clink Documentation

Release 0.1.0

Kevin Leptons

Sep 17, 2017

Contents

HTTP APIs framework

Kevin Leptons <Kevin.leptons@gmail.com>

May, 2017

CHAPTER 1

TABLE OF CONTENTS

Usage

Preface

Before dick in details into Clink, you should know "What is Clink?". After that, you get Clink's advantage and disadvantage then use Clink in effective way. This section help you get it.

Clink is HTTP API library. HTTP is big, large, huge domain and Clink doesn't try to work in all of concepts, it focuses to build HTTP APIs to get, process and save data. If you want something which help you build web front-end application, then go back, Clink is back-end side.

Although Clink focus on HTTP APIs, it also contains many concepts rather than HTTP API such as Application Architecture, Database Model, Authentication, Authorization, and so on. It provides best model to work immediately. That way reduces knowledge which developers must to know in details to help them focus on logic of application. In simple, you can write first logic code lines after you install Clink and don't worry about Routing, Authentication, etc.

Clink likes extensibility than customizable. You can see that if you look into Clink's built-in module, it limits option arguments, it is short, clean and do one thing as best as possible. In case you aren't comfort with it's behaviors, functions, etc, that time you write new module instead try to custom it.

It doesn't meant that you can custom any things in Clink. For example, clink.service.auth.AuthConf defines 4 hours for token life, 30 days for refresh token life in default, but you still can use your values.

Clink written in Python and Python runs on cross platform but Clink run on Unix-like platform. Because Clink is server side library, it need to work as well as possible on server side, when Unix-like is best choice for server side. Most popular Unix-like platform for server side is operating system base on Linux kernel such as Debian, CentOS, etc...

Now, if you agreed with our terms: **HTTP API**, **Backend Programming**, **Extensibility instead of Customizable**, **Unix-like Platform**, then inspect Clink.

Quickstart

Installation

```
# python3 is use as interpreter for python language
# curl use to test api
$ apt-get install python3 wget curl

# pip is python package manager
$ wget https://bootstrap.pypa.io/get-pip.py
$ python get-pip.py

# install clink through pip
$ pip install clink
```

Writting

Listing 1.1: app.py

```
# STEP 1: get clink library
from clink import stamp, mapper, App, AppConf, Controller


# STEP 2: get an WSGI server
from wsgiref.simple_server import make_server


# STEP 3: create application
conf = AppConf('book-api')
app = App(conf)


# STEP 4: define controller
@stamp()
@mapper.path('/book')
class BookCtl(Controller):
    @mapper.get('/item')
    def get_item(self, req, res):
        res.body = {
            'name': 'How to Die',
            'author': 'Death'
        }


# STEP 5: add controller to application
app.add_ctl(BookCtl)


# STEP 6: load components
app.load()


# STEP 7: serve application on WSGI server
address = 'localhost'
port = 8080
print('Prepare API on http://%s:%i/book/item' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python app.py &> /dev/null &
[1] 10297

$ curl localhost:8080/book/item
{"name": "Linux Programming Interface", "author": "Michael Kerrisk"}

$ kill %1
```

Pipe Line

General Model

```
                    handler-0                         handler-n
                       |                                 |
[o]---[req, res]---[req, res]---[req, res]---....---[req, res]------>[x]
 |                                   |                                |
in msg                            handler-i                       out msg
```

in msg is HTTP request message from client. **out msg** is HTTP response message to client. **handler-i** is handler i-th in pipe line. **req** and **res** store data during processing.

In starting of HTTP request message, two object are created **req** and **res**. **req** contains data of incoming message such as remote address, query string, body of messgae, etc. **res** contains data of response message such as header, body of message, etc.

Then application calls handlers from 0 to n to process request. Each handler do something with **[req, res]**. It can reads data from **req**, writes data to **res**.

Final, application send back **res** to client, finish request processing.

