# hostray

*Release 0.7.5.1*

**Apr 15, 2020**

# Contents

**hostray** is a pure python project adding simple, scalable, and configurable module framework and utilties to open-source web frameworks. It's currently based on Tornado Web Server.

**prerequest**: python 3.6+, pip

**Install** hostray with pip: `pip install hostray`

# CHAPTER 1

## Change log

- **0.7.5.1 - Apr. 15, 2020**: * Add missing dependency, requests
- **0.7.5 - Mar. 20, 2020**:
  - Use aiohttp for sending requests asynchronously.
  - Set logger level to Error when debug set to false in server_config.yaml.
  - Method dispose() of component classes become awaitable.
  - Add certification authority 'ca' parameter for ssl settings in server_config.yaml.
  - Fix bugs.
- **0.7.3 - Dec 31, 2019**:
  - Bug Fix:
    * Arcpath (path in compressed file) might be incorrect when packing projects.
    * Changing test server port in unittest of **hostray** library to avoid conflict with default port 8888
    * The 'required' parameter of ConfigValidator classes does not work properly in all cases.
- **0.7.2 - Dec 8, 2019**:
  - Initalizing github project.

CHAPTER 2

# Documentation

## 2.1 Getting Start

**Table of Contents**



### 2.1.1 Hello, World

1. In command prompt, create a project named 'hello': `python3 -m hostray create hello`
2. Start the project: `python3 -m hostray start hello`
3. Open browser to view the hello api response, http://localhost:8888/hello
4. To stop server, press **ctrl+c** in the command prompt

### 2.1.2 Hierarchy of Project

```
project/
    component/                  # reserved folder contains the component modules.
        __init__.py
        ...
```

(continues on next page)

(continued from previous page)

```
    controller/                 # reserved folder contains the REST api controller␣
↪modules
        __init__.py
        ...
    unit_test/                  # reserved folder contains the unittest test case␣
↪modules
        __init__.py
        ...
    pack.yaml                   # defines the files to be packed to a compressed file
    requirements.txt            # defines the required pip modules
    server_config.yaml          # server configuration
```

### 2.1.3 Commands

```
>python3 -m hostray -h
usage: hostray [-h] {start,pack,test,create} ...

optional arguments:
  -h, --help            show this help message and exit

command:
  {start,pack,test,create}
    start               start server with specfied server project directory
                        path
    pack                pack server with specfied server project directory
                        path
    test                test hostray library or specfied server project
                        directory path
    create              create a server template with specfied server project
                        directory path
```

### 2.1.4 Use Component in API Controller

The hello project had defined HelloComponent, and here is a example shows how to access it in controller:

```
from hostray.web.controller import RequestController
from component import HelloComponentTypes

class HelloController(RequestController):
    async def get(self):
        hello_comp = self.component_manager.get_component(HelloComponentTypes.Hello)
        self.write(hello_comp.hello())
```

It also works with `tornado.web.RequestHandler` since `component_manager` is an attribute of `self.application`:

```
from tornado.web import RequestHandler
from component import HelloComponentTypes

class HelloTornadoHandler(RequestHandler):
    async def get(self):
```

(continues on next page)

(continued from previous page)

```
        hello_comp = self.application.component_manager.get_
↪component(HelloComponentTypes.Hello)
        self.write(hello_comp.hello())
```

**hostray** is currently based on Tornado Web Server, so please visit the web-site for advanced web framework topics.

## 2.2 Build-in Modules and Configuration

**Table of Contents**



### 2.2.1 Server Configuration

- **name** server name
- **port** port number
- **debug** enable(True)/disable(False) debug mode
- **ssl**: enable ssl module if specified and start project with subcommand `-s` which means hosting on tornado.httpserver.HTTPServer
- **component** block of component configurations
- **controller** block of component configurations

config:

```
# in server_config.yaml

name: hostray server                # server name
port: 8888                          # port number
debug: True                         # enable debug mode
ssl:
    crt: xxx.crt                    # absolute path of ssl certificate
    key: xxx.key                    # absolute path of private key file
    ca: xxx.ca                      # optional: absolute path of ca file

controller:
```

(continues on next page)

(continued from previous page)

```
    /api_route:                     # api routing path
        enum: controller_key        # key of ControllerType
        params:                     # input arguments of initialize()
            ...
    ...

component:
    component_key:                  # key of ComponentType
        ...                         # configuration vary
    ...
```

### 2.2.2 Controllers

The controllers are the implementation of RESTful APIs to handle the incomming requests.

config format:

```
controller:
    /api_route:                     # api routing path
        enum: key                   # key of ControllerType
        params:                     # input arguments of initialize()
            ...
```

#### Build-in Controllers

**enum hostray.web.controller.DefaultControllerType.Frontend** Enable a general web http server, this controller directly import tornado.web.StaticFileHandler

**value** `('frontend', 'tornado.web', 'StaticFileHandler')`

**parameter**

- **path**: relative directory path serve frontend files under project directory
- **default_filename**: default index file

rest: **get**

config:

```
controller:
    /(.*):                                  # handle all routes
        enum: frontend
        params:
            path: frontend
            default_filename: index.html
```

**enum hostray.web.controller.DefaultControllerType.SystemAlive** Response 1 to check server is alive

**value** `('server_alive', 'default_controller', 'SystemAliveController')`

rest: **get**

config:

```
controller:
    /alive:
        enum: server_alive
```

**enum hostray.web.controller.DefaultControllerType.ComponentsInfo** Response with the information of server loaded components by calling info()

**value** `('components_info', 'default_controller', 'ComponentsInfoController')`

rest: **get**

config:

```
controller:
    /components_info:
        enum: components_info
```

### 2.2.3 Components

The components of **hostray** is the functional utilities. **hostray** implements a simple composite pattern to extend the functionalities of project. **Configuration format vary**.

