hyssop

Release 1.1.7

hsky77

Dec 17, 2022

CONTENTS

hyssop is a python project that defines project hierarchy and creates scalable component architecture which is configurable in yaml format.

prerequest: python 3.6+, pip

Install hyssop with pip: `pip install hyssop`

CHAPTER

ONE

INTRODUCTION

1.1 hyssop

Table of Contents

hyssop is a python project that defines project hierarchy and creates scalable component architecture which is configurable in yaml format.

prerequest: python 3.6+, pip

Install hyssop with pip: `pip install hyssop`

1.1.1 Hierarchy of Project

```
project/
    component/                  # reserved folder contains the components.
        __init__.py
        ...
    unit_test/                  # reserved folder contains the unittest test cases
        __init__.py
        ...
    pack.yaml                   # defines the files to be packed to a compressed file
    requirements.txt            # defines the required pip modules
    project_config.yml          # server configuration
```

1.1.2 Commands

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

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

command:
  {start,pack,test,create,version}
    test                test hyssop library or specfied project directory path
    create              create a project template with specfied project directory path
    pack                pack project with specfied project directory path
    version             print version number to console
```

1.1.3 How to create components?

The **hyssop** projects reserve the directories named **component**, and **unit_test** as the packages of Component classes, and Unit Test cases. It's similar to how python defines and imports packages which contain the files of `__init__.py`.

To get start, create a template project named "hello" by typing command: `python3 -m hyssop create hello`

The project directory should looks like:

```
hello/
    component/
        __init__.py
        hello.py
    unit_test/
        __init__.py
        ut1.py
    project_config.yml
    pack.yml
```

`component/__init__.py` defines the subclasses of `ComponentTypes` that specfies the import pathes. The values are `listes` or `tuples` store `keys in project_config.yml`, `modules`, `classes or functions`.

```
# component/__init__.py

from hyssop.project.component import ComponentTypes

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

In `component/hello.py`, it defines the `HelloComponent` inherits from `Component` class.

```
# component/hello.py

from hyssop.project.component import Component

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

(continues on next page)

(continued from previous page)

```
↪p1)

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

In `project_config.yml`, add the key **'hello'** under component block. That indicates `HelloComponent` should be loaded.

```
# project_config.yml:

name: hyssop Project
debug: False
component:
    hello:
        p1: 'This is p1'
```

1.1.4 How to test components?

`unit_test/__init__.py` defines the subclasses of `UnitTestTypes` that specfies the import path of test cases.

```
# unit_test/__init__.py

from hyssop.unit_test import UnitTestTypes

class UTTypes(UnitTestTypes):
    UT1 = ('ut1', 'ut1', 'UT1TestCase')
```

In `unit_test/ut1.py`, it defines the `UT1TestCase` inherits from `UnitTestCase` class.

```
# unit_test/ut1.py

from hyssop.unit_test import UnitTestCase

class UT1TestCase(UnitTestCase):
    def test(self):
        # implement unit test here...
        import os
        from component import HelloComponentTypes
        from hyssop.project.mixin import ProjectMixin

        project = ProjectMixin()
        project.load_project(os.path.dirname(os.path.dirname(__file__)))
        comp = project.component_manager.get_component(
            HelloComponentTypes.Hello)
        print(comp.hello())
```

Then, type the command `python3 -m hyssop test hello` to run all the test cases which define in `unit_test/__init__.py`.

1.1.5 Configuration Validator

hyssop provides configurations validator verifies the configurations. The following example shows how validator of HelloComponent could be customized.

```
# component/__init__.py

from hyssop.web.component import ComponentTypesConfigComponentValidator
from hyssop.project.config_validator import ConfigContainerMeta, ConfigElementMeta

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

1.2 hyssop-aiodb

Table of Contents

hyssop-aiodb provides the hyssop components that integrates sqlalchemy, aiomysql, aiosqlite to access SQL database asynchronously.

prerequest: python 3.6+, pip

Install hyssop with pip: `pip install hyssop_aiodb`

1.2.1 Usage

- Add the AioDBComponentTypes into `component/__init__.py` of your hyssop project.

```
# component/__init__.py

from hyssop_aiodb.component import AioDBComponentTypes
```

If your component has to depend on AioDBComponent, you could do the following way to be sure the object of AioDBComponent will be initialized at the first cleaned up at the last.