Detail Model

```
   [o]------- in msg
    |
[req, res]--- lv 0: receiving handling                              [1]
    |
[req, res]--- lv 1: pre-routing handling                            [n]
    |
[req, res]--- lv 2: routing                                         [1]
    |
[req, res]--- lv 3: pre-main handling                               [n]
    |
[req, res]--- lv 4: main handling                                   [1]
    |
[req, res]--- lv 5: responding handling                             [n]
    |
[req, res]--- lv 6: sending handling                                [1]
    |            |
   [x]           |--- error occurs
 out msg         |
            [req, res]--- lv 7: error handling                      [n]
                 |
            [req, res]--- lv 8: error logging handling              [1]
```

```
           |
        [req, res]--- lv 9: error sending handling              [1]
           |           |
          [x]          |--- error occurs
        out msg        |--- server application handling
                       |
                      [x]
                   undefined
```

Right number is number of handlers. **n** mean that 0 to n. 1 mean that only one.

Level 0: Receiving Handling

This level MUST be perform quickly. First action is create **req**, set request data such as query string, header, body message in stream format and other default values to it. Second action is create **req** and set default values to it.

Level 1: Pre-routing Handling

This level MUST be perform quickly. Normal, this level contains one handler. it performs request logging.

Level 2: Routing

This level MUST be perform quickly. It finds main handler to perform in level 4 from request method, path and content type.

Level 3: Pre-main Handling

This level perform data converting such as convert JSON text to Python dictionary. Be careful, if this level contains too many handlers, it causes performance issue.

Level 4: Main Handling

This is main processing what developer MUST defines. For example, client send POST request with data in JSON format to application:

> {“name”: “A book”, “author”: “Some body”}

It mean that application MUST create new book with above data. Other levels doesn’t do it. After all, it only finds main handling, maps text data into Python dictionary and call main handling with [req, res]. This level MUST validate data, put it to datbase and set response data. Then next levels will send data to client.

Level 5: Responding Handling

This level perform data converting such as convert dictionary to text format. Be careful, if this level contains too many handlers, it causes performance issues.

Level 6: Sending Handling

This level receive input as text stream, or string then send it to client.

Level 7: Error Handling

From level 0 to level 6 if error occurs, this level handles for it. It can changes [req, res] contents. Main target of this level is provides HTTP Status Code and Message to client.

Level 8: Error Logging Handling

Normal, this level contains one handler, it performs log error information.

Level 9: Error Sending Handling

Send **res** to client. Note that **res** was handled, set Status Code and Message correspond with error was occurs.

Exception

After all, if any error occurs during level 7 to level 9, it is handle by server program.

Components

For now, Clink is design follow Pipe Line Model, that mean handlers had format look like below:

```
def handle(req, res, ...):
    pass
```

However, handler need to shares, accesses shared resources such as database connection, configurations, etc. We avoid to do it though function's arguments because each time invoke handler, we MUST detect what is resources handler required, where is it, etc. In simple, it take down performance and take up complicated.

Awesome solution is **Components**. **Components** is concept includes component, auto creation, depedency solving, and get instances. All of that concept be perform by **Injector**. For example:

Listing 1.2: components.py:

```
from clink.com import Component, Injector, stamp
from clink.type import Request, Response


# this is component
@stamp()
class AccountService(Component):
    def create(self, info):
        print('created:', info)


# this other component
@stamp(AccountService)
class AccountCtl(Component):
    def __init__(self, account_service):
        # this is dependency
        self._account_service = account_service

    # this is component's function
    def create(self, req, res):
        info = req.body
        self._account_service.create(info)


# this is injector
i = Injector()


# add component into injector
i.add_com(AccountCtl)


# this is auto creation, dependency solving
i.load()


# this is getting instances
acc_ctl = i.ref(AccountCtl)


# try call function
req = Request()
res = Response()
req.body = {'name': 'Micheal', 'email': 'michael@mars.com'}
acc_ctl.create(req, res)


# get other instace
acc_sv = i.ref(AccountService)
print(acc_sv)
```

Test it

```
$ python components.py
created: {'name': 'Micheal', 'email': 'michael@mars.com'}
<__main__.AccountService object at 0x7f4716069f28>
```

Component

Component is wrapped context. It puts all of references to dependencies and component's functions together. So functions can accesses quickly to dependencies.

@com(AccountService) specify that AccountCtl depends on AccountService.

AccountCtl extend from **Component** to mark that AccountCtl is an component. However, Component doesn't contains any things.

Auto Creation

Because Component is class, it MUST be create to run. It's perform automatically by injector. With abow example, during auto creation, an instance of AccountService will pass into first param of AccountCtl.__init__() without handle from developers.

