aiogremlin Documentation

Release 0.1.0

David M. Brown

Jan 23, 2018

Contents

aiogremlin is an asynchronous DSL based on the official *Gremlin-Python* GLV designed for integration with event loop based asynchronous Python networking libraries, including asyncio, aiohttp, and tornado. It uses the *async/await* syntax introduced in PEP 492, and is therefore Python 3.5+ only.

aiogremlin tries to follow *Gremlin-Python* as closely as possible both in terms of API and implementation. It is released according to the TinkerPop release schedule.

Note that this *NOT* an official Apache project component, it is a *THIRD PARTY PACKAGE!*

CHAPTER 1

Releases

The latest release of *aiogremlin* is **3.3.1**.

CHAPTER 2

Requirements

- Python 3.5 +
- TinkerPop 3.2.6 +

CHAPTER 3

Dependencies

- aiohttp 2.2.5 +
- gremlinpython
- PyYAML 3.12 +

CHAPTER 4

Installation

Install using pip:

```
$ pip install aiogremlin==<tinkerpop_version>
```

For this version, a separate install of gremlinpython is required:

```
$ pip install gremlinpython==<tinkerpop_version> --no-deps
```

CHAPTER 5

Getting Started

aiogremlin has a simple API that is quite easy to use. However, as it relies heavily on asyncio and aiohttp, it is helpful to be familiar with the basics of these modules.

aiogremlin is *very* similar to Gremlin-Python, except it is all async, all the time.

5.1 Minimal Example

Submit a script to the Gremlin Server:

```
>>> import asyncio
>>> from aiogremlin import DriverRemoteConnection, Graph

>>> loop = asyncio.get_event_loop()

>>> async def go(loop):
...     remote_connection = await DriverRemoteConnection.open(
...         'ws://localhost:8182/gremlin', 'g')
...     g = Graph().traversal().withRemote(remote_connection)
...     vertices = await g.V().toList()
...     return vertices

>>> results = loop.run_until_complete(go(loop))
>>> results
# [v[1], v[2], v[3], v[4], v[5], v[6]]
```

The above example demonstrates how *aiogremlin* uses the event loop to drive communication with the Gremlin Server, but the **rest of examples are written as if they were run in a Python interpreter**. In reality, **this isn't possible**, so remember, code *must* be wrapped in a coroutine and run with the event loop.

5.2 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:

```
$ python setup.py test
```

In the future there will be CI and more info on contributing.

Contents:

5.2.1 Using `aiogremlin`

Before you get started, make sure you have the Gremlin Server up and running. All of the following example assume a running Gremlin Server version 3.2.4 at *ws://localhost:8182/gremlin* using the *conf/gremlin-server-modern-py.yaml* configuration:

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

5.2.1.1 Using the Gremlin Language Variant

aiogremlin is used almost exactly like the official Gremlin-Python, except that all operations are asynchronous. Thus when coding with *aiogremlin* coroutines and the *async/await* syntax are used in combination with an *asyncio* compatible event loop implementation (*tornado*, *ZeroMQ*, *uvloop*, etc.).

The following examples assume that you are already familiar with *asyncio*, coroutines, and the event loop. For readability, they strip away these details to focus on the syntax used by *aiogremlin*.

To create a traversal source, simply use *DriverRemoteConnection* combined with *Graph*:

```
>>> remote_connection = await DriverRemoteConnection.open(
...     'ws://localhost:8182/gremlin', 'g')
>>>  g = Graph().traversal().withRemote(remote_connection)
```

In *aiogremlin*, a *Traversal* implements the Asynchronous Iterator Protocol as defined by PEP 492:

```
>>> async for vertex in g.V():
...     print(vertex)
```

Furthermore, it implements several convience methods - *toList*, *toSet*, and *next*:

```
>>> vertex_list = await g.V().toList()
>>> vertex_set = await g.V().toSet()
>>> next_vertex = await g.V().next()  # returns next result from the stream
```

Traversal also contains a reference to a *AsyncRemoteTraversalSideEffects* object that can be used to fetch side effects cached by the server (when applicable):

```
>>> t = g.V().aggregate('a')
>>> await t.iterate()  # evaluate the traversal
>>> keys = await t.side_effects.keys()
>>> se = await t.side_effects.get('a')
>>> await t.side_effects.close()
```

