetAt(2)
>>> it2.Name
'????'
>>> it2.Price
180
```

こんな感じで、SQLからもオブジェクトからも操作できることが確認できました。

Embedded Python の登場で、Python の便利なモジュールや公開されているサンプルコードが IRIS から実行しやすくなりました。

みなさんのお手元でもぜひお試しください！

そして、お試しいただいた内容など、お気軽にコミュニティに投稿してみてください！お待ちしてます

#Embedded Python #Python #SQL #オブジェクトデータモデル #InterSystems IRIS #InterSystems IRIS for Health