#### Build-in Default Components

**Attention:** **default components** are always loaded when server start.

**enum hostray.web.component.DefaultComponentTypes.Localization** Provides language localization, parameter `dir` is the path of directory that store the language `.csv` files under project directory. Class Reference

**value** `('localization', 'default_component', 'LocalizationComponent')`

**parameters**

- **dir** - optional: load all of the .csv files in local/ under project directory if specified
- **lang** - optional: setup language, default: en

config:

```
component:
    localization:
        dir: 'local'
        lang: 'en'
```

**.csv file example**

```
code,en,tw
10000,"this is code 10000", code 10000
```

**enum hostray.web.component.DefaultComponentTypes.Logger** Provides **hostray** customized logger, parameter `dir` is the path of directory that store the log outputs under project directory

**value** `('logger', 'default_component', 'LoggerComponent')`

**parameters**

- **dir** - optional. If specified, save log to the folder under porject directory

config:

```
component:
    logger:
        dir: 'logs'
```

**enum hostray.web.component.DefaultComponentTypes.Callback** Callback management with customized `enums`, no configuration needed

**value** `('callback', 'default_component', 'CallbackComponent')`

**enum hostray.web.component.DefaultComponentTypes.WorkerPool** Provides blocking access thread pools to execute functions

**value** `('worker_pool', 'default_component', 'WorkerPoolComponent')`

**parameters** **pool_id** : **workers** - specified pool id and the number of workers of that pool

config:

```
component:
    worker_pool:
        default: 2      # pool_id default with the worker maximum is 2
```

**enum hostray.web.component.DefaultComponentTypes.TaskQueue** Provides non-blocking access thread pool to execute functions

**value** `('task_queue', 'default_component', 'TaskQueueComponent')`

**parameters**

- **worker_count** - number of queues

```
component:
    task_queue:
        worker_count: 2     # 2 task queue workers
```

### Build-in Optional Components

**enum hostray.web.component.OptionalComponentTypes.Service** Invokes web api, specified method name to enable rest mehtods

**value** `('services', 'optional_component', 'ServicesComponent')`

**parameters**

- **url** - url
- **route** - api route
- **name** - id
- **method_names** - rest method names

config:

```
component:
    services:
        https://www.google.com:          # url
            /:                           # api_route
                name: google             # name of this invoker
                get:                     # enable method get
```

**enum hostray.web.component.OptionalComponentTypes.MemoryCache** Simple backend Session(cache) system

**value** `('memory_cache', 'optional_component', 'MemoryCacheComponent')`

**parameters**

- **sess_lifetime** - session lifetime in seconds
- **renew_lifetime** - renew lifetime when accquire session
- **renew_id** - renew session id (token) when accquire session
- **save_file** - save/reload cache via file if specified when server start/stop

config:

```
component:
    memory_cache:
        sess_lifetime: 600
        save_file: file_name
        renew_lifetime: False
        renew_id: False
```

**enum hostray.web.component.OptionalComponentTypes.OrmDB** Orm component for accessing databases based on sqlalchemy which support many backend databses.

**value** `('orm_db', 'optional_component', 'OrmDBComponent')`

**parameters**

- **db_id** - specified and used in code
  - **module** - switch parameter: `sqlite_memory`, `sqlite`, `mysql`
  - **connection_refresh** - minimum interval in seconds to refresh connection, no effect in module `sqlite_memory`
  - **worker** - number of db access worker (connections)
  - **db_connection_parameters** - vary in different modules, check the following config example

config:

```
component:
    orm_db:
        db_0:                                # id of db module
            module: sqlite_memory            # switch: use sqlite_memory
            worker: 1                        # number of db access worker␣
↪(connection)
            connection_refresh: 60           # no effect

        db_1:
```

(continues on next page)

(continued from previous page)

```
            module: sqlite                  # switch: use sqlite
            worker: 1
            connection_refresh: 60          # minimum interval in␣
↪seconds to refresh connection
            file_name: data.db              # sqlite file path under␣
↪project directory

        db_2:
            module: mysql                   # switch: use mysql
            worker: 1
            connection_refresh: 60          # minimum interval in␣
↪seconds to refresh connection
            host: xxx.xxx.xxx.xxx           # mysql host ip
            port: 3306                      # mysql host port
            db_name: xxxxxxx                # mysql database_name
            user: xxxxxxxx                  # mysql login user
            password: xxxxxxxx              # mysql login password
```

**Note:** The worker instances hold the sessions and database connections and refresh them until next db accession considers the parameter 'connection_refresh' as the minimum interval.

**Note:** Module 'sqlite_memory' does not refresh connections since it is a memory database and will be released if the connection closed.

### 2.2.4 Unittest Cases

**hostray** reserves module **unit_test** base on unittest to test the server project or **hostray** library. Define enum inherits hostray.unit_test.UnitTestTypes to allow **hostray** tests projects

- Run test in command prompt:
  - Test hostray library: `python3 -m hostray test`
  - Test hostray project: `python3 -m hostray test <project directory path>`

### 2.2.5 Packing Project

Packing project by typing `python3 -m hostray pack <project directory path>` in command prompt.

The optional flags of command `pack`:

- Adding `-w` downloads and pack the wheel `.whl` lists in `requirements.txt`.
- In default, `.py` files are compiled to `.pyc`. Adding `-d` to disable the compilation.

In **hostray** project, `pack.yaml` indicated the files should be packed. The block of `include` lists the external **files** or **directories**, and the block of `exclude` lists the **files**, **directories**, or **extensions** should be ignored.

example:

```
# inside pack.yaml...

include:
- some_file.txt          # pack some_file.txt
- some_dir/              # pack directory 'some_dir' recursively

exclude:
- '.log'                 # excludes files with extension '.log'
- some_dir2/             # excludes files and sub directories under some_dir2␣
↪recursively
- some_file2.txt         # excludes some_file2.txt
```

## 2.3 Modules Framework

**Table of Contents**



### 2.3.1 Reserved Modules

**hostray** project reserves the directories named **controller**, **component**, and **unit_test** as the modules of functionalities, rest apis, and unit testing cases. It's very similar to how python recognizes the packages, **hostray** load the modules with the directory contains `__init__.py` that defines specific type `enum`.

For example, the component module of hello project created in Getting Start, the directory hierarchy looks like:

```
hello/
    component/
        __init__.py
        hello.py
    server_config.yaml
```

In `__init__.py`, it defines the subclass of `ComponentTypes`. `ComponentTypes` is a customized subclass of Eunm class. Its value is a `tuple` stores `(key, package, class_or_function)` for **hostray** maps the component configurations in `server_config.yaml` with the component classes should be imported. The code of `__init__.py` looks like:

```
from hostray.web.component import ComponentTypes

class HelloComponentTypes(ComponentTypes):
    Hello = ('hello', 'hello', 'HelloComponent')
```

In `hello.py`, it defines the `HelloComponent` inherits from `Component` class, the code looks like:

```
from hostray.web.component import Component
from . import HelloComponentTypes
```

(continues on next page)

(continued from previous page)

```
class HelloComponent(Component):
    def init(self, component_manager, p1, *arugs, **kwargs) -> None:
        print('init Hello component load from', __package__, 'and the parameters p1:',
↪ p1)

    def hello(self):
        return 'Hello World, This is hostray generate hello component'
```