Don't forget to close the *DriverRemoteConnection* when finished:

```
>>> await remote_connection.close()
```

5.2.1.2 Using `DriverRemoteConnection`

The *DriverRemoteConnection* object allows you to configure you database connection in one of two ways:

1. Passing configuration values as kwargs or a `dict` to the classmethod *open*:

```
>>> remote_connection = await DriverRemoteConnection.open(
...     'ws://localhost:8182/gremlin', 'g', port=9430)
```

2. Passing a *Cluster* object to the classmethod *using*:

```
>>> import asyncio
>>> from aiogremlin import Cluster
>>> loop = asyncio.get_event_loop()
>>> cluster = await Cluster.open(loop, port=9430, aliases={'g': 'g'})
>>> remote_connection = await DriverRemoteConnection.using(cluster)
```

In the case that the *DriverRemoteConnection* is created with *using*, it is not necessary to close the *DriverRemoteConnection*, but the underlying *Cluster* must be closed:

```
>>> await cluster.close()
```

Configuration options are specified in the final section of this document.

DriverRemoteConnection is also an asynchronous context manager. It can be used as follows:

```
>>> async with remote_connection:
...     g = Graph().traversal().withRemote(remote_connection)
...     # traverse
# remote_connection is closed upon exit
```

Taking this one step further, the *open* can be awaited in the async context manager statement:

```
>>> async with await DriverRemoteConnection.open() as remote_connection:
...     g = Graph().traversal().withRemote(remote_connection)
...     # traverse
# remote connection is closed upon exit
```

5.2.1.3 Using the `driver` Module

aiogremlin also includes an asynchronous driver modeled after the official Gremlin-Python driver implementation. However, instead of using threads for asynchronous I/O, it uses an `asyncio` based implemenation.

To submit a raw Gremlin script to the server, use the *Client*. This class should not be instantiated directly, instead use a *Cluster* object:

```
>>> cluster = await Cluster.open(loop)
>>> client = await cluster.connect()
>>> result_set = await client.submit('g.V().hasLabel(x)', {'x': 'person'})
```

The *ResultSet* returned by *Client* implements the async interator protocol:

```
>>> async for v in result_set:
...     print(v)
```

It also provides a convenience method *all* that aggregates and returns the result of the script in a `list`:

```
>>> results = await result_set.all()
```

Closing the client will close the underlying cluster:

```
>>> await client.close()
```

5.2.1.4 Configuring the `Cluster` object

Configuration options can be set on *Cluster* in one of two ways, either passed as keyword arguments to *Cluster*, or stored in a configuration file and passed to the *open* using the kwarg *configfile*. Configuration files can be either YAML or JSON format. Currently, *Cluster* uses the following configuration:

Key	Description	Default
scheme	URI scheme, typically 'ws' or 'wss' for secure websockets	'ws'
hosts	A list of hosts the cluster will connect to	['localhost']
port	The port of the Gremlin Server to connect to, same for all hosts	8182
ssl_certfile	File containing ssl certificate	''
ssl_keyfile	File containing ssl key	''
ssl_password	File containing password for ssl keyfile	''
username	Username for Gremlin Server authentication	''
password	Password for Gremlin Server authentication	''
response_timeout	Timeout for reading responses from the stream	*None*
max_conns	The maximum number of connections open at any time to this host	4
min_conns	The minimum number of connection open at any time to this host	1
max_times_acquired	The maximum number of times a single pool connection can be acquired and shared	16
max_inflight	The maximum number of unresolved messages that may be pending on any one connection	64
message_serializer	String denoting the class used for message serialization, currently only supports basic GraphSONMessageSerializer	'classpath'

5.2.2 aiogremlin API

5.2.2.1 aiogremlin package

5.2.2.1.1 Subpackages

aiogremlin.driver package

Subpackages

aiogremlin.driver.aiohttp package

Submodules

aiogremlin.driver.aiohttp.transport module

class aiogremlin.driver.aiohttp.transport.**AiohttpTransport**(*loop*)
Bases: *gremlin_python.driver.transport.AbstractBaseTransport*

close()

closed

connect(*url*, *, *ssl_context=None*)

read()

write(*message*)

Module contents

Submodules

aiogremlin.driver.client module

Client for the Tinkerpop 3 Gremlin Server.

class aiogremlin.driver.client.**Client**(*cluster*, *loop*, *, *hostname=None*, *aliases=None*)
Bases: object

Client that utilizes a *Cluster* to access a cluster of Gremlin Server hosts. Issues requests to hosts using a round robin strategy.

