gremlinclient Documentation

Release 0.1.0

David M. Brown

June 15, 2016

Contents

gremlinclient is an asynchronous multi-client Python driver for the TinkerPop 3 Gremlin Server. By default, it uses the Tornado websocket client implementation to communicate with the server, but it also supports aiohttp for a pure Asyncio implementation–support for Pulsar and requests-futures coming soon.

CHAPTER 1

Releases

The latest release of *gremlinclient* is **0.2.6**.

CHAPTER 2

Requirements

`gremlinclient` with Tornado requires Python 2.7+. That said, there are a variety of client/library combinations that work with different versions of Python.

Tornado

- Python 2.7+

Tornado w/Asyncio

- Python 3.3+

Tornado w/Trollius

- Python 2.7

aiohttp

- Python 3.4+

CHAPTER 3

Installation

Install using pip:

```
$ pip install gremlinclient
```

CHAPTER 4

Getting Started

Submit a script to the Gremlin Server with Python 2.7 or 3.3+ using Tornado:

```
>>> from tornado import gen
>>> from tornado.ioloop import IOLoop
>>> from gremlinclient.tornado_client import submit

>>> loop = IOLoop.current()

>>> @gen.coroutine
... def go():
...     resp = yield submit("ws://localhost:8182/", "1 + 1")
...     while True:
...         msg = yield resp.read()
...         if msg is None:
...             break
...         print(msg)
>>> loop.run_sync(go)

Message(status_code=200, data=[2], message=u'', metadata={})
```

Contents:

4.1 Using `gremlinclient`

Before you get started, make sure you have the Gremlin Server up and running. All of the following examples use the Tornado client with PEP 492 Python 3.5 async/await syntax, but they can all be adjusted as shown in the *Using aiohttp* and *Tornado Asyncio Integration* sections.

4.1.1 Simple `API`

Submit a script with *`submit`*:

```
>>> async def do_submit():
...     resp = await submit(
...         "ws://localhost:8182/", "1 + 1")
...     while True:
...         msg = await resp.read()
...         if msg is None:
...             break  # connection closes automatically
...         print(msg)
```

Get a database connection with *create_connection*:

```
>>> async def get_conn():
...     conn = await create_connection("ws://localhost:8182/")
...     resp = conn.send(
...         "ws://localhost:8182/", "1 + 1")
...     while True:
...         msg = await resp.read()
...         if msg is None:
...             break
...     conn.close()  # make sure conn is closed when done
```

4.1.2 The GraphDatabase object

Get a database connection from *GraphDatabase*:

```
>>> async def get_conn():
...     graph = GraphDatabase("ws://localhost:8182/")
...     conn = await graph.connect()
...     ...
...     conn.close()
```

Get a database session connection from *GraphDatabase*:

```
>>> async def get_conn():
...     graph = GraphDatabase("ws://localhost:8182/")
...     sess = await graph.session()  # session inherits from Connection
...     ...
...     sess.close()
```

4.1.3 The Pool object

Reuse websocket connections with *Pool*:

```
>>> async def get_conn():
...     pool = Pool("ws://localhost:8182/")
...     conn = await pool.acquire()
...     ...
...     pool.release(conn)
...     pool.close()  # Close all released conns
```

Automatically release connections to *Pool* after read:

```
>>> async def get_conn():
...     pool = Pool("ws://localhost:8182/", force_release=True)
...     conn = await pool.acquire()
...     resp = conn.send("1 + 1")
...     while True:
...         msg = await resp.read()
...         if msg is None:
...             break  # conn is automatically released to pool.
...      pool.close()
```

For more info, see the *Tornado Client docs*

4.1.4 The `RemoteConnection` object

The remote connection object provides a synchronous interface designed to be used with the official TinkerPop Gremlin-Python Gremlin Language Variant (GLV):

```
>>> from gremlin_python import PythonGraphTraversalSource, GroovyTranslator  # imports may change afi
>>> from gremlinclient.tornado_client import RemoteConnection
>>> remote_conn = RemoteConnection("ws://localhost:8182/")
>>> translator = GroovyTranslator("g")
>>> g = PythonGraphTraversalSource(translator,
...                                remote_connection=remote_conn)
```

This allows you to write Gremlin traversals using pure Python!:

```
>>> g.addV('person').property('name','stephen').next()
>>> g.V().toList()
```

Remember to call `next()` or `toList()` to submit the traversal to the server.