```
# component/__init__.py

from hyssop.project.component import ComponentTypes

class YourProjectComponentTypes(ComponentTypes):
    AioDB = ('aiodb', 'hyssop_aiodb.component.aiodb', 'AioDBComponent')
    YourComponent = ('your_component', 'your_component', 'YourComponent')
```

- Configuration

```
# project_config.yml

component:
    aiodb:
        db_1:
            module: 'sqlite'
            file_name: "db.sqlite3"
        db_2:
            module: 'mysql'
            connections: 1
            host: <your mysql server ip>
            port: <your mysql server port>
            db_name: <your mysql server db>
            user: <user account>
            password: <user password>
```

- Define the orm class of sqlalchemy.

```
from hyssop_aiodb.component.aiodb import get_declarative_base,␣
↪SQLAlchemyEntityMixin, AsyncEntityUW

MODULES = get_declarative_base()

class AccountEntity(MODULES, SQLAlchemyEntityMixin):
    __tablename__ = 'account'

    id = Column(Integer, primary_key=True)
    account = Column(String(40), nullable=False, unique=True, index=True)
    password = Column(String(40), nullable=False)

AccountUW = AsyncEntityUW(AccountEntity)    # provide simple methods to␣
↪access database tables
```

- Access database by AioDBComponent.

```
import asyncio

from hyssop_aiodb.component import AioDBComponentTypes
from hyssop.project.mixin import ProjectMixin

project = ProjectMixin()
project.load_project("path to your project directory")

aiodb_comp = project.component_manager.get_component(AioDBComponentTypes.
↪AioDB)
aiodb_comp.init_db_declarative_base("db_1", MODULES)

async_db = aiodb_comp.get_async_database("db_1")

async def test_async():
    async with async_db.get_connection_proxy() as connection:
        async with connection.get_cursor_proxy() as cursor:
```

(continues on next page)

(continued from previous page)

```
            account_data = {
                'account': "account1",
                'password': '1234',
            }

            account = await AccountUW.load(cursor, **account_data)

            if account is not None:
                account = await AccountUW.add(cursor, **account_data)

            account = await AccountUW.update(cursor, account, password='5678
↪')
            await AccountUW.delete(cursor, [account])

            await cursor.commit()


asyncio.get_event_loop().run_until_complete(test_async())
```

1.3 hyssop-aiohttp

Table of Contents

hyssop-aiohttp is the hyssop extension that bases aiohttp and related packages to implement http interfaces of components.

prerequests: python 3.6+, pip

dependencies: aiohttp, aiohttp-swagger, aiohttp-cors

Install hyssop_aiohttp with pip: `pip install hyssop_aiohttp`

1.3.1 Extended Functionalities

- Add async functions `on_before_server_start` to Component classes which runs after `Component.init()` and before aiohttp server start.
- Add "start" command to run api server by typing `python3 -m hyssop_aiohttp start <path of your project directory>`
- Add reserved directory named "controller" as the package of aiohttp api handlers into hyssop project.

```
project/
    controller/                 # aiohttp api handlers.
        __init__.py
        ...
```

- Changes of configurations with aiohttp related packages:

```
name: hyssop Server
port: 8888
debug: False
doc:
  api_route: <sub route of aiohttp-swagger>
  description: api document description
  version: api document version
  title: api document title
  contact: contact information

cors:
  - origin: localhost
    allow_credentials: True
    expose_headers: '*'
    allow_headers: '*'

controller:
  /sub_route:
    enum: <key of ControllerType to load AioHttpViews>
aiohttp:
  route_decorators:
    - <keys of ControllerType to load aiohttp routes decorated api handlers>
```

  - **port**: Port of aiohttp api server
  - **debug**: Aiohttp api server debug mode
  - **doc**: Settings of aiohttp-swagger
  - **cors**: Settings of aiohttp-cors
  - **controller**: Api Sub_routes
  - **aiohttp: Settings of aiohttp**
    - **route_decorators**: Keys of ControllerType to load handlers into aiohttp routes

1.3.2 Usage

- **Create hyssop-aiohttp project**:
  - Create project named hello by typing `python3 -m hyssop_aiohttp create hello`, the project hierarchy looks like the following block:

```
project/
    controller/                 # reserved folder contains aiohttp api␣
↪handlers.
        __init__.py
        ...
    component/                  # reserved folder contains the components.
        __init__.py
        ...
    unit_test/                  # reserved folder contains the unittest␣
↪test cases
```

(continues on next page)

(continued from previous page)

```
        __init__.py
        ...
    pack.yaml                   # defines the files to be packed to a␣
↪compressed file
    requirements.txt            # defines the required pip modules
    project_config.yml          # server configuration
```

- **Implement controllers**:
  - Add HelloControllerTypes inherits from ControllerType into the file `controller/__init__.py`.