Parameters

- **cluster** (aiogremlin.driver.cluster.Cluster) – Cluster used by client
- **loop** (*asyncio.BaseEventLoop*) –
- **aliases** (*dict*) – Optional mapping for aliases. Default is *None*

alias(*aliases*)

aliases
Read-only property

close()

cluster
Read-only property.

Returns The instance of *Cluster* associated with client.

message_serializer
Read-only property

submit(*message*, *bindings=None*)
coroutine Submit a script and bindings to the Gremlin Server.

Parameters

- **message** – Can be an instance of *RequestMessage<gremlin_python.driver.request.RequestMessage>* or *Bytecode<gremlin_python.process.traversal.Bytecode>* or a *str* representing a raw Gremlin script
- **bindings** (*dict*) – Optional bindings used with raw Grelmin

Returns *ResultSet* object

aiogremlin.driver.cluster module

class aiogremlin.driver.cluster.**Cluster**(*loop*, *aliases=None*, ***config*)
Bases: object

A cluster of Gremlin Server hosts. This object provides the main high level interface used by the *aiogremlin* module.

Parameters

- **loop** (*asyncio.BaseEventLoop*) –
- **aliases** (*dict*) – Optional mapping for aliases. Default is *None*
- **config** – Optional cluster configuration passed as kwargs or *dict*

DEFAULT_CONFIG = {'max_times_acquired': 16, 'ssl_password': '', 'max_conns': 4, 'p

close()
coroutine Close cluster and all connected hosts.

config
Read-only property.

Returns *dict* containing the cluster configuration

config_from_file(*filename*)
Load configuration from from file.

Parameters **filename** (*str*) – Path to the configuration file.

config_from_json(*filename*)
Load configuration from from JSON file.

Parameters **filename** (*str*) – Path to the configuration file.

config_from_module(*module*)
Load configuration from Python module.

Parameters **filename** (*str*) – Path to the configuration file.

config_from_yaml(*filename*)
Load configuration from from YAML file.

Parameters **filename** (*str*) – Path to the configuration file.

connect(*hostname=None*, *aliases=None*)
coroutine Get a connected client. Main API method.

Returns A connected instance of *Client<aiogremlin.driver.client.Client>*

establish_hosts()
coroutine Connect to all hosts as specified in configuration.

get_connection(*hostname=None*)
coroutine Get connection from next available host in a round robin fashion.

Returns *Connection*

hosts
Read-only property

classmethod open(*loop*, *, *aliases=None*, *configfile=None*, ***config*)
coroutine Open a cluster, connecting to all available hosts as specified in configuration.

Parameters

- **loop** (*asyncio.BaseEventLoop*) –
- **aliases** (*dict*) – Optional mapping for aliases. Default is *None*
- **configfile** (*str*) – Optional configuration file in .json or .yml format
- **config** – Optional cluster configuration passed as kwargs or *dict*

aiogremlin.driver.cluster.**my_import**(*name*)

aiogremlin.driver.connection module

class gremlin_python.driver.protocol.**AbstractBaseProtocol**

class aiogremlin.driver.connection.**Connection**(*url*, *transport*, *protocol*, *loop*, *username*, *password*, *max_inflight*, *response_timeout*, *message_serializer*, *provider*)

Bases: object

Main classd for interacting with the Gremlin Server. Encapsulates a websocket connection. Not instantiated directly. Instead use :py:meth::*Connection.open<aiogremlin.driver.connection.Connection.open>*.