For more info see `aiohttp_client.RemoteConnection` and `tornado_client.RemoteConnection`

4.2 Using aiohttp

`aiohttp` is not installed with *`gremlinclient`* by default. Use pip to install it:

```
$ pip install aiohttp
```

If you aren't using tornado, go ahead and uninstall it:

```
$ pip uninstall tornado
```

Using the `aiohttp client` is easy, it provides the exact same objects and API as the `Tornado client` with one small exception: the `close()` methods return `asyncio.Future` that must be yielded from or awaited. For example:

```
>>> from gremlinclient.aiohttp_client import GraphDatabase
>>> async def get_conn():
...     graph = GraphDatabase("ws://localhost:8182/")
...     conn = await graph.connect()
...     ...
...     await conn.close()  # await close
```

Or if you are using a *`Pool`*:

```
>>> from gremlinclient.aiohttp_client import Pool
>>> async def use_pool():
...     pool = Pool("ws://localhost:8182/")
...     conn = yield from pool.acquire()
...     ...
...     await pool.release(conn)
...     await pool.close()
```

For more info, see the *aiohttp client docs*

4.3 Tornado/Asyncio Integration

If you want run the Tornado client on the `Asyncio event loop` simply follow the patterns shown in the Tornado docs. Also, make sure to pass an `asyncio.Future` class to the *future_class* kwarg of the function or object you are using.

Submit a script to the Gremlin Server with Python 3.3+ and Asyncio:

```
>>> import asyncio
>>> from tornado.platform.asyncio import AsyncIOMainLoop
>>> from gremlinclient import submit

>>> AsyncIOMainLoop().install() # Use the asyncio event loop
>>> loop = asyncio.get_event_loop()

>>> @asyncio.coroutine
... def go():
...     resp = yield from submit(
...         "ws://localhost:8182/", "1 + 1",
...         future_class=asyncio.Future)
...     while True:
...         msg = yield from resp.read()
...         if msg is None:
...             break
...         print(msg)
>>> loop.run_until_complete(go())

Message(status_code=200, data=[2], message=u'', metadata={})
```

Submit a script with Python 3.5 using PEP492 async/await syntax (Asyncio):

```
>>> import asyncio
>>> from tornado.platform.asyncio import AsyncIOMainLoop
>>> from gremlinclient import submit

>>> AsyncIOMainLoop().install() # Use the asyncio event loop
>>> loop = asyncio.get_event_loop()

>>> async def go():
...     resp = await submit(
...         "ws://localhost:8182/", "1 + 1",
...         future_class=asyncio.Future)
...     while True:
...         msg = await resp.read()
...         if msg is None:
...             break
...         print(msg)
>>> loop.run_until_complete(go())

Message(status_code=200, data=[2], message=u'', metadata={})
```

You can do the same with Python 2.7 using Trollius, just pass `trollius.Future` class to the function:

```
>>> import trollius
>>> from tornado.platform.asyncio import AsyncIOMainLoop
>>> from gremlinclient import submit

>>> AsyncIOMainLoop().install() # Use the asyncio event loop
>>> loop = trollius.get_event_loop()
```

```
>>> @trollius.coroutine
... def go():
...     fut = submit(
...         "ws://localhost:8182/", "1 + 1",
...         future_class=trollius.Future)
...     resp = yield trollius.From(fut)
...     while True:
...         fut_msg = resp.read()
...         msg = yield trollius.From(fut_msg)
...         if msg is None:
...             break
...         print(msg)
>>> loop.run_until_complete(go())

Message(status_code=200, data=[2], message=u'', metadata={})
```

4.4 SSL with `gremlinclient`

Setting up SSL with *gremlinclient* is straightforward, but different depending on which client you choose. The following demonstrates using SSL with both the *aiohttp_client* and *tornado_client* modules.

SSL certs and server config are generally **up to the user**, but for **testing** you can get going with OpenSSL self-signed certificates. Something like:

```
$ openssl req -nodes -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days XXX
```

Then add something like this to the *conf/gremlin-server.yaml* file:

```
ssl: {
  enabled: true,
  keyCertChainFile: /path/to/cert.pem,
  keyFile: /path/to/key.pem}
```

Okay, both aiohttp and Tornado use Python's `ssl` module to create an `ssl.SSLContext`:

```
>>> import ssl
>>> sslcontext = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
>>> sslcontext.load_cert_chain(
...     '/path/to/cert.pem', keyfile='/path/to/key.pem')
```

4.4.1 `aiohttp_client`

To set up SSL with *aiohttp_client*, use the `aiohttp.TCPConnector` class:

```
>>> connector = aiohttp.TCPConnector(ssl_context=sslcontext)
```

Then pass this object as a kwarg to *submit*, *create_connection*, *GraphDatabase*, or *Pool*:

```
>>> stream = yield from submit(
...     "wss://localhost:8182/", "1 + 1", connector=connector)
```

Don't forget to use the "wss" protocol.