In `server_config.yaml`, add the key **'hello'** under component block. That tells **hostray** load **HelloComponent** when starting api server:

```
# in server_config.yaml

name: hostray Server
port: 8888
debug: False
component:
    hello:
        p1: 'This is p1'
```

### 2.3.2 Make Project's Modules Work

Briefly lists the things should be done for each reserved module:

- controller
  - Defines enums inherits from `hostray.web.controller.ControllerType` in `__init__.py`
  - Implements the controller inherits hostray build-in controllers or directly use tornado.web handlers
  - Configres the controller block in `server_config.yaml`
- component
  - Defines enums inherits from `hostray.web.component.ComponentTypes` in `__init__.py`
  - Implements the component class inherits from hostray.web.component.Component
  - Configres the component block in `server_config.yaml`
- unit_test
  - Defines enums inherits from `hostray.unit_test.UnitTestTypes` in `__init__.py`
  - Implements the test cases inherits from hostray.unit_test.UnitTestCase

### 2.3.3 Configuration Validator

**hostray** provides configurations validator checks the build-in components and controllers. The validator is extendable to validate project extended components and controllers, and the following example shows how to add validator of hello project's HelloComponent.

```
# in __init__.py of project's component module

from hostray.web.component import ComponentTypes
```

(continues on next page)

(continued from previous page)

```
from hostray.web.config_validator import ConfigContainerMeta, ConfigElementMeta,␣
↪HostrayWebConfigComponentValidator

# add hello validator to component config validator
HostrayWebConfigComponentValidator.set_cls_parameters(
    ConfigContainerMeta('hello', False,
        ConfigElementMeta('p1', str, True) # validate HelloComponent's 'p1' argument␣
↪is required and string type
    )
)

class HelloComponentTypes(ComponentTypes):
    Hello = ('hello', 'hello', 'HelloComponent')
```

## 2.4 Usage of Utils

**Table of Contents**



### 2.4.1 Worker

`hostray.util.worker` based on threading customized workers (thread classes) and pools to manage the non-async and async executions and provides context manager functions called `reserve_worker` and `reserve_worker_async` to exexutes the given functions in the same worker.

Workers are the wrappers of threading customized for different operations:

- `hostray.util.worker.Worker`: `run_method` executes the given function once
- `hostray.util.worker.FunctionQueueWorker`: `run_method` queues the given functions and execute them when worker is free
- `hostray.util.worker.FunctionLoopWorker`: `run_method` executes and loop the the given function with the specfied interval

examples:

```
import time
from hostray.util import Worker, FunctionLoopWorker, FunctionQueueWorker

def foo(index, kwindex):
    print(index)

print('Worker')
with Worker() as worker:
    worker.run_method(foo, 1, kwindex=2)
```

(continues on next page)

(continued from previous page)

```
    while worker.is_func_running:                    # wait for executions
        pass

print('FunctionQueueWorker')
with FunctionQueueWorker() as worker:
    worker.run_method(foo, 1, kwindex=2)
    worker.run_method(foo, 2, kwindex=2)
    worker.run_method(foo, 3, kwindex=2)

    while worker.pending_count > 0:                  # wait for executions
        pass

print('FunctionLoopWorker')
with FunctionLoopWorker(loop_interval_seconds=0.1) as worker:
    wait_time = 0.5
    worker.run_method(foo, 1, kwindex=2)

    start_time = time.time()
    while time.time() - start_time < wait_time:      # wait for executions
        pass

'''
Output:

Worker
1
FunctionQueueWorker
1
2
3
FunctionLoopWorker
1
1
1
1
1
'''
```

Pools manage workers to handle sync and async executions:

- `hostray.util.worker.WorkerPool`: worker management and handle sync functions
- `hostray.util.worker.AsyncWorkerPool`: add async functions to `hostray.util.workerWorkerPool`

example:

```
import asyncio
from hostray.util import AsyncWorkerPool

ap = AsyncWorkerPool(worker_limit=3)

def foo(index, kwindex):
    print(index)

print('run sync')
with ap.reserve_worker() as identity:
```

(continues on next page)

(continued from previous page)

```
    ap.run_method(foo, 1, 1, identity=None)          # run in free worker
    ap.run_method(foo, 2, 2, identity=identity)      # run in reserved worker

async def async_foo():
    async with ap.reserve_worker_async() as identity:
        await ap.run_method_async(foo, 1, 1, identity=None)          # run in␣
↪free worker
        await ap.run_method_async(foo, 2, 2, identity=identity)      # run in␣
↪reserved worker

loop = asyncio.get_event_loop()

print('run async')
loop.run_until_complete(async_foo())

ap.dispose()

'''
Output:

run sync
1
2
run async
1
2
'''
```

### 2.4.2 ORM

sqlalchemy is popular Python SQL toolkit and Object Relational Mapper (ORM). `hostray.util.orm` wraps the ORM modules of sqlalchemy to simplify usage of database

- `hostray.util.orm.DB_MODULE_NAME`: `enum` defines the type of database access modules in `SQLITE_MEMORY`, `SQLITE_FILE`, and `MYSQL`.
- `hostray.util.orm.get_declarative_base`: function returns key-managed singleton sqlalchemy.ext.declariative.api.DeclarativeMeta
- `hostray.util.orm.get_session_maker`: function returns sqlalchemy.orm.Session
- `hostray.util.orm.EntityBaseAddon`: add helper functions to entity classes
- `hostray.util.orm.OrmAccessWorkerPool`: class manages database access workers
- `hostray.util.orm.OrmDBEntityAccessor`: class defines how to access database with entity instances

example:

```
from enum import Enum
from datetime import datetime

from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import Session

from hostray.util.orm import (get_declarative_base, EntityBaseAddon, get_
↪session_maker,
```

(continues on next page)