Depedency Solving

@com decorator specify dependencies. During auto creation, dependencies of component will be recursive analyst, create automatically. That is reason why we don't add **AccountService** component but we still can get an instance of it.

Get Instances

Access instance of component by type.

Application

Application is where components combies together. First application define in **clink.Application** class. To create an HTTP APIs, you must create application.

Every application implement **clink.IWsgi**. It is an WSGI and allow application run on WSGI server.

Creation Steps

Step 1: Get Clink library

clink.App: Built-in application. It's an implementations of Pipe Line and components model, contains built-in components such as JSON conversion, logging, etc. It allows to create an HTTP API from configuration, add other components and ready to run on WSGI server.

clink.AppConf: Application configuration.

clink.stamp: An decorator helps you mark an class become component and specifies it's depedencies.

clink.mapper: An decorator helps you specify routing.

clink.Controller: An abstract class, doesn't contains any things, but it uses to mark an class is component and classify as controller. Then application knows that is controller, add it to routing.

Step 2: Get WSGI server

Here, we use built-in WSGI server in Python. You can chose other servers.

Step 3: Create application

clink.AppConf is an component, however others components may depends on it, so it MUST provide as a argument of **clink.App** constructor.

Step 4: Define components

This step defines components, it MUST be marked by **clink.com**.

We define an controller, it MUST be extend from **clink.Controller** to help application get knowledge about it. **mapper.path('book')** and **mapper.get('item')** specifies that: **BookCtl.get_item** will be call if client perform request **GET /book/item**.

Step 5: Add components to application

Simply, add type of components to application, then application knows “How to create it?. How to bring them togeter?”, etc.

Step 6: Load components

Application creates instance of components, solve depedency of components, etc. It ensure that everythings are ready to runs.

Step 7: Serve application in WSGI server

We create an WSGI server, specify address and port to bind and make it available on network.

Example

Listing 1.3: app.py

```python
# STEP 1: get clink library
from clink import stamp, mapper, App, AppConf, Controller


# STEP 2: get an WSGI server
from wsgiref.simple_server import make_server


# STEP 3: create application
conf = AppConf('book-api')
app = App(conf)


# STEP 4: define controller
@stamp()
@mapper.path('/book')
class BookCtl(Controller):
    @mapper.get('/item')
    def get_item(self, req, res):
        res.body = {
            'name': 'How to Die',
            'author': 'Death'
        }


# STEP 5: add controller to application
app.add_ctl(BookCtl)


# STEP 6: load components
app.load()


# STEP 7: serve application on WSGI server
address = 'localhost'
port = 8080
```

```
print('Prepare API on http://%s:%i/book/item' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python app.py &> /dev/null &
[1] 5946

$ curl localhost:8080/book/item
{"author": "Death", "name": "How to Die"}

$ kill %1
```

Routing

Clink's routing is simple but efficiency. It not allow parameters in path for explicit and performance. Let use query arguments instead of path parameters. For examples:

Parameter form	Argument form
/book/:id	/book/item?id=1243
/book/:id/notes	/book/item/notes?id=1243
/news/:year/:month/:date	/news/archive?year=2017&month=5&date=3

However, arguments isn't introduce here, it's introduce in **Controller** section.

Clink's routing provides routing methods by three factors: **method**, **path**, **content-type**. All of it can be done with **clink.route**.

- clink.route.get(path)
- clink.route.post(path, type=MIME_JSON)
- clink.route.put(path, type=MIME_JSON)
- clink.route.patch(path, type=MIME_JSON)
- clink.route.delete(path)

As you see, GET and DELETE method ignores content-type because it's no meaning. Other methods allow content-type with default value is MIME_JSON. You can access shortcut name for MIME type in **clink.mime.type**.