4.4.2 tornado_client

To set up SSL with *tornado_client*, we create a `request_factory()` that creates `HTTPRequest` objects with the `ssl.SSLContext` as a frozen kwarg and use this as our `connector`:

```
>>> from functools import partial
>>> request_factory = partial(
...     httpclient.HTTPRequest, ssl_options=sslcontext)
```

Then pass this object as a kwarg to *submit*, *create_connection*, *GraphDatabase*, or *Pool*:

```
>>> stream = yield from submit(
...     "wss://localhost:8182/", "1 + 1", connector=request_factory)
```

Again, don't forget to use the "wss" protocol.

4.5 GremlinClient API

4.5.1 tornado_client package

Module contents

tornado_client.client module

class gremlinclient.tornado_client.client.**GraphDatabase**(*url*, *timeout=None*, *username=''*, *password=''*, *loop=None*, *future_class=None*, *connector=None*)

Bases: *gremlinclient.graph.GraphDatabase*

This class generates connections to the Gremlin Server.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **loop** – If param is `None`, *tornado.ioloop.IOLoop.current* is used for getting default event loop (optional)
- **future_class** (*class*) – type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`
- **connector** (*func*) – a factory for generating `tornado.HTTPRequest` objects. used with ssl

class gremlinclient.tornado_client.client.**Pool**(*url*, *graph=None*, *timeout=None*, *username=''*, *password=''*, *maxsize=256*, *loop=None*, *force_release=False*, *future_class=None*, *connector=None*)

Bases: *gremlinclient.pool.Pool*

Pool of *gremlinclient.connection.Connection* objects.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **graph** (`gremlinclient.tornado_client.client.GraphDatabase`) – The graph instance used to create connections
- **maxsize** (*int*) – Maximum number of connections.
- **loop** – event loop
- **future_class** (*class*) – type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`
- **connector** (*func*) – a factory for generating `tornado.HTTPRequest` objects. used with ssl

class `gremlinclient.tornado_client.client.`**Response**(*conn*, *future_class*, *loop=None*)

Bases: `gremlinclient.response.Response`

Wrapper for Tornado websocket client connection.

Parameters **conn** (*tornado.websocket.WebSocketClientConnection*) – The websocket connection

close()

Close underlying client connection.

Returns type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`

closed

Returns bool True is conn is closed.

conn

Returns Underlying connection.

receive(*callback=None*)

Read a message off the websocket. :param callback: To be called on message read.

Returns :py:class:type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`

send(*msg*, *binary=True*)

Send a message

Parameters

- **msg** – The message to be sent.
- **binary** (*bool*) – Whether or not the message is encoded as bytes.

gremlinclient.tornado_client.client.**create_connection**(*url*, *timeout=None*, *username=''*, *password=''*, *loop=None*, *session=None*, *force_close=False*, *future_class=None*, *connector=None*)

Get a database connection from the Gremlin Server.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **loop** – If param is `None`, `tornado.ioloop.IOLoop.current()` is used for getting default event loop (optional)
- **force_close** (*bool*) – force connection to close after read.
- **future_class** (*class*) – type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`
- **session** (*str*) – Session id (optional). Typically a uuid
- **connector** (*func*) – a factory for generating `tornado.HTTPRequest` objects. used with ssl

Returns *gremlinclient.connection.Connection* object:

gremlinclient.tornado_client.client.**submit**(*url*, *gremlin*, *bindings=None*, *lang='gremlin-groovy'*, *aliases=None*, *op='eval'*, *processor=''*, *timeout=None*, *session=None*, *loop=None*, *username=''*, *password=''*, *future_class=None*, *connector=None*)

Submit a script to the Gremlin Server.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **gremlin** (*str*) – Gremlin script to submit to server.
- **bindings** (*dict*) – A mapping of bindings for Gremlin script.
- **lang** (*str*) – Language of scripts submitted to the server. "gremlin-groovy" by default
- **aliases** (*dict*) – Rebind `Graph` and `TraversalSource` objects to different variable names in the current request
- **op** (*str*) – Gremlin Server op argument. "eval" by default.
- **processor** (*str*) – Gremlin Server processor argument. "" by default.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **session** (*str*) – Session id (optional). Typically a uuid
- **loop** – If param is `None`, `tornado.ioloop.IOLoop.current()` is used for getting default event loop (optional)

- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **future_class** (*class*) – type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`
- **connector** (*func*) – a factory for generating `tornado.HTTPRequest` objects. used with ssl

Returns *gremlinclient.connection.Stream* object:

tornado_client.remote_connection module

4.5.2 aiohttp_client package

Module contents

aiohttp_client.client module

class `gremlinclient.aiohttp_client.client.`**GraphDatabase**(*url*, *timeout=None*, *username=''*, *password=''*, *loop=None*, *future_class=None*, *connector=None*)

Bases: *gremlinclient.graph.GraphDatabase*

This class generates connections to the Gremlin Server.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **loop** – If param is `None`, *asyncio.get_event_loop* is used for getting default event loop (optional)
- **future_class** (*class*) – type of Future - `asyncio.Future`
- **connector** (*aiohttp.TCPConnector*) – `aiohttp.TCPConnector` object. used with ssl

class `gremlinclient.aiohttp_client.client.`**Pool**(*url*, *timeout=None*, *username=''*, *password=''*, *maxsize=256*, *loop=None*, *future_class=None*, *force_release=False*, *connector=None*)

Bases: *gremlinclient.pool.Pool*

Pool of *gremlinclient.connection.Connection* objects.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout

- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **graph** (gremlinclient.aiohttp_client.client.GraphDatabase) – The graph instance used to create connections
- **maxsize** (*int*) – Maximum number of connections.
- **loop** – event loop
- **future_class** (*class*) – type of Future - asyncio.Future by default
- **connector** (*aiohttp.TCPConnector*) – aiohttp.TCPConnector object. used with ssl

close()
Close pool. :returns: asyncio.Future

release(*conn*)
Release a connection back to the pool.

Parameters **gremlinclient.connection.Connection** – The connection to be released

Returns asyncio.Future

class gremlinclient.aiohttp_client.client.**Response**(*conn*, *future_class*, *loop=None*)
Bases: gremlinclient.response.Response

Wrapper for aiohttp websocket client connection.

Parameters **conn** (*aiohttp.ClientWebSocketResponse*) – The websocket connection

close()
Close underlying client connection :returns: asyncio.Future

closed

Returns bool. True if conn is closed

receive(*callback=None*)
Read a message off the websocket. :param callback: To be called on message read.

Returns asyncio.Future

send(*msg*, *binary=True*)
Send a message

Parameters

- **msg** – The message to be sent.
- **binary** (*bool*) – Whether or not the message is encoded as bytes.

gremlinclient.aiohttp_client.client.**create_connection**(*url*, *timeout=None*, *username=''*, *password=''*, *loop=None*, *session=None*, *force_close=False*, *future_class=None*, *connector=None*)
Get a database connection from the Gremlin Server.

Parameters

- **url** (*str*) – url for Gremlin Server.

- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **loop** – If param is `None`, `asyncio.get_event_loop()` is used for getting default event loop (optional)
- **force_close** (*bool*) – force connection to close after read.
- **future_class** (*class*) – type of Future - `asyncio.Future` by default
- **session** (*str*) – Session id (optional). Typically a uuid
- **connector** (*aiohttp.TCPConnector*) – `aiohttp.TCPConnector` object. used with ssl

Returns *gremlinclient.connection.Connection* object:

gremlinclient.aiohttp_client.client.**submit**(*url*, *gremlin*, *bindings=None*, *lang='gremlin-groovy'*, *aliases=None*, *op='eval'*, *processor=''*, *timeout=None*, *session=None*, *loop=None*, *username=''*, *password=''*, *future_class=None*, *connector=None*)

Submit a script to the Gremlin Server.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **gremlin** (*str*) – Gremlin script to submit to server.
- **bindings** (*dict*) – A mapping of bindings for Gremlin script.
- **lang** (*str*) – Language of scripts submitted to the server. "gremlin-groovy" by default
- **aliases** (*dict*) – Rebind `Graph` and `TraversalSource` objects to different variable names in the current request
- **op** (*str*) – Gremlin Server op argument. "eval" by default.
- **processor** (*str*) – Gremlin Server processor argument. "" by default.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **session** (*str*) – Session id (optional). Typically a uuid
- **loop** – If param is `None`, `asyncio.get_event_loop()` is used for getting default event loop (optional)
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **future_class** (*class*) – type of Future - `asyncio.Future` by default
- **connector** (*aiohttp.TCPConnector*) – `aiohttp.TCPConnector` object. used with ssl