(continued from previous page)

```
                                OrmAccessWorkerPool, OrmDBEntityAccessor, DB_
↪MODULE_NAME)

# 1. create a DeclarativeBase metaclass to collect orm table schema
DeclarativeBase = get_declarative_base()


class GenderType(Enum):
    Male = 'male'
    Female = 'female'


# 2. define entity class inherits DeclarativeBase.
#    EntityBaseAddon defines the helper functions for OrmDBEntityAccessor
class PersonEntity(DeclarativeBase, EntityBaseAddon):
    __tablename__ = 'person'

    id = Column(Integer, primary_key=True)
    name = Column(String(40), nullable=False)
    age = Column(Integer, nullable=False)
    gender = Column(String(6), nullable=False)
    secret = Column(String(40))
    note = Column(String(100))
    schedule = Column(DateTime, default=datetime.now)

    column_type_validations = {'gender': GenderType}        # helper for␣
↪OrmDBEntityAccessor
    column_fix = ['name']                                   # helper for␣
↪OrmDBEntityAccessor
    client_excluded_columns = ['secret']                    # helper for␣
↪OrmDBEntityAccessor


# 3. define accessor class inherits OrmDBEntityAccessor
#    accessor define how to access database table
#    OrmDBEntityAccessor define basic usage to access database single table
class PersonAccessor(OrmDBEntityAccessor):
    def __init__(self):
        super().__init__(PersonEntity) # use PersonEntity

if __name__ == '__main__':
    # get sqlalchemy Session instance
    sess = get_session_maker(DB_MODULE_NAME.SQLITE_MEMORY, DeclarativeBase)()

    # access database
    accessor = PersonAccessor()
    entity = accessor.add(sess, name='someone', age=30,
                                gender='male', secret='my secret', note='this␣
↪is note')

    accessor.save(sess)
    accessor.refresh(sess, entity)
    accessor.set_attribute(sess, entity, age=50, gender='female')

    try:
        accessor.set_attribute(sess, entity, name='sometwo') # exception␣
↪column 'name' fixed
```

(continues on next page)

(continued from previous page)

```
        accessor.save(sess)
    except:
        accessor.rollback(sess)
        accessor.refresh(sess, entity)

    print(entity.to_dict())
    sess.close()

'''
output:

{'note': 'this is note', 'gender': 'male', 'age': 30, 'id': 1, 'schedule':
↪'2019-12-16 17:49:28.385881', 'secret': 'my secret', 'name': 'someone'}
'''
```

example using pool:

```
from enum import Enum
from datetime import datetime

from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import Session

from hostray.util.orm import (get_declarative_base, EntityBaseAddon,
                              OrmAccessWorkerPool, OrmDBEntityAccessor, DB_
↪MODULE_NAME)

# 1. create a DeclarativeBase metaclass to collect table schema
DeclarativeBase = get_declarative_base()

class GenderType(Enum):
    Male = 'male'
    Female = 'female'


# 2. define entity class inherits DeclarativeBase.
#    EntityBaseAddon defines the helper functions for OrmDBEntityAccessor
class PersonEntity(DeclarativeBase, EntityBaseAddon):
    __tablename__ = 'person'

    id = Column(Integer, primary_key=True)
    name = Column(String(40), nullable=False)
    age = Column(Integer, nullable=False)
    gender = Column(String(6), nullable=False)
    secret = Column(String(40))
    note = Column(String(100))
    schedule = Column(DateTime, default=datetime.now)

    column_type_validations = {'gender': GenderType}        # helper for␣
↪OrmDBEntityAccessor
    column_fix = ['name']                                   # helper for␣
↪OrmDBEntityAccessor
    client_excluded_columns = ['secret']                    # helper for␣
↪OrmDBEntityAccessor


# 3. define accessor class inherits OrmDBEntityAccessor
```

(continues on next page)

(continued from previous page)

```
#    accessor define how to access database table
#    OrmDBEntityAccessor define basic usage to access database single table
class PersonAccessor(OrmDBEntityAccessor):
    def __init__(self):
        super().__init__(PersonEntity)  # use PersonEntity


if __name__ == '__main__':
    pool = OrmAccessWorkerPool()
    pool.set_session_maker(DB_MODULE_NAME.SQLITE_MEMORY, DeclarativeBase)
    accessor = PersonAccessor()

    with pool.reserve_worker() as identity:
        entity = pool.run_method(accessor.add, name='someone', age=30,
                                 gender='male', secret='my secret', note=
↪'this is note', identity=identity)

        pool.run_method(accessor.save, identity=identity)
        pool.run_method(accessor.refresh, entity, identity=identity)
        try:

            pool.run_method(accessor.set_attribute, entity,
                            name='sometwo', identity=identity)          #␣
↪exception column 'name' fixed
            pool.run_method(accessor.save, identity=identity)
        except:
            pool.run_method(accessor.rollback, identity=identity)
            pool.run_method(accessor.refresh, entity, identity=identity)

        print(entity.to_dict())

    pool.dispose()

'''
output:

{'schedule': '2019-12-16 17:44:33.229172', 'secret': 'my secret', 'gender':
↪'male', 'name': 'someone', 'note': 'this is note', 'age': 30, 'id': 1}
'''
```

### 2.4.3 More Support Utilities

- **Localization** store and mapping the Localized Messages

  example:

```
from hostray.util import Localization

local = Localization()
local.import_csv(['xxx.csv'])           # import language file

print(local.get_message(1111))          # print the code refered message
```

- **Logger** is customized logging module to specfied the logger's handlers

  example:

```
from hostray.util import get_Hostray_logger

logger = get_Hostray_logger('test', log_to_resource=True)    # log to␣
↪current working directory
logger.info('hello')
```

- **hostray** datetime helper:

  example:

```
from hostray.util import datetime_to_str, str_to_datetime, DATETIME_TYPE

dt = str_to_datetime('2019-12-17T12:02:58')          # parse dot net␣
↪format string
print(dt)                                            # python datetime␣
↪string
print(datetime_to_str(dt, DATETIME_TYPE.DTF1))       # to dot net datetime␣
↪string

'''
output

2019-12-17 12:02:58
2019-12-17T12:02:58.000000
'''
```

- **Callback** is a `enum` managed async and sync callback function container.

  example:

```
import asyncio
from enum import Enum

from hostray.util import Callbacks

# 1. define enum and functions
class TestCallbackType(Enum):
    Event_A = 'a'
    Event_A_Async = 'a_async'


def test_func_1(i, kwindex):
    print('test_func_1', i)


def test_func_2(i, kwindex):
    print('test_func_2', i)


async def test_func_async_1(i, kwindex):
    print('test_func_async_1', i)


async def test_func_async_2(i, kwindex):
    print('test_func_async_2', i)

cb = Callbacks(TestCallbackType)
```

(continues on next page)

(continued from previous page)

```
# 2. add callbacks
cb.add_callback(TestCallbackType.Event_A, test_func_1)
cb.add_callback(TestCallbackType.Event_A, test_func_2)
cb.add_callback(TestCallbackType.Event_A_Async,
                test_func_async_1)
cb.add_callback(TestCallbackType.Event_A_Async,
                test_func_async_2)

# 3. invoke callbacks
cb.execute_callback(TestCallbackType.Event_A, 1, kwindex=2)

loop = asyncio.get_event_loop()
loop.run_until_complete(cb.execute_callback_async(
    TestCallbackType.Event_A_Async, 1, kwindex=2))

'''
output:

test_func_1 1
test_func_2 1
test_func_async_2 1
test_func_async_1 1
'''
```