Example

Listing 1.4: routing.py

```
# STEP 1: get clink library
from clink import stamp, mapper, App, AppConf, Controller


# STEP 2: get an WSGI server
from wsgiref.simple_server import make_server


# STEP 3: create application configuration and application
conf = AppConf('book-api', 'Hell Corporation', '1st, Hell street')
```

```
app = App(conf)


# STEP 4: define component - controllers
@stamp()
@mapper.path('/book')
class BookCtl(Controller):
    @mapper.get('/item')
    def get_item(self, req, res):
        res.body = {
            'name': 'How to Die',
            'author': 'Death'
        }

# ===[BEGIN] ADD MORE ROUTES HERE =============================================
    @mapper.post('/item', 'text/plain')
    def create_item(self, req, res):
        res.body = {'msg': 'created'}

    @mapper.patch('/item')
    def patch_item(self, req, res):
        res.body = {'msg': 'patched'}

    @mapper.put('/item')
    def put_item(self, req, res):
        res.body = {'msg': 'putted'}

    @mapper.delete('/item')
    def delete_item(self, req, res):
        res.body = {'msg': 'deleted'}
# ===[END] ADD MORE ROUTES HERE ===============================================


# STEP 5: add components to application
app.add_ctl(BookCtl)


# STEP 6: load components
app.load()


# STEP 7: serve application on WSGI server
address = 'localhost'
port = 8080
print('Prepare API on http://%s:%i/book/item' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python routing.py &> /dev/null &
[1] 7071

$ curl -X GET localhost:8080/book/item; echo
{"name": "How to Die", "author": "Death"}
```

```
$ curl -X POST -H "Content-Type: text/plain" localhost:8080/book/item; \
  echo
{"msg": "created"}

$ curl -X POST -H "Content-Type: application/json" \
  localhost:8080/book/item; echo
{"status_name": "406 Not Accepted", "message": null, "status": 406}

$ curl localhost:8080/not-exist-path; echo
{"status_name": "404 Not Found", "message": null, "status": 404}

$ kill %1
```

Logging

clink.App provides built-in logging, it writes log data to two log files:

- /var/tmp/<app-name>/request.log
- /var/tmp/<app-name>/error.log

Other logging mechanisms are in developing.

Controller

You know Controller in before sections, now let access request data and modify response. Get back to **app_creation**, remove BookCtl and add RootCtl, we have example below explains how to do it:

Built-in controllers in *clink.ctl*

Example

Listing 1.5: controller.py

```python
# STEP 1: get clink library
from clink import stamp, mapper, App, AppConf, Controller
from clink.error.http import Http401Error, Http404Error


# STEP 2: get an WSGI server
from wsgiref.simple_server import make_server


# STEP 3: create application configuration and application
conf = AppConf('book-api', 'Hell Corporation', '1st, Hell street')
app = App(conf)


# STEP 4: define component - controllers
# ===[BEGIN] REMOVE BOOKCTL AND ADD ROOTCTL ===================================
@stamp()
@mapper.path('/req')
class RootCtl(Controller):
    @mapper.get('/info')
    def get_info(self, req, res):
        res.body = {
            'path': req.path,
            'query_str': req.query_str,
            'content_type': req.content_type,
            'content_length': req.content_length,
            'server_name': req.server_name,
            'server_port': req.server_port,
            'server_protocol': req.server_protocol,
            'remote_addr': req.remote_addr,
            'header': req.header,
            'body': req.body,
        }

    @mapper.get('/no-content')
    def no_content(self, req, res):
        res.status = 204

    @mapper.get('/not-found')
    def not_found(self, req, res):
        raise Http404Error(req, 'Nothing here')

    @mapper.get('/no-auth')
    def no_auth(self, req, res):
        raise Http401Error(req, 'Go back. You are alien')
# ===[END] REMOVE BOOKCTL AND ADD ROOTCTL ====================================


# STEP 5: add components to application
app.add_ctl(RootCtl)


# STEP 6: load components
app.load()


# STEP 7: serve application on WSGI server
address = 'localhost'
port = 8080
print('Prepare API on http://%s:%i/req/info' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python controller.py &> /dev/null &
[1] 6556

$ curl -X GET localhost:8080/req/info; echo
{"query_str": null, "body": null, "header": {"HOST": "localhost:8080",
"ACCEPT": "*/*", "USER_AGENT": "curl/7.38.0"}, "content_type": null,
"remote_addr": "127.0.0.1", "server_name": "localhost",
"server_protocol": "HTTP/1.1", "server_port": 8080, "path": "/req/info",
"content_length": 0}

$ curl -X GET -D - localhost:8080/req/no-content; echo
HTTP/1.0 204 No Content
```

```
Date: Sat, 20 May 2017 14:56:20 GMT
Server: WSGIServer/0.2 CPython/3.4.2
Content-Type: application/json
Content-Length: 0

$ curl -X GET localhost:8080/req/not-found; echo
{"status_name": "404 Not Found", "message": "Nothing here", "status": 404}

$ curl -X GET localhost:8080/req/no-auth; echo
{"status_name": "401 Unauthorized", "message": "Go back. You are alien",
"status": 401}

$ kill %1
```

