文章
Michael Lei · 五月 8, 2021 阅读大约需 8 分钟

将 Python JDBC 连接到 IRIS 数据库 - 快速笔记

关键字：Python，JDBC，SQL，IRIS，Jupyter Notebook，Pandas，Numpy ，机器学习

1. 目的

这是一个用于演示的 5 分钟快速笔记，通过 Jupyter Notebook 中的 Python 3 调用 IRIS JDBC 驱动程序，以经由 SQL 语法从 IRIS 数据库实例读取数据和向 IRIS 数据库实例写入数据。

去年，我发表了关于将 Python 绑定到 Cache 数据库的简要笔记（第 4.7 节）。 如何使用 Python 挂入 IRIS 数据库以将其数据读入 Pandas 数据框和 NumPy 数组进行常规分析，然后再将一些经过预处理或标准化的数据写回 IRIS 中，准备进一步用于 ML/DL 管道，现在可能是时候回顾一些选项和讨论了。

一些立即浮现的快速**选项**：

1. **ODBC**：Python 3 和原生 SQL 的 PyODBC？
2. **JDBC**：Pyhton 3 和原生 SQL 的 JayDeBeApi？
3. **Spark**：PySpark 和 SQL？
4. **Python Native API for IRIS**：超越先前的 Python Binding for Cache？
5. **IPtyhon Magic SQL %%sql**? 它可以支持 IRIS 了吗？

这里有漏掉其他选项吗？ 我有兴趣尝试任何选项。

2. 范围

我们是不是应该从普通的 JDBC 方法开始？ 下一个快速笔记将总结 ODBC、Spark 和 Python Native API。

范围内：

此快速演示涉及以下常见组件：

Anaconda
Jupyter Notebook
Python 3
JayDeBeApi
JPyPe
Pandas
NumPy
一个 IRIS 2019.x 实例

范围外：

本快速笔记不会涉及以下内容，但它们也很重要，可以使用特定的站点解决方案、部署和服务单独解决：

安全端到端。
非功能性能等。
问题排查和支持。
许可。

3. 演示

3.1 运行 IRIS 实例：

我只运行了一个 IRIS 2019.4 容器，作为 “ 远程 ” 数据库服务器。 您可以使用任何您有权利访问的 IRIS 实例。

zhongli@UKM5530ZHONGLI MINGW64 /c/Program Files/Docker Toolbox
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
d86be69a03ab quickml-demo "/iris-main" 3 days ago Up 3 days (healthy) 0.0.0.0:9091->51773/tcp,
0.0.0.0:9092->52773/tcp quickml

3.2 Anaconda 和 Jupyter Notebook：

我们将在笔记本电脑中重用相同的设置方法，这里对应 Anaconda（第 4.1 节），这里对应 Jupyter Notebook（第 4 节）。 Python 3.x 在这一步安装。

3.3 安装 JayDeBeApi 和 JPyPe：

启动 JupyterNotebook，然后在其单元格中运行以下内容**设置 Python-to-JDBC/Java 桥**：

!conda install --yes -c conda-forge jaydebeapi

JayDeBeApi 在撰写本文时（2020 年 1 月）使用 JPype 0.7，该版本由于一个已知错误无法运行，必须降级为 0.6.3
!conda install --yes -c conda-forge JPype1=0.6.3 --force-reinstall

