ipython-gremlin Documentation

Release 0.0.4

David M. Brown

Mar 16, 2017

Contents

ipython-gremlin is an IPython extension module that allows the user to magically submit scripts to the TinkerPop Gremlin Server using *%gremlin* (line) and *%%gremlin* (cell) magic...

ipython-gremlin also provides basic integration with pandas and NetworkX data/graph analysis libraries to translate Gremlin traversal results to the data structures commonly used in Python scientific computing.

This work is based on Javier de la Rosa's excellent *ipython-cypher* extension.

Check out this example IPython notebook

CHAPTER 1

Releases

The latest release of `ipython-gremlin` is **1.0.0**.

CHAPTER 2

Requirements

- Python 3.5 +
- TinkerPop 3.2.4

CHAPTER 3

Dependencies

- aiogremlin 3.2.4
- ipython 5.3.0

To leverage the full power of `ipython-gremlin`, please install commonly used scientific computing libraries:

```
$ pip install pandas
$ pip install networkx
$ pip install matplotlib
```

CHAPTER 4

Installation

Install using pip:

```
$ pip install ipython-gremlin
```

CHAPTER 5

Getting Started

Load the extension:

```
%load_ext gremlin
```

Submit a script to the Gremlin Server:

```
%gremlin g.V()
```

Store query results in a variable:

```
verts = %grelmin g.V()
```

Get a pandas `pandas.DataFrame`:

```
verts = %gremlin g.V()
df = verts.get_dataframe()
```

Get a NetworkX `networkx.MultiDiGraph` from a collection of elements:

```
edges = %gremlin g.E()
graph = edges.get_graph()
```

Contribute

Contributions are welcome. If you find a bug, or have a suggestion, please open an issue on Github. If you would like to make a pull request, please make sure to add appropriate tests and run them:

```
$ ipython setup.py test
```

I am particularly interested in adding features that integrate Pandas and NetworkX. ipython-cypher has some good examples of this.

Contents:

Using ipython-gremlin

The following guide demonstrates the usage of ipython-gremlin through an example interactive live session. It assumes that an instance of Gremlin Server version 3.2.4 is running is running at *localhost:8182* using the *gremlin-server-modern.yaml* configuration file:

```
$ ./bin/gremlin-server.sh conf/gremlin-server-modern.yaml
```

Using Gremlin Magic

The first thing to do is load the extension:

```
In [1]: %load_ext gremlin
```

By default, *ipython-gremlin* will connect to *ws://localhost:8182/gremlin*. This can be changed at the beginning of the session by configuring the `GremlinMagic` object, but the easy way to do this is to simply set the current connection to the desired url:

```
In [2]: %gremlin.connection.set_current ws://localhost:8182/gremlin
Alias-- localhost --created for database at ws://localhost:8182/gremlin
```

Notice the output stating that an alias has been created for this connection. Connection aliases will be further discussed later in this document.

Line Magic

The first two statements shown here use what is called "line magic", which is denoted by the % symbol followed by a command name as well as any number of arguments.

Line magic is generally the best way to use *ipython-gremlin*, as it allows you to bind the results of a magic traversal to a variable, or integrate with Python code in other ways:

```
In [3]: verts = %gremlin g.V()

In [4]: verts
Out[4]: [v[1], v[2], v[3], v[4], v[5], v[6]]

In [5]: for v in verts:
   ...:     vid = v.id
   ...:     props = %gremlin g.V(vid).properties()
   ...:     print(props)
   [vp[name->marko], vp[age->29]]
   [vp[name->vadas], vp[age->27]]
   [vp[name->lop], vp[lang->java]]
   [vp[name->josh], vp[age->32]]
   [vp[name->ripple], vp[lang->java]]
   [vp[name->peter], vp[age->35]]
```

Notice that *ipython-gremlin* can detect bindings stored in the user namespace:

```
In [6]: vid = 1

In [7]: %gremlin g.V(vid)
Out[7]: [v[1]]
```

Cell Magic

ipython-gremlin also supports "cell magic", denoted by *%%*. This allows multi-line traversals to be submitted to the server. The downside to cell magic is that the results cannot be bound to a variable in the IPython namespace. Regardless, it can be useful for things like creating elements or executing management commands:

```
In [8]: %%gremlin
    ...: graph.addVertex('dog').property('name', 'ewok')
    ...: // ... do stuff ...
Out[8]: [v[13]]
```

Integration with `pandas` and `networkx`

When possible, results generated by traversal can be converted into `pandas.DataFrame` and `networkx.MultiDiGraph` objects. This helps facilitate plotting, transformation, and analysis.