## 2.5 `hostray.web` Reference

**Table of Contents**



### 2.5.1 Controllers

**class** hostray.web.controller.**ControllerAddon**

A helper class defines quick function access components

sub_classes:

- hostray.web.controller.RequestController
- hostray.web.controller.WebSocketController

**get_localized_message**(*code: Union[str, int], *args*) → str

quick function to get localized message by

**run_method_async**(*func: Callable*, **args*, *pool_id: str = 'default'*, ***kwargs*) → Any

awaitable, quick function to execute function in pool

**log_info**(*msg: str*, **args*, *exc_info=None*, *extra=None*, *stack_info=False*) → None
quick function to log info level message

**log_warning**(*msg: str*, **args*, *exc_info=None*, *extra=None*, *stack_info=False*) → None
quick function to log warning level message

**log_error**(*msg: str*, **args*, *exc_info=None*, *extra=None*, *stack_info=False*) → None
quick function to log error level message by

**invoke_service_async**(*service_name : str*, *method : str = 'get'*, *streaming_callback : Callable = None*, ***kwargs*) → Response
awaitable, quick function to send http request by service Component

**class** hostray.web.controller.**RequestController**
Class inherits from tornado.web.RequestHandler. Please check the usage of tornado documentation

**class** hostray.web.controller.**StreamingDownloadController**
Abstract class inherits from hostray.web.controller.RequestController.

**_prepare_binary**() → bytes
override this awaitable function to prepare binary data for downloading

**class** hostray.web.controller.**StreamingUploadController**
Abstract class inherits from hostray.web.controller.RequestController.

**_on_chunk_received(self, headers, chunk, bytes_size_received):**
override this function to process incoming chunk data

**_on_data_received(self, headers, bytes_size_received):**
override this function to do process after data transaction completed

**class** hostray.web.controller.**StreamingFileUploadController**
Class inherits from hostray.web.controller.RequestController.

**class** hostray.web.controller.**WebSocketController**
Class inherits from tornado.websocket.WebSocketHandler. Please check the usage of tornado documentation

### 2.5.2 Components

**class** hostray.web.component.default_component.**Component**
Base abstract class of component

**init**(*component_manager*, **arugs*, ***kwargs*) → None
called when component_manager initialize component objects

**info**() → Dict
return define meta information of component

**dispose**(*component_manager*) → None
called when server stop

---

**Note:** Be aware of the component dependencies when server start/stop, the loaded components are sorted by the order of enums:

**server start DefaultComponentTypes -> OptionalComponentTypes -> Project_ComponentTypes**

**server stop Project_ComponentTypes -> OptionalComponentTypes -> DefaultComponentTypes**

---

**class** hostray.web.component.default_component.**ComponentManager**
Contain and manage the loaded components

**@property components -> List[Component]**
return list of loaded components

**@property info -> Dict**
return info of loaded components

**dispose**() → None
call dispose() of loaded components

**boardcast**(*method: str*, **arugs*, ***kwargs*) → List[Tuple[ComponentTypes, Any]]
invokes the non-awaitable method of stored components and return a list of returns from each component method

- **method**: str, method name
- ***args**: variable number of arguments of method
- ****kwargs**: keyworded, variable-length argument list of method

**boardcast_async**(*method: str*, **arugs*, ***kwargs*) → List[Tuple[ComponentTypes, Any]]
invokes both awaitable and non-awaitable method of stored components and return a list of returns from each component method

- **method**: str, method name
- ***args**: variable number of arguments of method
- ****kwargs**: keyworded, variable-length argument list of method

**invoke**(*enum_type: ComponentTypes*, *method: str*, **arugs*, ***kwargs*) → Any
execute component mehtod by giving the method name and arguments

- **enum_type**: ComponentTypes enum type
- **method**: str, method name
- ***args**: variable number of arguments of method
- ****kwargs**: keyworded, variable-length argument list of method

**invoke_async**(*enum_type: ComponentTypes*, *method: str*, **arugs*, ***kwargs*) → Any
asynchronously execute component mehtod by giving the method name and arguments

- **enum_type**: ComponentTypes enum type
- **method**: str, method name
- ***args**: variable number of arguments of method
- ****kwargs**: keyworded, variable-length argument list of method

**set_component**(*component: Component*) → None
add or replace component instance

- **component**: Component instance

**get_component**(*enum_type: ComponentTypes*) → Union[Component, None]
return stored component instance or None

- **enum_type**: ComponentTypes enum type

**pick_component**(*enum_types: List[ComponentTypes]*) → Union[Component, None]
return the first founded stored component object of enum_types

- **enum_type**: ComponentTypes enum type

**has_component** (*enum_type: ComponentTypes*) → bool
check whether component exists

- **enum_type**: ComponentTypes enum type

**sort_components** (*order_list: List[ComponentTypes]*) → None
sort component object with ComponentTypes in order

- **order_list**: list of ComponentTypes

**class** hostray.web.component.default_component.**LocalizationComponent**

**set_language** (*lang: str*) → None
set language

- **lang**: key of language such as 'en'

**get_message** (*code: str*, **args*) → str
return the message refer to 'code' and *args

- **code**: localized message code
- ***args**: variable number of arguments of `str`

sample:

```
from hostray.web.controller import RequestController
from hostray.web.component import DefaultComponentTypes

class FooController(RequestController):
    async def get(self):
        comp = self.component_manager.get_component(DefaultComponentTypes.
↪Localization)
        self.write(comp.get_message(10000))
```

**class** hostray.web.component.default_component.**LoggerComponent**

**set_default_logger_echo** (*echo: bool*) → None
enable/disable default loggers print to stdout

- **echo**: print log to command prompt

```
default_loggers = ['tornado.access',
                   'tornado.application',
                   'tornado.general',
                   'sqlalchemy']
```

**get_logger** (*name: str*, *sub_dir: str = ''*, *mode: str = 'a'*, *encoding: str = 'utf-8'*, *echo: bool = False*)
→ HostrayLogger
get HostrayLogger singleton object

- **name**: logger name
- **sub_dir**: specfied sub dir of log dir if enable logging to file
- **mode**: filemode
- **encoding**: text encoding
- **echo**: print log to command prompt

sample:

```
from hostray.web.controller import RequestController
from hostray.web.component import DefaultComponentTypes

class FooController(RequestController):
    async def get(self):
        comp = self.component_manager.get_component(DefaultComponentTypes.Logger)
        logger = comp.get_logger('some_logger')
```