Service

Service is component. You should use service when you need:

- Carry big logic processing. Entry of main logic proccessing is handlers of controllers, when it too long, you should move it into services.
- Reuse logic processing. May be some handlers must do same of logic processing, that borrow to write again and again, best way is services. That mean code is more easy to read, debug and maintain.
- Access to shared resources. Shared resources is configurations, connection to database, it's some things only exist during runtime. Service is most simple way to to do it because serivce is component and Injector give it's dependencies automatically.

Clink also provides built-in services, check it out in *clink.service*

Example

Now, let create an service, share way to public newsparer or magazine. It accepts type and content of newspaper, magazine then generate string includes information of application with type and content. Two controllers use this service though injector.

Listing 1.6: service.py

```python
# STEP 1: get clink library
from clink import stamp, mapper, App, AppConf, Controller, Service, Version
from clink.mime.type import MIME_PLAINTEXT
from bson import ObjectId


# STEP 2: get an WSGI server
from wsgiref.simple_server import make_server


# STEP 3: create application configuration and application
conf = AppConf('book-api', 'MIT License', Version(0, 1, 0),
               'Hell Corporation', '1st, Hell street')
app = App(conf)


# STEP 4: define components
# ===[BEGIN] DEFINE SERVICES ==============================================
@stamp(AppConf)
class PubService(Service):
    def __init__(self, app_conf):
        self._app_conf = app_conf

    def publish(self, type, content):
        id = ObjectId()
        parts = (
            'Id: ', str(id), '\n',
            'Type: ', type, '\n',
            'Content:\n\n', content, '\n\n',
            self._app_conf.name,
            ' v', str(self._app_conf.version), '\n',
            self._app_conf.org_name, '\n',
            self._app_conf.org_addr,
        )
        return ''.join(parts)


@stamp(PubService)
@mapper.path('/newsparer')
class NewsCtl(Controller):
    def __init__(self, pub_sv):
        self._pub_sv = pub_sv

    @mapper.post('/', MIME_PLAINTEXT)
    def publish(self, req, res):
        content = req.body.decode('utf-8')
        pub = self._pub_sv.publish('NEWSPAPER', content)
        res.body = pub.encode('utf-8')
        res.content_type = MIME_PLAINTEXT


@stamp(PubService)
@mapper.path('/magazine')
class MagazineCtl(Controller):
    def __init__(self, pub_sv):
        self._pub_sv = pub_sv

    @mapper.post('/', MIME_PLAINTEXT)
    def publish(self, req, res):
        content = req.body.decode('utf-8')
        pub = self._pub_sv.publish('MAGAZINE', content)
        res.body = pub.encode('utf-8')
        res.content_type = MIME_PLAINTEXT

# ===[END] DEFINE SERVICES ====================================================


# STEP 5: add components to application
app.add_ctl(NewsCtl)
app.add_ctl(MagazineCtl)

# STEP 6: load components
app.load()

# STEP 7: serve application on WSGI server
address = 'localhost'
```

```
port = 8080
print('Prepare API on http://%s:%i' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python service.py &> /dev/null &
[1] 5864

$ curl -X POST -H "Content-Type: text/plain" \
  -d "This is awesome newsparer" localhost:8080/newsparer; echo
Id: 5920522fe7798916e88e93fd
Type: NEWSPAPER
Content:

This is awesome newsparer

book-api
Hell Corporation
1st, Hell street

$ curl -X POST -H "Content-Type: text/plain" \
  -d "This is awesome magazine" localhost:8080/magazine; echo
Id: 59be3309e779894758b26f86
Type: NEWSPAPER
Content:

This is awesome newsparer

book-api v0.1.0
Hell Corporation
1st, Hell street

$ kill %1
```

Data Conversion

Data conversion divides into two type:

