python-joern Documentation

Release 0.2.5

Fabian Yamaguchi

June 06, 2014

Contents

Welcome to python-joern's documentation. Python-joern provides a thin python wrapper for the code analysis system joern and a library of utility traversals that can be used to quickly navigate in the code property graph. If you use joern, you will almost definitely want to install this library as well.

CHAPTER 1

Installation

1.1 Installing python-joern

python-joern can be installed using pip:

```
sudo pip2 install git+git://github.com/fabsx00/python-joern.git
```

This will (hopefully) automatically take care of installing dependencies. If dependencies are not installed correctly, you can try to manually install them as discussed in the following section.

1.2 Manually Installing Dependencies

The following steps are only required if for some reason, installation of python-joern fails due to unresolved dependencies.

Install *py2neo 1.6.1* from

https://pypi.python.org/packages/source/p/py2neo/py2neo-1.6.1.tar.gz

On Linux and BSD systems, executing the following commands will typically suffice:

```
wget https://pypi.python.org/packages/source/p/py2neo/py2neo-1.6.1.tar.gz;
tar xfz py2neo-1.6.1.tar.gz;
cd py2neo-1.6.1;
sudo python2 setup.py install;
```

Install the gremlin-plugin for neo4j from https://github.com/fabsx00/py2neo-gremlin/releases/tag/0.1:

```
wget https://github.com/fabsx00/py2neo-gremlin/archive/0.1.tar.gz
tar xfz 0.1.tar.gz
cd py2neo-gremlin-0.1
sudo python2 setup.py install
```

CHAPTER 2

Basic Usage

Python-joern currently provides a single class, JoernSteps, that allows to connect to the database server and run queries. The following is a simple sample script that employs JoernSteps to configure the database connection, connect to the server and run a Gremlin query.

```
from joern.all import JoernSteps

j = JoernSteps()

j.setGraphDbURL('http://localhost:7474/db/data/')

# j.addStepsDir('Use this to inject utility traversals')

j.connectToDatabase()

res =  j.runGremlinQuery('getFunctionsByName("main")')
# res =  j.runCypherQuery('...')

for r in res: print r
```

The sample script employs all methods offered by JoernSteps. We now discuss each of these methods in detail.

2.1 setGraphDbURL(url)

Sets the URL of the graph database server. The REST API of the Neo4J Database server is exposed on port 7474 by default. If your server runs on a different port or server, you can use setGraphDbURL to specify the alternate URL.

2.2 addStepsDir(dirname)

Add a source directory for utility traversals. By default, python-joern will inject all utility traversals contained in any of the source files in joern/joernsteps into the database before running scripts. Additional traversals specific to your application or analysis are best placed in a separate directory. python-joern can be instructed to honor this additional directory using addStepsDir.

2.3 connectToDatabase()

Connect to the database. Call this method once the connection has been configured to connect to the database server. A connection is required before queries can be executed.

2.4 runGremlinQuery(query)

Run the specified Gremlin query. The supplied query is executed and the result is returned. Depending on the query, the result may have a different data type, however, it is typically an iterable containing nodes that match the query.

2.5 runCypherQuery(query)

Run the specified Cypher query. The supplied query is executed and the result is returned. Depending on the query, the result may have a different data type, however, it is typically an iterable containing nodes that match the query.

CHAPTER 3

Chunking

Running the same traversal on a large set of start nodes often leads to unacceptable performance as all nodes and edges touched by the traversal are kept in server memory before returning results. For example, the query:

```
getAllStatements().astNodes().id
```

which retrieves all astNodes that are part of statements, can already completely exhaust memory.

If traversals are independent, the query can be chunked to gain high performance. The following example code shows how this works:

```
from joern.all import JoernSteps

j = JoernSteps()
j.connectToDatabase()

ids =  j.runGremlinQuery('getAllStatements.id')

CHUNK_SIZE = 256
for chunk in j.chunks(ids, CHUNK_SIZE):

   query = """ idListToNodes(%s).astNodes().id """ % (chunk)

   for r in j.runGremlinQuery(query): print r
```

This will execute the query in batches of 256 start nodes each.

CHAPTER 4

Utility Traversals

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*