3.4 通过 JDBC 连接到 IRIS 数据库

这里有一个正式的使用 JDBC 连接到 IRIS 的文档。

对于通过 JDBC 执行 Python SQL，我以下面的代码为例。 它连接到此 IRIS
实例的 “ USER ” 命名空间内的数据表 “ DataMining.IrisDataset ” 。
1. Set environment variables, if necessary
#import
os
#os.environ['JAVA_HOME']='C:\Progra~1\Java\jdk1.8.0_241'
#os.environ['CLASSPATH'] = 'C:\InterS
ystems\IRIS20194\dev\java\lib\JDK18\intersystems-
jdbc-3.0.0.jar'
#os.environ['HADOOP_HOME']='C:\hadoop\bin' #winutil binary must be in Hadoop's Home
2. Get jdbc connection and cursor
import jaydebeapi
url =
"jdbc:IRIS://192.168.99.101:9091/USER"
driver = 'com.intersystems.jdbc.IRISDriver'
user =
"SUPERUSER"
password = "SYS"
#libx =
"C:/InterSystems/IRIS20194/dev/java/lib/JDK18"
jarfile =
"C:/InterSystems/IRIS20194/dev/java/lib/JDK18/intersystems-jdbc-3.0.0.jar"
**conn = jaydebeapi.connect(driver, url, [user, password], jarfile)
curs = conn.cursor()**
3. specify the source data table
dataTable = 'DataMining.IrisDataset'

4. Get the result and display
curs.execute("select TOP 20 * from %s" %
dataTable)
result = curs.fetchall()
print("Total records: " + str(len(result)))
for i in
range(len(result)):
 print(result[i])
5. CLose and clean - I keep them open for next
accesses.
#curs.close()
#conn.close()

```
Total records: 150
(1, 1.4, 0.2, 5.1, 3.5, 'Iris-setosa')
(2, 1.4, 0.2, 4.9, 3.0, 'Iris-setosa')
(3, 1.3, 0.2, 4.7, 3.2, 'Iris-setosa')
... ...
(49, 1.5, 0.2, 5.3, 3.7, 'Iris-setosa')
(50, 1.4, 0.2, 5.0, 3.3, 'Iris-setosa')
(51, 4.7, 1.4, 7.0, 3.2, 'Iris-versicolor')
... ...
(145, 5.7, 2.5, 6.7, 3.3, 'Iris-virginica')
... ...
(148, 5.2, 2.0, 6.5, 3.0, 'Iris-virginica')
(149, 5.4, 2.3, 6.2, 3.4, 'Iris-virginica')
(150, 5.1, 1.8, 5.9, 3.0, 'Iris-virginica')
```

测试表明 JDBC 上的 Python 可以正常运行。 以下只是常规 ML
管道的一些常规数据分析和预处理，由于我们可能会在后续的演示和比较中反复涉及，因此为方便起见在这里附上。

3.5 将 SQL 结果转换为 Pandas DataFrame，再转换为 NumPy 数组

如果还没有安装 Pandas 和 NumPy 软件包，可以通过 Conda 安装，类似于上面 3.3 节。

然后运行以下示例：

```
### transform SQL results "sqlData"to Pandas dataframe "df", then further to NumPy array "arrayN" for further ML pipelines
import pandas as pd
sqlData = "SELECT * from DataMining.IrisDataset"
df= pd.io.sql.readsql(sqlData, conn)
df = df.drop('ID', 1)
df = df[['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Species']]
# set the labels to 0, 1, 2, for NumPy matrix
df.replace('Iris-setosa', 0, inplace=True)
df.replace('Iris-versicolor', 1, inplace=True)
df.replace('Iris-virginica', 2, inplace=True)
# turn dataframe into Numpy array
arrayN = df.tonumpy()
### 6. CLose and clean - if connection is not needed anymore?
#curs.close()
#conn.close()
```

我们例行查看一下当前数据：

```
df.head(5)
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df.describe()
```