- Request conversion: Depend on request content-type, map raw data into Python object. For example, map JSON string into Python dictionary. In default, Clink convert JSON string and URL Encode to dictionary or list, etc. This handlers must extend from **Lv3Handler** - Pre-Main Handling.
- Response conversion: Depend on response content-type, convert Python object into string, because send handler accept string as body of response message. In default, Clink convert dictionary, list, etc to JSON string. This handlers must extend from **Lv5Handler** - Responding Handling

Now, let create data conversion handlers to know “How it work?”. We create two handlers:

- **ReqTextHandler** converts ‘text/plain’ to list of words.
- **ResTextHandler** joins list of words to string

And we create main handler to converts list of words to uppercase: **TextCtl.process_text**

Example

Listing 1.7: data_conversion.py

```python
# STEP 1: get clink library
from clink import stamp, mapper, App, AppConf, Controller
from clink.iface import ILv3Handler, ILv5Handler
from clink.mime.type import MIME_PLAINTEXT


# STEP 2: get an WSGI server
from wsgiref.simple_server import make_server


# STEP 3: create application configuration and application
conf = AppConf('book-api', 'Hell Corporation', '1st, Hell street')
app = App(conf)


# STEP 4: define components
# ===[BEGIN] ADD DATA CONVERSION HANDLERS ====================================
@stamp()
class ReqTextHandler(ILv3Handler):
    def handle(self, req, res):
        if req.content_type != MIME_PLAINTEXT:
            return
        if req.body is None:
            req.body = []
        else:
            req.body = req.body.decode('utf-8').split(' ')


@stamp()
class ResTextHandler(ILv5Handler):
    def handle(self, req, res):
        if res.content_type != MIME_PLAINTEXT:
            return
        if res.body is None:
            res.body = ''
        else:
            res.body = ' '.join(res.body).encode('utf-8')


@stamp()
@mapper.path('/text')
class TextCtl(Controller):
    @mapper.post('/', MIME_PLAINTEXT)
    def process_text(self, req, res):
        res.body = [w.upper() for w in req.body]
        res.content_type = MIME_PLAINTEXT
# ===[END] ADD DATA CONVERSION HANDLERS ======================================


# STEP 5: add components to application
app.add_handler(ReqTextHandler)
app.add_handler(ResTextHandler)
app.add_ctl(TextCtl)
```

```
# STEP 6: load components
app.load()

# STEP 7: serve application on WSGI server
address = 'localhost'
port = 8080
print('Prepare API on http://%s:%i/text' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python data_conversion.py &> /dev/null &
[1] 6456

$ curl -X POST -H "Content-Type: text/plain" \
  -d "my name is kevin" localhost:8080/text; echo
MY NAME IS KEVIN

$ kill %1
```

Data Validation

Data validation use many CPU resource, any mistake on data validation will down system into hole. Clink provides a mechanism to controll all of this processing in *clink.dflow*. In simple, put one data validation in service.

Example

Listing 1.8: data_validation.py

```
# STEP 1: get clink library
from clink import stamp, mapper, App, AppConf, Controller, Service
from clink.dflow import verify


# STEP 2: get an WSGI server
from wsgiref.simple_server import make_server


# STEP 3: create application configuration and application
conf = AppConf('book-api', 'Hell Corporation', '1st, Hell street')
app = App(conf)


# STEP 4: define component - service, controllers
# ===[BEGIN] TRY DATA VALIDATION =============================================
POST_BOOK_SCHEMA = {
    'type': 'object',
    'additionalProperties': False,
    'required': ['name', 'author'],
    'properties': {
        'name': {'type': 'string', 'pattern': '^[a-zA-Z0-9 ]{2,32}$'},
        'author': {'type': 'string', 'pattern': '^[a-zA-Z0-9 ]{2,16}$'}
    }
}


@stamp()
class BookSv(Service):
    @verify(None, POST_BOOK_SCHEMA)
    def create_one(self, book):
        # if data is invalid, clink.dflow.FormatError will be raise
        # then application catch and handle it

        pass


@stamp(BookSv)
@mapper.path('/book')
class BookCtl(Controller):
    def __init__(self, book_sv):
        self._book_sv = book_sv

    @mapper.post('/item')
    def create_one_book(self, req, res):
        self._book_sv.create_one(req.body)
        res.body = {'message': 'created'}
# ===[END] TRY DATA VALIDATION ===============================================


# STEP 5: add components to application
app.add_ctl(BookCtl)


# STEP 6: load components
app.load()


# STEP 7: serve application on WSGI server
address = 'localhost'
port = 8080
print('Prepare API on http://%s:%i/book/item' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python data_validation.py &> /dev/null &
[1] 9977

$ curl -X POST -H "Content-Type: application/json" \
-d '{"name": "Dead Note"}' localhost:8080/book/item; echo
{"message": {"value": "{'name': 'Dead Note'}", "name": "book",
"schema": {"type": "object", "properties": {"name": {"type":
"string", "pattern": "^[a-zA-Z0-9 ]{2,32}$"}, "author": {"type":
"string", "pattern": "^[a-zA-Z0-9 ]{2,16}$"}}, "required":
["name", "author"], "additionalProperties": false}},
"status_name": "400 Bad Request", "status": 400}
```

```
$ curl -X POST -H "Content-Type: application/json" \
-d '{"name": "Dead Note", "author": "Death"}' \
localhost:8080/book/item; echo
{"message": "created"}

$ kill %1
```