`pandas`

Gremlin traversals can produce complex results and nested datastructures that are difficult to map to a two dimensional table. Despite this, many traversals return relatively simple results that are easy to translate to a tabular structure. For example:

```
In [9]: values = %gremlin g.V().valueMap(true)

In [10]: values.dataframe
Out[10]:
    age  id     label    lang      name
0  [29]   1    person     NaN   [marko]
1  [27]   2    person     NaN   [vadas]
2   NaN   3  software  [java]     [lop]
3  [32]   4    person     NaN    [josh]
4   NaN   5  software  [java]  [ripple]
5  [35]   6    person     NaN   [peter]

In [11]: edges = %gremlin g.E()

In [12]: edges.dataframe
Out[12]:
   id  inV    label  outV
0   7    2    knows     1
1   8    4    knows     1
2   9    3  created     1
3  10    5  created     4
4  11    3  created     4
5  12    3  created     6
```

`NetworkX`

When a traversal returns a collection of elements, they can be used to produce a NetworkX graph. Typically a container of edges or paths is the best way to utilize this functionality:

```
In [13]: edges.graph
Out[13]: <networkx.classes.multidigraph.MultiDiGraph at 0x7f806b8ee828>

In [14]: paths = %gremlin g.V().outE().inV().outE().inV().path()

In [15]: paths.graph
Out[15]: <networkx.classes.multidigraph.MultiDiGraph at 0x7f806b8a2208>
```

Managing Connections

Typically, `Gremlin-Python` manages connections using a `Cluster` object that maintains connection pools to a series of hosts specified using configuration parameters. `ipython-gremlin` has a simpler use case than many apps using `Gremlin-Python` or `aiogremlin`, and therefore doesn't use a `Cluster` under the hood. Instead, it manages a registry of simple connections to individual hosts. To add a new host, simply use the line magic shown above:

```
In [16]: %gremlin.connection.set_current ws://davebshow@myhost:8182/gremlin
Alias-- davebshow@myhost --created for database at ws://localhost:8182/gremlin
```

If necessary, this creates a new connection to the specified host, and sets this connection as the current connection to be used by `ipython-gremlin`. An alias for this connection is created automagically, which can be used to refer to this connection. For example, to switch back to the default *localhost* connection:

```
In [17]: %gremlin.connection.set_current localhost
Now using connection at ws://localhost:8182/gremlin
```

Connection Aliases

`ipython-gremlin` allows the creation of custom aliases. If it is the first time a connection is created (using *%gremlin.connection.set_current*), the alias can be set during creation:

```
In [18]: %gremlin.connection.set_current ws://localhost:8182/gremlin as TestDB
Alias-- TestDB --created for database at ws://localhost:8182/gremlin
Now using connection at ws://localhost:8182/gremlin
```

For an existing connection, an alias can be set using the *$gremlin.connection.set_alias* line magic:

```
In [19]: %gremlin.connection.set_alias ws://localhost:8182/gremlin as TestDB2
Alias-- TestDB2 --created for database at ws://localhost:8182/gremlin
```

Getting the Current Connection

The current connection can be accessed using the line magic *%gremlin.connection.current*:

```
In [20]: %gremlin.connection.current
Out[20]: Connection at ws://localhost:8182/gremlin aliased as localhost
```

Cell Magic and Connection Management

Connections can also be managed using cell magic (*%%gremlin*). Connections are created, aliased, and set as current simply by passing a connection string on the first line of the cell:

```
In [21]: %%gremlin ws://localhost:8182/gremlin as MyHost
...: g.V()
...:
Out[21]: [v[1], v[2], v[3], v[4], v[5], v[6]]

In [22]: %%gremlin MyHost
   ...: g.V()
   ...:
Out[22]: [v[1], v[2], v[3], v[4], v[5], v[6]]
```

Closing Connections

`ipython-gremlin` tries to clean up conextions using the `atexit` module, but it is a good idea to explicitly close connections at the end of the interactive session. This is done using line magic:

```
In [23]: %gremlin.connection.close
```