Returns *gremlinclient.connection.Stream* object:

aiohttp_client.remote_connection module

4.5.3 gremlinclient package

gremlinclient.api module

gremlinclient.connection module

class gremlinclient.connection.**Connection**(*conn*, *future_class*, *timeout=None*, *username=''*, *password=''*, *loop=None*, *force_close=False*, *pool=None*, *force_release=False*, *session=None*)

Bases: object

This class encapsulates a connection to the Gremlin Server. Don't directly create *Connection* instances. Use *gremlinclient.graph.GraphDatabase.connect()* or gremlinclient.api.create_connection() instead.

Parameters

- **conn** (*tornado.websocket.WebSocketClientConnection*) – client websocket connection.
- **timeout** (*float*) – timeout for establishing connection (optional). Values 0 or None mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **loop** – If param is None, *tornado.ioloop.IOLoop.current* is used for getting default event loop (optional)
- **force_close** (*bool*) – force connection to close after read.
- **future_class** (*class*) – type of Future - asyncio.Future, trollius.Future, or tornado.concurrent.Future
- **pool** (gremlinclient.pool.Pool) – Connection pool. None by default
- **force_release** (*bool*) – If possible, force release to pool after read.
- **session** (*str*) – Session id (optional). Typically a uuid

close()
Close the underlying websocket connection, detach from pool, and set to close.

closed
Readonly property. Return True if client has been closed or client connection has been closed :returns: bool

conn
Read only property for websocket connection. :returns: tornado.websocket.WebSocketClientConnection

release()
Release connection to associated pool.

send(*gremlin*, *bindings=None*, *lang='gremlin-groovy'*, *aliases=None*, *op='eval'*, *processor=''*, *session=None*, *timeout=None*, *handler=None*, *request_id=None*)
Send a script to the Gremlin Server.

Parameters

- **gremlin** (*str*) – Gremlin script to submit to server.

- **bindings** (*dict*) – A mapping of bindings for Gremlin script.
- **lang** (*str*) – Language of scripts submitted to the server. "gremlin-groovy" by default
- **aliases** (*dict*) – Rebind `Graph` and `TraversalSource` objects to different variable names in the current request
- **op** (*str*) – Gremlin Server op argument. "eval" by default.
- **processor** (*str*) – Gremlin Server processor argument. "" by default.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **session** (*str*) – Session id (optional). Typically a uuid
- **loop** – If param is `None`, *tornado.ioloop.IOLoop.current* is used for getting default event loop (optional)

Returns *gremlinclient.connection.Stream* object

class gremlinclient.connection.**Message**(*status_code*, *data*, *message*, *metadata*)
Bases: `tuple`

data
Alias for field number 1

message
Alias for field number 2

metadata
Alias for field number 3

status_code
Alias for field number 0

class gremlinclient.connection.**Session**(**args*, ***kwargs*)
Bases: *gremlinclient.connection.Connection*

Child of *gremlinclient.connection.Connection* object that is bound to a session that maintains state across messages with the server. Don't directly create Connection instances. Use *gremlinclient.graph.GraphDatabase.session()* instead.

Parameters

- **conn** (*tornado.websocket.WebSocketClientConnection*) – client websocket connection.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **loop** – If param is `None`, *tornado.ioloop.IOLoop.current* is used for getting default event loop (optional)
- **force_close** (*bool*) – force connection to close after read.
- **future_class** (*class*) – type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`
- **pool** (`gremlinclient.pool.Pool`) – Connection pool. None by default
- **force_release** (*bool*) – If possible, force release to pool after read.

- **session** (*str*) – Session id (optional). Typically a uuid

send(*gremlin*, *bindings=None*, *lang='gremlin-groovy'*, *aliases=None*, *op='eval'*, *timeout=None*, *handler=None*)
send a script to the Gremlin Server using sessions.