现在，我们得到了一个 DataFrame，以及一个来自源数据表的标准化 NumPy 数组。

当然，我们在这里可以尝试各种常规分析，一个 ML 人员会按照下述步骤开始，在 Python 中替换 R（链接）。

数据源引自此处

3.6 拆分数据并通过 SQL 写回 IRIS 数据库：

当然，我们可以像往常一样将数据拆分为训练集和验证集或测试集，然后将它们写回临时数据库表，实现 IRIS 一些即将推出的精彩 ML 功能：

```
  import numpy as np
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
  # keep e.g. 20% = 30 rows as test data; trained on another e.g. 80% = 120 rows
X = arrayN[:,0:4]
y = arrayN[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20, random_state=1, shuffle=True)
  # make 80% of random rows into a Train set
labels1 = np.reshape(Y_train,(120,1))
train = np.concatenate([X_train, labels1],axis=-1)
  # make 20% of left rows into Test set
lTest1 = np.reshape(Y_validation,(30,1))
test = np.concatenate([X_validation, lTest1],axis=-1)
  # write the train data set into a Pandas frame
dfTrain = pd.DataFrame({'SepalLength':train[:, 0], 'SepalWidth':train[:, 1], 'PetalLength':train[:, 2], 'PetalWidth':train[:, 3], 'Species':train[:, 4]})
dfTrain['Species'].replace(0, 'Iris-setosa', inplace=True)
dfTrain['Species'].replace(1, 'Iris-versicolor', inplace=True)
dfTrain['Species'].replace(2, 'Iris-virginica', inplace=True)
  # write the test data into another Pandas frame
dfTest = pd.DataFrame({'SepalLength':test[:, 0], 'SepalWidth':test[:, 1], 'PetalLength':test[:, 2], 'PetalWidth':test[:, 3], 'Species':test[:, 4]})
dfTest['Species'].replace(0, 'Iris-setosa', inplace=True)
dfTest['Species'].replace(1, 'Iris-versicolor', inplace=True)
dfTest['Species'].replace(2, 'Iris-virginica', inplace=True)
  ### 3. specify temp table names
#dataTable = 'DataMining.IrisDataset'
dtTrain = 'TRAIN02'
dtTest = "TEST02"
  ### 4. Create 2 temporary tables - you can try drop tables then re-create them every time
curs.execute("Create Table %s (%s DOUBLE, %s DOUBLE, %s DOUBLE, %s DOUBLE, %s VARCHAR(100))" % (dtTrain, dfTrain.columns[0], dfTrain.columns[1], dfTrain.columns[2], dfTrain.columns[3], dfTrain.columns[4]))
curs.execute("Create Table %s (%s DOUBLE, %s DOUBLE, %s DOUBLE, %s DOUBLE, %s VARCHAR(100))" % (dtTest, dfTest.columns[0], dfTest.columns[1], dfTest.columns[2], dfTest.columns[3], dfTest.columns[4]))
  ### 5. write Train set and Test set into the tales. You can try to delete old record then insert everytime.
curs.fast_executemany = True
curs.executemany( "INSERT INTO %s (SepalLength, SepalWidth, PetalLength, PetalWidth, Species) VALUES (?, ?, ?, ? ,?)" % dtTrain,
  list(dfTrain.itertuples(index=False, name=None)) )
curs.executemany( "INSERT INTO %s (SepalLength, SepalWidth, PetalLength, PetalWidth, Species) VALUES (?, ?, ?, ? ,?)" % dtTest,
  list(dfTest.itertuples(index=False, name=None)) )
  ### 6. CLose and clean - if connection is not needed anymore?
#curs.close()
```

#conn.close()

现在，如果切换到 IRIS 管理控制台或终端 SQL 控制台，应该看到已创建 2 个临时表：120 行的 TRAIN02 和 30 行的 TEST02。

那么本篇快速笔记到这里就结束了。

4. 注意事项

- 以上内容可能会被更改或完善。

5. 未来计划

我们将使用 IRIS 的 PyODBC、PySPark 和 Python Native API 替换第 3.3 和 3.4 节。除非有人愿意帮忙编写一篇快速笔记，我也将对此不胜感激。

#JDBC #ODBC #Python #机器学习 #InterSystems IRIS

源 URL:https://cn.community.intersystems.com/post/%E5%B0%86-python-jdbc-%E8%BF%9E%E6%8E%A5%E5%88%B0-iris-%E6%95%B0%E6%8D%AE%E5%BA%93-%E5%BF%AB%E9%80%9F%E7%AC%94%E8%AE%B0