Parameters

- **url** (*str*) – url for host Gremlin Server
- **transport** (gremlin_python.driver.transport.AbstractBaseTransport) – Transport implementation
- **protocol** (gremlin_python.driver.protocol.AbstractBaseProtocol) – Protocol implementation
- **loop** (*asyncio.BaseEventLoop*) –
- **username** (*str*) – Username for database auth
- **password** (*str*) – Password for database auth

- **max_inflight** (*int*) – Maximum number of unprocessed requests at any one time on the connection
- **response_timeout** (*float*) – (optional) *None* by default

close()
coroutine Close underlying connection and mark as closed.

closed
Read-only property. Check if connection has been closed.

Returns *bool*

message_serializer

classmethod open(*url*, *loop*, ***, *protocol=None*, *transport_factory=None*, *ssl_context=None*, *username=''*, *password=''*, *max_inflight=64*, *response_timeout=None*, *message_serializer=<class 'gremlin_python.driver.serializer.GraphSONMessageSerializer'>*, *provider=<class 'aiogremlin.driver.provider.TinkerGraph'>*)
coroutine Open a connection to the Gremlin Server.

Parameters

- **url** (*str*) – url for host Gremlin Server
- **loop** (*asyncio.BaseEventLoop*) –
- **protocol** (gremlin_python.driver.protocol.AbstractBaseProtocol) – Protocol implementation
- **transport_factory** – Factory function for transports
- **ssl_context** (*ssl.SSLContext*) –
- **username** (*str*) – Username for database auth
- **password** (*str*) – Password for database auth
- **max_inflight** (*int*) – Maximum number of unprocessed requests at any one time on the connection
- **response_timeout** (*float*) – (optional) *None* by default
- **message_serializer** – Message serializer implementation
- **provider** – Graph provider object implementation

Returns *Connection*

submit(*message*)
Submit a script and bindings to the Gremlin Server

Parameters **message** (*RequestMessage<gremlin_python.driver.request.RequestMessage>*) –

Returns *ResultSet* object

url
Readonly property.

Returns str The url association with this connection.

write(*message*)
Submit a script and bindings to the Gremlin Server

Parameters **message** (*RequestMessage<gremlin_python.driver.request.RequestMessage>*) –

Returns *ResultSet* object

aiogremlin.driver.pool module

class aiogremlin.driver.pool.**ConnectionPool**(*url*, *loop*, *ssl_context*, *username*, *password*, *max_conns*, *min_conns*, *max_times_acquired*, *max_inflight*, *response_timeout*, *message_serializer*, *provider*)

Bases: object

A pool of connections to a Gremlin Server host.

Parameters

- **url** (*str*) – url for host Gremlin Server
- **loop** (*asyncio.BaseEventLoop*) –
- **ssl_context** (*ssl.SSLContext*) –
- **username** (*str*) – Username for database auth
- **password** (*str*) – Password for database auth
- **response_timeout** (*float*) – (optional) *None* by default
- **max_conns** (*int*) – Maximum number of conns to a host
- **min_connsd** (*int*) – Minimum number of conns to a host
- **max_times_acquired** (*int*) – Maximum number of times a conn can be shared by multiple coroutines (clients)
- **max_inflight** (*int*) – Maximum number of unprocessed requests at any one time on the connection

acquire()
coroutine Acquire a new connection from the pool.

close()
coroutine Close connection pool.

init_pool()
coroutine Open minumum number of connections to host

release(*conn*)
Release connection back to pool after use.

Parameters **conn** (PooledConnection) –

url
Readonly property.

Returns str

class aiogremlin.driver.pool.**PooledConnection**(*conn*, *pool*)
Bases: object

Wrapper for *Connection* that helps manage tomfoolery associated with connection pooling.

Parameters

- **conn** (aiogremlin.driver.connection.Connection) –
- **pool** (aiogremlin.driver.pool.ConnectionPool) –

close()
Close underlying connection

closed
Readonly property.

Returns bool

decrement_acquired()
Decrement times acquired attribute by 1

increment_acquired()
Increment times acquired attribute by 1

release()

release_task(*resp*)

submit(*message*)
coroutine Submit a script and bindings to the Gremlin Server

Parameters

- **processor** (*str*) – Gremlin Server processor argument
- **op** (*str*) – Gremlin Server op argument
- **args** – Keyword arguments for Gremlin Server. Depend on processor and op.

Returns aiohttp.ClientResponse object

times_acquired
Readonly property.