**class** hostray.web.component.default_component.**CallbackComponent**

**get_callback_obj**(*enum_cls: Enum*) → Callbacks
return callback function instance

- **enum_cls**: class of enum

**add_callback**(*callback_enum_type: Enum*, *callback: Callable*) → None
registered callback function instance

- **callback_enum_type**: type class of enum
- **callback**: callback function

**remove_callback**(*callback_enum_type: Enum*, *callback: Callable*) → None
remove callback function instance

- **callback_enum_type**: type class of enum
- **callback**: callback function

**execute_callback**(*callback_enum_type: Enum*, **args*, ***kwargs*) → None
execute registered callback functions

- **callback_enum_type**: type class of enum
- ***args**: variable number of arguments of callback functions
- ****kwargs**: keyworded, variable-length argument list of callback functions

**execute_callback_async**(*callback_enum_type: Enum*, **args*, ***kwargs*) → None
asynchronously execute registered callback functions

- **callback_enum_type**: type class of enum
- ***args**: variable number of arguments of callback functions
- ****kwargs**: keyworded, variable-length argument list of callback functions

**class** hostray.web.component.default_component.**TaskQueueComponent**

**run_method_in_queue**(*func: Callable*, **args*, *on_finish: Callable[[Any], None] = None*, *on_exception: Callable[[Exception], None] = None*, ***kwargs*) → None
queue function and execute in differet thread

- **func**: function object
- ***args**: variable number of arguments of function object
- **on_finish**: callback when function finished
- **on_exception**: callback when function exception occurs
- ****kwargs**: keyworded, variable-length argument list of function object

**Attention: run_method_in_queue() Does Not** block the thread

**class** hostray.web.component.default_component.**WorkerPoolComponent**

**set_pool**(*pool_id: str = 'default'*, *worker_limit: int = 3*) → None
creates pool if it does not exist and setup the worker maximum by 'pool_id'

- **pool_id**: the id of pool
- **worker_limit**: maximum of workers

**run_method**(*func: Callable*, **args*, *pool_id: str = 'default'*, ***kwargs*) → Any
execute func in pool with specfied 'pool_id'

- **func**: function object
- ***args**: variable number of arguments of function object
- ****kwargs**: keyworded, variable-length argument list of function object

**run_method_async**(*func: Callable*, **args*, *pool_id: str = 'default'*, ***kwargs*) → Any
asynchronously execute func in pool with specfied 'pool_id'

- **func**: function object
- ***args**: variable number of arguments of function object
- ****kwargs**: keyworded, variable-length argument list of function object

**Attention: run_method() Does** block the thread

**class** hostray.web.component.optional_component.**MemoryCacheComponent**

**get_expired_datetime**(*session_id: str*) → datetime
Return the datetime the session id expired

- **session_id**: session id

**get**(*session_id: str = ''*, *renew_lifetime: bool = False*, *renew_id: bool = False*) → Tuple[dict, str]
Return tuple (cache, session_id).

- **session_id**: session id
- **renew_lifetime**: renew the expired datetime of the session_id
- **renew_id**: return new session_id if set to True

**save_to_file**() → None
save current cache to file if the config parameter 'save_file' specfied

**load_from_file**() → None
load file if the config parameter 'save_file' specfied to cache

**clear_session**(*session_id: str*) → None
clear cache of the session_id

- **session_id**: session id

**class** hostray.web.component.optional_component.**OrmDBComponent**

Managing sqlalchemy db access worker pools and execute `hostray.util.orm.OrmDBEntityAccessor`

**get_pool_obj**(*db_id: str*) → OrmAccessWorkerPool
return the db access wokrer pool object of db_id

- **db_id**: id of db access wokrer pool

**get_db_settings**(*db_id: str*) → Dict
return the db setting of db_id

- **db_id**: id of db access wokrer pool

**init_db_declarative_base**(*db_id: str*, *declared_entity_base: DeclarativeMeta*) → None
create and initialize sqlalchemy orm meta class and engine of db_id

- **db_id**: id of db access wokrer pool
- **declared_entity_base**: sqlalchemy orm meta class

**reserve_worker**(*db_id: str*) → str
contextmanager wrapped funciton to reserve worker, return the identity `str`

- **db_id**: id of db access wokrer pool

**reserve_worker_async**(*db_id: str*) → str
asynccontextmanager wrapped funciton to reserve worker, return the identity `str`

- **db_id**: id of db access wokrer pool

**reset_session**(*db_id: str*, *force_reconnect: bool = False*) → None
reset db session and connection

- **db_id**: id of db access wokrer pool
- **force_reconnect**: ignore minimum interval 'connection_refresh' and reset db session and connection

**reset_session_async**(*db_id: str*, *force_reconnect: bool = False*) → None
asynchronously reset db session and connection

- **db_id**: id of db access wokrer pool
- **force_reconnect**: ignore minimum interval 'connection_refresh' and reset db session and connection

**run_accessor**(*db_id: str*, *accessor_func: Callable*, **args*, *identity: str = None*, ***kwargs*) → Any
execute function of `hostray.util.orm.OrmDBEntityAccessor`

- **db_id**: id of db access wokrer pool
- **accessor_func**: function of `hostray.util.orm.OrmDBEntityAccessor`
- ***args**: variable number of arguments of accessor function object
- ****kwargs**: keyworded, variable-length argument list of accessor function object

**run_accessor_async**(*db_id: str*, *accessor_func: Callable*, **args*, *identity: str = None*, ***kwargs*) → Any
asynchronously execute function of `hostray.util.orm.OrmDBEntityAccessor`

- **db_id**: id of db access wokrer pool
- **accessor_func**: function of `hostray.util.orm.OrmDBEntityAccessor`
- ***args**: variable number of arguments of accessor function object
- ****kwargs**: keyworded, variable-length argument list of accessor function object

**class** hostray.web.component.optional_component.**ServicesComponent**

**invoke**(*service_name: str*, *method='get'*, *streaming_callback: Callable = None*, ***kwargs*) → requests.Response
seed http request to config specfied service_name and return `requests.Response` object

- **service_name**: config specfied service_name
- **method**: http methods `['get', 'post', 'patch', 'put', 'delete', 'option']`
- **streaming_callback**: streaming operation callback function, check Reference
- ****kwargs**: keyworded, variable-length argument list of http method parameters

**invoke_async**(*service_name: str*, *method='get'*, *streaming_callback: Callable = None*, ***kwargs*) → requests.Response
asynchronously seed http request to config specfied service_name and return `requests.Response` object