Or, more simply:

    In [24]: %gremlin.close

To close an individual connection, pass a valid connection string as an argument to *%gremlin.connection.close*:

```
In [25]: %gremlin.connection.close localhost
```

Configuration

GremlinMagic also supports the `Configurable` interface. It provides a range of connection configuration options. These options are displayed using the *%config* cell magic:

```
In [26]: %config GremlinMagic
GremlinMagic options
--------------------
GremlinMagic.aliases=<Dict>
    Current: {'g': 'g'}
    Aliases for underlying graph
GremlinMagic.password=<Unicode>
    Current: ''
    Password used in SASL authentication
GremlinMagic.response_timeout=<Float>
    Current: None
    Timeout for server response
GremlinMagic.ssl_context=<Instance>
    Current: None
    `ssl.SSLContext` object for SSL
GremlinMagic.uri=<Unicode>
    Current: 'ws://localhost:8182/gremlin'
    Default database URI if none is defined inline
GremlinMagic.username=<Unicode>
    Current: ''
    Username used in SASL authentication
```

Changing this configuration is easy:

```
In [27]: %config GremlinMagic.username = 'davebshow'

In [28]: %config GremlinMagic.username
Out[28]: 'davebshow'
```

NOTE Configuration changes do not affect connections that have already been established. Please update configuration before performing other *%gremlin* line magic.

That's it! I hope you enjoy using `ipython-gremlin`!

ipython-gremlin API

ipython-gremlin package

Submodules

gremlin.config module

Default configuration for GremlinMagic

class `gremlin.config.`**`DefaultConfig`**(*aliases*, *response_timeout*, *username*, *password*, *uri*, *ssl_context*)

Bases: `tuple`

`aliases`
Alias for field number 0

`password`
Alias for field number 3

`response_timeout`
Alias for field number 1

`ssl_context`
Alias for field number 5

`uri`
Alias for field number 4

`username`
Alias for field number 2

gremlin.magic module

IPython Gremlin custom magic

class `gremlin.magic.`**`GremlinMagic`**(*shell=None*, ***kwargs*)

Bases: `IPython.core.magic.Magics`

First of all, keep him out of the light, he hates bright light, especially sunlight, it'll kill him. Second, don't give him any water, not even to drink. But the most important rule, the rule you can never forget, no matter how much he cries, no matter how much he begs, never feed him after midnight.

`aliases`
An instance of a Python dict.

close (*line*)
Explicity close underlying DB connections

close_connection (*line*)

get_current_connection (*line*)
Get the currently used connection object

gremlin (*line*, *cell=None*, *local_ns={}*)
I make the illogical logical

magics = {'line': {'gremlin.close': 'close', 'gremlin.connection.close': 'close_connection', 'gremlin': 'gremlin', 'gremlin.

password
A trait for unicode strings.

registered = True

response_timeout
A float trait.

set_connection_alias (*line*)
Set alias for specified connection

set_current_connection (*line*)
Set specified connection as current

ssl_context
A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

uri
A trait for unicode strings.

username
A trait for unicode strings.

gremlin.magic.**close**()
uh oh

gremlin.magic.**load_ipython_extension**(*ip*)
Load the extension in IPython.

gremlin.registry module

Simple interface for keeping track of DB connections and aliases

class gremlin.registry.**ConnectionRegistry**
Bases: object

Issue and keep track of database connections.

classmethod close (*connection_str=None*)
Close DB connections

connections = {}

current = None

classmethod get (*descriptors*, *config=None*)
Get a connection from the registry based on the descriptor

classmethod **set_connection_alias**(*descriptors*, *config*)

classmethod **set_current_connection**(*descriptors*, *config*)

class gremlin.registry.**RegistryConnection**(*conn*, *alias*)
Bases: object

Wrapper for aiogremlin.driver.connection.Connection

alias

close()
Close underlying connections

loop

uri

write(*message*)
Write message to Gremlin Server

gremlin.utils module

Utility functions to support GremlinMagic operations

gremlin.utils.**parse**(*connection_str*)
Parse connection string passed by user

gremlin.utils.**submit**(*gremlin*, *user_ns*, *aliases*, *conn*)
Submit a script to the Gremlin Server using the IPython namespace using the IPython namespace to pass bindings using Magics configuration and a connection registered with *ConnectionRegistry*

Module contents

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index