Returns int

write(*message*)
coroutine Submit a script and bindings to the Gremlin Server

Parameters

- **processor** (*str*) – Gremlin Server processor argument
- **op** (*str*) – Gremlin Server op argument
- **args** – Keyword arguments for Gremlin Server. Depend on processor and op.

Returns aiohttp.ClientResponse object

aiogremlin.driver.protocol module

class aiogremlin.driver.protocol.**GremlinServerWSProtocol**(*message_serializer*, *username=''*, *password=''*)
Bases: *gremlin_python.driver.protocol.AbstractBaseProtocol*

Implemenation of the Gremlin Server Websocket protocol

connection_made(*transport*)

data_received(*data*, *results_dict*)

write(*request_id*, *request_message*)

class aiogremlin.driver.protocol.**Message**(*status_code*, *data*, *message*)
Bases: tuple

data
Alias for field number 1

message
Alias for field number 2

status_code
Alias for field number 0

aiogremlin.driver.provider module

class aiogremlin.driver.provider.**Provider**
Bases: object

Superclass for provider plugins

DEFAULT_OP_ARGS = {}

classmethod get_default_op_args(*processor*)

class aiogremlin.driver.provider.**TinkerGraph**
Bases: *aiogremlin.driver.provider.Provider*

Default provider

static get_hashable_id(*val*)

aiogremlin.driver.resultset module

class aiogremlin.driver.resultset.**ResultSet**(*request_id*, *timeout*, *loop*)
Bases: object

Gremlin Server response implementated as an async iterator.

aggregate_to

all()

close()

done
Readonly property.

Returns *asyncio.Event* object

one()
Get a single message from the response stream

queue_result(*result*)

request_id

stream

aiogremlin.driver.resultset.**error_handler**(*fn*)

aiogremlin.driver.server module

class aiogremlin.driver.server.**GremlinServer**(*url*, *loop*, ***config*)
Bases: object

Class that wraps a connection pool. Currently doesn't do much, but may be useful in the future....

Parameters **pool** (pool.ConnectionPool) –

close()
coroutine Close underlying connection pool.

get_connection()
coroutine Acquire a connection from the pool.

initialize()

classmethod open(*url*, *loop*, ***config*)
coroutine Establish connection pool and host to Gremlin Server.

Parameters

- **url** (*str*) – url for host Gremlin Server
- **loop** (*asyncio.BaseEventLoop*) –
- **ssl_context** (*ssl.SSLContext*) –
- **username** (*str*) – Username for database auth
- **password** (*str*) – Password for database auth
- **response_timeout** (*float*) – (optional) *None* by default
- **max_conns** (*int*) – Maximum number of conns to a host
- **min_connsd** (*int*) – Minimum number of conns to a host
- **max_times_acquired** (*int*) – Maximum number of times a conn can be shared by multiple coroutines (clients)
- **max_inflight** (*int*) – Maximum number of unprocessed requests at any one time on the connection

Returns *GremlinServer*

pool
Readonly property.

Returns *ConnectionPool*

url

Module contents

aiogremlin.process package

Submodules

aiogremlin.process.graph_traversal module

class aiogremlin.process.graph_traversal.**AsyncGraphTraversal**(*graph*, *traversal_strategies*, *bytecode*)

Bases: *gremlin_python.process.graph_traversal.GraphTraversal*

Implements async iteration protocol and updates relevant methods

iterate()
Iterate over results.

next(*amount=None*)
Return iterator with optionaly defined amount of items.

nextTraverser()
Return next traverser.

toList()
Reture results as `list`.

toSet()
Return results as `set`.

class aiogremlin.process.graph_traversal.**AsyncGraphTraversalSource**(**args*, ***kwargs*)

Bases: *gremlin_python.process.graph_traversal.GraphTraversalSource*

get_graph_traversal_source()

withRemote(*remote_connection*)

aiogremlin.process.traversal module

class aiogremlin.process.traversal.**AsyncTraversalStrategies**(*traversal_strategies=None*)

Bases: *gremlin_python.process.traversal.TraversalStrategies*

apply_strategies(*traversal*)

Module contents

aiogremlin.remote package

Submodules

aiogremlin.remote.driver_remote_connection module

class aiogremlin.remote.driver_remote_connection.**DriverRemoteConnection**(*client*, *loop*, ***, *cluster=None*)