Parameters

- **gremlin** (*str*) – Gremlin script to submit to server.
- **bindings** (*dict*) – A mapping of bindings for Gremlin script.
- **lang** (*str*) – Language of scripts submitted to the server. "gremlin-groovy" by default
- **aliases** (*dict*) – Rebind `Graph` and `TraversalSource` objects to different variable names in the current request
- **op** (*str*) – Gremlin Server op argument. "eval" by default.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **loop** – If param is `None`, *tornado.ioloop.IOLoop.current* is used for getting default event loop (optional)

Returns *gremlinclient.connection.Stream* object

class gremlinclient.connection.**Stream**(*conn*, *session*, *processor*, *handler*, *loop*, *username*, *password*, *force_close*, *force_release*, *future_class*)

Bases: object

This object provides an interface for reading the response sent by the Gremlin Server over the websocket connection. Don't directly create stream instances, they should by returned by *gremlinclient.connection.Connection.send()* or *gremlinclient.connection.Session.send()*

Parameters

- **conn** (gremlinclient.connection.Connection) – client websocket connection.
- **session** (*str*) – Session id. Typically a uuid
- **processor** (*str*) – Gremlin Server processor argument. "" by default.
- **loop** – If param is `None`, *tornado.ioloop.IOLoop.current* is used for getting default event loop (optional)
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **force_close** (*bool*) – force connection to close after read.
- **force_release** (*bool*) – If possible, force release to pool after read.
- **future_class** (*class*) – type of Future - asyncio.Future, trollius.Future, or tornado.concurrent.Future

add_handler(*handler*)

read()
Read a message from the response stream.

Returns Future - asyncio.Future, trollius.Future, or tornado.concurrent.Future

gremlinclient.graph module

class gremlinclient.graph.**GraphDatabase**(*url*, *timeout=None*, *username=''*, *password=''*, *loop=None*, *validate_cert=False*, *future_class=None*, *session_class=<class 'gremlinclient.connection.Session'>*)

Bases: object

This class generates connections to the Gremlin Server.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **timeout** (*float*) – timeout for establishing connection (optional). Values 0 or None mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **loop** – If param is None, *tornado.ioloop.IOLoop.current* is used for getting default event loop (optional)
- **validate_cert** (*bool*) – validate ssl certificate. False by default
- **future_class** (*class*) – type of Future - asyncio.Future, trollius.Future, or tornado.concurrent.Future

connect(*session=None*, *force_close=False*, *force_release=False*, *pool=None*)

Get a connection to the graph database.

Parameters

- **session** (*str*) – Session id (optional). Typically a uuid
- **force_close** (*bool*) – force connection to close after read.
- **force_release** (*bool*) – If possible, force release to pool after read.
- **pool** (gremlinclient.pool.Pool) – Associated connection pool.

Returns *gremlinclient.connection.Connection*

future_class

session(*connector=None*, *session=None*, *force_close=False*, *force_release=False*, *pool=None*)

Get a session connection to the graph database.

Parameters

- **session** (*str*) – Session id (optional). Typically a uuid
- **force_close** (*bool*) – force connection to close after read.
- **force_release** (*bool*) – If possible, force release to pool after read.
- **pool** (gremlinclient.pool.Pool) – Associated connection pool.

Returns *gremlinclient.connection.Session*

gremlinclient.pool module

class gremlinclient.pool.**Pool**(*graph*, *maxsize=256*, *loop=None*, *force_release=False*, *future_class=None*)

Bases: object

Pool of *gremlinclient.connection.Connection* objects.

Parameters

- **url** (*str*) – url for Gremlin Server.
- **timeout** (*float*) – timeout for establishing connection (optional). Values `0` or `None` mean no timeout
- **username** (*str*) – Username for SASL auth
- **password** (*str*) – Password for SASL auth
- **graph** (`gremlinclient.graph.GraphDatabase`) – The graph instances used to create connections
- **maxsize** (*int*) – Maximum number of connections.
- **loop** – event loop
- **validate_cert** (*bool*) – validate ssl certificate. False by default
- **future_class** (*class*) – type of Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`

acquire()
Acquire a connection from the Pool

Returns Future - `asyncio.Future`, `trollius.Future`, or `tornado.concurrent.Future`

close()
Close pool

closed
Check if pool has been closed

Returns bool

freesize
Number of free connections

Returns int

future_class

Returns `type` Concrete class of the future instances created by this pool, for example :py:class: *asyncio.Future*

graph
Associated graph instance used for creating connections

Returns *gremlinclient.graph.GraphDatabase*

maxsize
Maximum number of connections

Returns in

pool
Object that stores unused connections

Returns `collections.deque`

release(*conn*)
Release a connection back to the pool.

Parameters `gremlinclient.connection.Connection` – The connection to be released

`size`

Total number of connections

Returns int

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

g

S