- **service_name**: config specfied service_name
- **method**: http methods `['get', 'post', 'patch', 'put', 'delete', 'option']`
- **streaming_callback**: streaming operation callback function, check Reference
- ****kwargs**: keyworded, variable-length argument list of http method parameters

### 2.5.3 Unit_test

**class** hostray.unit_test.**UnitTestCase**
Abstract class of test case

**test**() → None
override this function to implement unittest code

### 2.5.4 Configuration Validator

**class** hostray.web.config_validator.**ConfigBaseElementMeta**
base config element metaclass

**set_cls_parameters**(**cls_parameters*) → None
@**classmethod**, set the sub class elements

- ***parameters**: variable number of arguments of ConfigBaseElementMeta

**get_cls_parameter**(*key_routes*, *delimeter="."*) → type
@**classmethod**, get the sub class elements

- **key_routes**: route in `str`
- **delimeter**: delimeter of route.split()

**get_parameter**(*key_routes: str*, *delimeter: str = '.'*)
return parameter of specfied key_routes

- **key_routes**: route in `str`
- **delimeter**: delimeter of route.split()

**class** hostray.web.config_validator.**ConfigContainerMeta**
Configration validation element metaclass contain sub elements

**__new__**(*name: str*, *required: bool*, **parameters*) → type

- **name**: name of type
- **required**: specfied is this element is required in config
- ***parameters**: variable number of arguments of ConfigBaseElementMeta

**copy**(*name*) → type

- **name**: name of copied type

**class** hostray.web.config_validator.**ConfigElementMeta**

Configration validation element metaclass store parameters

**__new__**(*name: str*, *parameter_type: Any*, *required: bool*) → type

- **name**: name of type
- **parameter_type**: variable type such `str, int, float`
- **required**: specfied is this element is required in config

**copy**(*name*) → type

- **name**: name of copied type

**class** hostray.web.config_validator.**ConfigScalableContainerMeta**

scalable configration validation element metaclass contain sub elements metaclass

**__new__**(*parameter_type: Union[str, int], *parameters*) → type

- **parameter_type**: variable type such `str, int, float`
- ***parameters**: variable number of arguments of ConfigBaseElementMeta

**copy**(*name*) → type

- **name**: name of copied type

**class** hostray.web.config_validator.**ConfigScalableElementMeta**

scalable configration validation element metaclass

**__new__**(*element_type: Union[str, int], parameter_type: Any*) → type

- **element_type**: scalable key variable type such as `str, int, float`
- **parameter_type**: variable type such as `str, int, float`

**copy**(*name*) → type

- **name**: name of copied type

**class** hostray.web.config_validator.**ConfigSwitchableElementMeta**

switchable configration validation element metaclass

**__new__**(*name: str*, *parameter_type: Any*, *required: bool*, **parameters*) → type

- **name**: name of type
- **parameter_type**: variable type
- **required**: specfied is this element is required in config
- ***parameters**: variable number of arguments of ConfigBaseElementMeta

**copy**(*name*) → type

- **name**: name of copied type

**class** hostray.web.config_validator.**HostrayWebConfigValidator**
default validator to validate `server_config.yaml`.

**class** hostray.web.config_validator.**HostrayWebConfigControllerValidator**
default validator to validate the controller block of `server_config.yaml`.

**class** hostray.web.config_validator.**HostrayWebConfigComponentValidator**
default validator to validate the component block of `server_config.yaml`.

## 2.6 hostray.util Reference

**Table of Contents**



### 2.6.1 Worker

**class** hostray.util.worker.**Worker**
property:

- `is_func_running -> bool`: check if worker is running a function

**run_method**(*func: Callable, *args, on_finish: Callable[[Any], None] = None, on_exception: Callable[[Exception], None] = None, **kwargs*) → bool
return True if the given function instance is going to be executed

- **func**: function instance to be executed
- **on_finish**: callback with the argument of function return after function runned
- ***args**: variable number of arguments of method
- **on_exception**: callback with the argument of Exception after function Exception occured
- ****kwargs**: keyworded, variable-length argument list of method

**run_method_and_wait**(*func: Callable*, **args*, ***kwargs*) → Any
execute function and return the function return (thread-blocking)

- **func**: function instance to be executed
- ***args**: variable number of arguments of method
- ****kwargs**: keyworded, variable-length argument list of method

**run_method_and_wait_async**(*func: Callable*, **args*, ***kwargs*) → Awaitable
asynchronously execute function and return the function return

- **func**: function instance to be executed
- ***args**: variable number of arguments of method
- ****kwargs**: keyworded, variable-length argument list of method

**class** hostray.util.worker.**FunctionQueueWorker**

property:

- `pending_count -> int`: return the len of queue

**run_method**(*func: Callable, *args, on_finish: Callable[[Any], None] = None, on_exception: Callable[[Exception], None] = None, **kwargs*) → None

queue the function instance to be executed when worker is free

- **func**: function instance to be executed
- **on_finish**: callback with the argument of function return after function runned
- ***args**: variable number of arguments of method
- **on_exception**: callback with the argument of Exception after function Exception occured
- ****kwargs**: keyworded, variable-length argument list of method

**class** hostray.util.worker.**FunctionLoopWorker**

**run_method**(*func: Callable, *args, on_finish: Callable[[Any], None] = None, on_exception: Callable[[Exception], None] = None, **kwargs*) → None

start and loop the given function instance

- **func**: function instance to be executed
- **on_finish**: callback with the argument of function return after function runned for each time
- ***args**: variable number of arguments of method
- **on_exception**: callback with the argument of Exception after function Exception occured for each time
- ****kwargs**: keyworded, variable-length argument list of method

**stop**()

stop if worker is looping function

**class** hostray.util.worker.**WorkerPool**

property:

- `workers() -> List[FunctionQueueWorker]`

**dispose**() → None

**info**() → Dict

**reserve_worker**() → str

@contextmanager, yield string of identity to reserved worker instance

**run_method**(*func: Callable*, **args*, *identity: str = None*, ***kwargs*) → Any

- **func**: function instance to be executed
- ***args**: variable number of arguments of method
- **identity**: identity string from `reserve_worker`
- ****kwargs**: keyworded, variable-length argument list of method

**broadcast_method**(*func_name: str*, **args*, ***kwargs*) → List[Any]

invoke each worker's function named func_name if it has.