Bases: object

Remote connection to a Gremlin Server. Do not instantiate directly, instead use *DriverRemoteConnection.open()* or *DriverRemoteConnection.using()*

Parameters

- **client** (aiogremlin.driver.client.Client) –
- **loop** (*asyncio.BaseEventLoop*) –
- **cluster** (aiogremlin.driver.cluster.Cluster) –

client

close()

Close underlying cluster if applicable. If created with *DriverRemoteConnection.using()*, cluster is NOT closed.

config

classmethod open(*url=None*, *aliases=None*, *loop=None*, ***, *graphson_reader=None*, *graphson_writer=None*, ***config*)

Parameters

- **url** (*str*) – Optional url for host Gremlin Server
- **aliases** (*dict*) – Optional mapping for aliases. Default is *None*. Also accepts *str* argument which will be assigned to *g*
- **loop** (*asyncio.BaseEventLoop*) –
- **graphson_reader** – Custom graphson_reader
- **graphson_writer** – Custom graphson_writer
- **config** – Optional cluster configuration passed as kwargs or *dict*

submit(*bytecode*)

Submit bytecode to the Gremlin Server

classmethod using(*cluster*, *aliases=None*)

Create a *DriverRemoteConnection* using a specific *Cluster*

Parameters

- **cluster** (aiogremlin.driver.cluster.Cluster) –

- **aliases** (*dict*) – Optional mapping for aliases. Default is *None*. Also accepts *str* argument which will be assigned to *g*

aiogremlin.remote.driver_remote_side_effects module

class aiogremlin.remote.driver_remote_side_effects.**AsyncRemoteTraversalSideEffects**(*side_effect*, *client*)

Bases: *gremlin_python.process.traversal.TraversalSideEffects*

close()
Release side effects

get(*key*)
Get side effects associated with a specific key

keys()
Get side effect keys associated with Traversal

aiogremlin.remote.remote_connection module

class aiogremlin.remote.remote_connection.**AsyncRemoteStrategy**(*remote_connection*)

Bases: *gremlin_python.driver.remote_connection.RemoteStrategy*

apply(*traversal*)

Module contents

aiogremlin.structure package

Submodules

aiogremlin.structure.graph module

class aiogremlin.structure.graph.**Graph**

Bases: *gremlin_python.structure.graph.Graph*

traversal(*traversal_source_class=None*)

Module contents

5.2.2.1.2 Submodules

5.2.2.1.3 aiogremlin.exception module

exception aiogremlin.exception.**ClientError**

Bases: Exception

exception aiogremlin.exception.**ConfigError**

Bases: Exception

exception aiogremlin.exception.**ConfigurationError**

Bases: Exception

exception aiogremlin.exception.**ElementError**
Bases: Exception

exception aiogremlin.exception.**GremlinServerError**(*status_code*, *msg*)
Bases: Exception

exception aiogremlin.exception.**MappingError**
Bases: Exception

exception aiogremlin.exception.**ResponseTimeoutError**
Bases: Exception

exception aiogremlin.exception.**ValidationError**
Bases: Exception

5.2.2.1.4 Module contents

5.2.2.2 External References

class gremlin_python.driver.transport.**AbstractBaseTransport**

class gremlin_python.driver.remote_connection.**RemoteStrategy**(*remote_connection*)

class gremlin_python.driver.request.**RequestMessage**(*processor*, *op*, *args*)

class gremlin_python.process.graph_traversal.**GraphTraversal**(*graph*, *traversal_strategies*, *bytecode*)

class gremlin_python.process.graph_traversal.**GraphTraversalSource**(*graph*, *traversal_strategies*, *bytecode=None*)

class gremlin_python.process.traversal.**Bytecode**(*bytecode=None*)

class gremlin_python.process.traversal.**Traversal**(*graph*, *traversal_strategies*, *bytecode*)

class gremlin_python.process.traversal.**TraversalSideEffects**

class gremlin_python.process.traversal.**TraversalStrategies**(*traversal_strategies=None*)

class gremlin_python.structure.graph.**Graph**

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

D

E

G

H

I

K

M