```
from hyssop.project.web import ControllerType

class HelloControllerTypes(ControllerType):
    HelloController = ('hello_world', 'hello', 'hello')
    HelloViewController = ('hello_view', 'hello', 'HelloView')
```

  - Implement the handlers classes or functions.

```
from aiohttp import web

from hyssop_aiohttp import routes, AioHttpView

from component import HelloComponentTypes

class HelloView(AioHttpView):
    async def get(self):
        """
        ---
        tags:
        - hello view
        summary: hello world view get
        description: simple test controller
        produces:
        - text/html
        responses:
            200:
        description: return hello view message
        """
        comp = self.request.app.component_manager.get_
↪component(HelloComponentTypes.Hello)
        return web.Response(text=comp.hello())

@routes.get('/hello')
async def hello(request):
    """
    ---
    tags:
    - hello
    summary: hello world get
    description: simple test controller
    produces:
```

(continues on next page)

(continued from previous page)

```
    - text/html
    responses:
        200:
            description: return hello message
    """
    comp = request.app.component_manager.get_component(HelloComponentTypes.
↪Hello)
    return web.Response(text=comp.hello())
```

- Configurations:

```
# project_config.yml

controller:
  /hello_view:          # handle incoming requests from /hello_view by key
↪'hello_view'
    enum: hello_view
aiohttp:
  route_decorators:
    - 'hello_world'     # load aiohttp routes decorated functions to server
```

- Test the handlers:
  - Run the api server by typing `python3 -m hyssop_aiohttp start hello` in command prompt.
  - Click http://localhost:8888/hello, http://localhost:8888/hello_view

CHAPTER

TWO

CHANGE LOG

- **hyssop**
  - **1.1.5 - Oct. 7, 2021**:
    * Remove package dependency of "coloredlogs".
  - **1.1.4 - Oct. 7, 2021**:
    * Fix bug - logger component: sub folder path of logger file.
  - **1.1.3 - Mar. 21, 2021**:
    * Add parameter "replace_duplicated_code" to Localization import_csv() in util.
    * Fix bug: logger file path is incorrect
  - **1.1.1 - Mar. 06, 2021**:
    * Fix bugs.
  - **1.1.0 - Jan. 10, 2021**:
    * Refactor the project and remove the web framework tornado dependencies.
  - **1.0.2 - Oct. 14, 2020**:
    * Fix bugs.
  - **1.0.0 - Aug. 20, 2020**:
    * Initalize project.
- **hyssop-aiohttp**
  - **0.0.7 - Sep. 18, 2021**:
    * Fix bugs of loading aiohttp route_decorators
  - **0.0.6 - Mar. 27, 2021**:
    * Fix bug: add aiohttp.server to default loggers.
    * Add [aiohttp-cors](https://github.com/aio-libs/aiohttp-cors) applys apis and views
  - **0.0.3 - Mar. 06, 2021**:
    * Fix bug of aio client streaming callback.
  - **0.0.2 - Feb. 15, 2021**:
    * Fix get_argument() of AioHttpRequest with given default value still raise Exception.
  - **0.0.1 - Jan. 10, 2021**:

        * Integrate with [aiohttp](https://docs.aiohttp.org/en/stable/) web framework.

- **hyssop-aiodb**
  - **0.0.7 - Apr. 7, 2021**:
    * Fix bug of value convertion in util
    * Fix bug of AsyncEntityUW update().
    * Add timer to reconnect db connections
  - **0.0.3 - Feb. 15, 2021**:
    * Fix UW class update bug with bool values.
  - **0.0.2 - Feb. 02, 2021**:
    * Add mysql connection proxy pool
    * Fix aiodb mysql cursor retrived inserted row bug
  - **0.0.1 - Jan. 10, 2021**:
    * Re-implement database module with [aiomysql](https://aiomysql.readthedocs.io/en/latest/index.html) & [aiosqlite](https://aiosqlite.omnilib.dev/en/stable/index.html).