- **func_name**: function name to be invoked
- ***args**: variable number of arguments of method

- ****kwargs**: keyworded, variable-length argument list of method

**class** hostray.util.worker.**AsyncWorkerPool**
inherit from hostray.util.worker.WorkerPool and add asynchronous functions

**reserve_worker_async**() → str
@asynccontextmanager, yield string of identity to reserved worker instance, **hostray** implements a unofficial one since Python 3.6 does not have it.

**run_method_async**(*func: Callable*, **args*, *identity: str = None*, ***kwargs*) → Any

- **func**: function instance to be executed
- ***args**: variable number of arguments of method
- **identity**: identity string from `reserve_worker`
- ****kwargs**: keyworded, variable-length argument list of method

**broadcast_method_async**(*func_name: str*, **args*, ***kwargs*) → List[Any]
asynchronously invoke each worker's **Awaitable** function named func_name if it has.

- **func_name**: function name to be invoked
- ***args**: variable number of arguments of method
- ****kwargs**: keyworded, variable-length argument list of method

### 2.6.2 Orm

**get_declarative_base**(*key: str = 'default'*) → DeclarativeMeta
return key managed `DeclarativeMeta` metaclass

- **key**: key to managed `DeclarativeMeta` metaclass

**get_session_maker**(*db_module: DB_MODULE_NAME*, *declared_entity_base: DeclarativeMeta*, *autoflush: bool = False*, ***kwargs*) → Session
return sqlalchemy.orm.Session class type

- **db_module**: enum `hostray.util.orm.DB_MODULE_NAME`
- **declared_entity_base**: all orm entity class should inherits from `sqlalchemy.ext.declarative.api.DeclarativeMeta` before call this function
- **autoflush**: enable/disable `sqlalchemy.orm.Session autoflush`

**class** hostray.util.orm.**EntityBaseAddon**
define entity helper functions

property:

- `column_type_validations: Dict[str, Any] = {}`
  indicate the column type for validation
- `column_fix: List[str] = []`
  indicate the columns are not allowed to update value
- `client_excluded_columns: List[str] = []`
  indicate the excluded columns for the entity data should be response to client
- `dt_converter = PY_DT_Converter`
  indicate datetime converter from database to json serializable dict

- `identity -> Tuple[Any]`

  return `tuple` of columns as identification

- `primary_key_args -> Dict[str, Any]`

  return key-value dict of primary key columns

- `non_primary_key_args -> Dict[str, Any]`

  return key-value dict of non primary key columns

**primary_keys**() → List[str]
return list of primary key column names

**non_primary_keys**() → List[str]
return list of non primary key column names

**columns**() → List[str]
return list of column names

**get_primary_key_args**(***kwargs*) → Dict[str, Any]
return key-value dict of primary key columns exist in `**kwargs`

- ****kwargs**: keyworded, variable-length argument list of method

**get_non_primary_key_args**(***kwargs*) → Dict[str, Any]
return key-value dict of non primary key columns exist in `**kwargs`

- ****kwargs**: keyworded, variable-length argument list of method

**get_entity_args**(***kwargs*) → Dict[str, Any]
return key-value dict of entity variables exist in `**kwargs`

- ****kwargs**: keyworded, variable-length argument list of method

**get_non_entity_args**(***kwargs*) → Dict[str, Any]
return key-value dict of non entity variables exist in `**kwargs`

- ****kwargs**: keyworded, variable-length argument list of method

**parameter_validation**(*check_fix: bool = True*, ***kwargs*) → None
validate variables in `**kwargs` by specfied `column_type_validations`

- **check_fix**: raise Exception if check_fix is True
- ****kwargs**: keyworded, variable-length argument list of method

**to_client_dict**() → Dict[str, Any]
return dict excludes the keys specfied in `client_excluded_columns`

**to_dict**() → Dict[str, Any]
return dict of entity columns

**equals**(*r: Entity*) → bool
return True if r equals this entity

**class** hostray.util.orm.**OrmDBEntityAccessor**
db access worker owns db session and connection instance based on sqlalchemy.

**set_orm_engine**(*db_module: DB_MODULE_NAME*, *declared_entity_base: DeclarativeMeta*, *autoflush: bool = False*, ***kwargs*) → None
setup parameters to create sqlalchemy.engine.Engine instance

- **db_module**: enum `hostray.util.orm.DB_MODULE_NAME`
- **declared_entity_base**: DeclarativeMeta contains the schema meta of entity class

- **autoflush**: set autoflash refer to sqlalchemy.orm.session.sessionmaker

**close_session**() → None:
release the session and connection.

**Attention:** close_session() should also be called in worker thread

**class** hostray.util.orm.**OrmAccessWorkerPool**
pool of hostray.util.orm.OrmDBEntityAccessor. inherit from hostray.util.worker.AsyncWorkerPool

**enable_orm_log**(*echo: bool = False*) → None
enable/disable sqlalchemy default logger stdout output

**set_session_maker**(*db_module: DB_MODULE_NAME*, *declared_entity_base: DeclarativeMeta*, *autoflush: bool = False*, ***kwargs*) → None
setup parameters to create sqlalchemy.engine.Engine instance

- **db_module**: enum hostray.util.orm.DB_MODULE_NAME
- **declared_entity_base**: DeclarativeMeta contains the schema meta of entity class
- **autoflush**: set autoflash refer to sqlalchemy.orm.session.sessionmaker

**reset_connection**() → None
release all of the workers' session and connection.

**reset_connection_async**() → None
asynchronously release all of the workers' session and connection.

### 2.6.3 Util

**get_class**(*module: str*, **attrs*) → type
return type or function instance of imported module

example:

```
cls = get_class("module", "class / static function", "class static function")
```

**join_to_abs_path**(**paths*) → str
return os.path.join() absolute path in linux format which means replace '\' to '/'

**join_path**(**paths*) → str
return os.path.join() path in linux format which means replace '\' to '/'

**walk_to_file_paths**(*file_or_directory: str*) → List[str]
return a list of absolutely path from the input directory path recursively or file

**size_bytes_to_string**(*f_size: int, units: List[str] = ['bytes', 'KB', 'MB', 'GB', 'TB', 'PB']*) → str
return byte size string in unit

**generate_base64_uid**(*byte_length: int = 32*, *urlsafe: bool = True*) → str
return customized uuid string

**convert_tuple_to_dict**(*t: tuple*, *key_name: str*) → Dict
return customized dict from tuple

example:

```
d = convert_tuple_to_dict((1, 2, 3), 'n'))

# d is {'n_1': 1, 'n_2': 2, 'n_3': 3}
```

**get_host_ip**(*remote_host: str = '8.8.8.8'*, *port: int = 80*) → str
return the host ip, no guarantee to get actual host ip

# Index

## J



## S



## W