APIs

clink

App

class clink.**App**(*conf*)

Application brings APIs to HTTP

Parameters **conf** (*clink.AppConf*) –

add_handler(*handler_type*)

Add handler to application

Parameters **handler_type** (*class*) –

add_prim(**args*)

add_ctl(*ctl_type*)

Add controller to application

Parameters **ctl** (*class*) –

add_ctls(*module*)

Search controllers in module and add them to application

load()

Creeate instance of all of components, put it to ready state

__call__(*wsgi_env*, *wsgi_send*)

Implemention of WSGI. That mean you can call instance of App by WSGI server to make application is available on network.

Parameters

- **wsgi_env** (*dict*) –
- **wsgi_send** (*function*) –

Example

Listing 1.9: app.py

```
from clink import App, AppConf, stamp, mapper, Controller
from wsgiref.simple_server import make_server

conf = AppConf('book-api', 'Hell Corporation', '1st, Hell street')
app = App(conf)
```

```
@stamp()
@mapper.path('book')
class BookCtl(Controller):
    @mapper.get('item')
    def get_item(self, req, res):
        res.body = {
            'name': 'How to Die',
            'author': 'Death'
        }


app.add_com(BookCtl)
app.load()


address = 'localhost'
port = 8080
print('Prepare API on http://%s:%i/book/item' % (address, port))
httpd = make_server(address, port, app)
httpd.serve_forever()
```

Testing

```
$ python app.py
[1] 5940

$ curl localhost:8080/book/item
{"name": "How to Die", "author": "Death"}

$ kill %1
```

Types

class `clink.type.request.`**`Request`**

Carries data of request from client to application

`method` = None

`path` = None

`query_str` = None

`content_type` = None

`content_length` = None

`server_name` = None

`server_port` = None

`server_protocol` = None

`remote_addr` = None

`remote_port` = None

`header` = {}

body = None

args = {}

class `clink.type.response.`**`Response`**

Carries data of response message from application to client

status = None

content_type = None

header = {}

body = None

class `clink.type.conf.`**`AppConf`**(*name*, *license='?'*, *version=<clink.type.version.Version object>*, *org_name='?'*, *org_addr='?'*)

Essential information about application

Parameters

- **name** (*str*) –
- **license** (*str*) –
- **version** (Version) –
- **org_name** (*str*) –
- **org_addr** (*str*) –

class `clink.type.conf.`**`AuthConf`**(*root_pwd*, *root_email*, *root_email_pwd*, *root_email_server*, *root_email_server_port*, *jwt_key*, *token_time=14400*, *rtoken_time=2592000*)

Authorization configuration

Parameters

- **root_pwd** (*str*) –
- **root_email** (*str*) –
- **root_email_pwd** (*str*) –
- **root_email_server** (*str*) –
- **root_email_server_port** (*int*) –
- **jwt_key** (*str*) –
- **token_time** (*int*) –
- **rtoken_time** (*int*) –

class `clink.type.version.`**`Version`**(*major*, *minor*, *rev*)

Strict version

Parameters

- **major** (*int*) –
- **minor** (*int*) –
- **rev** (*int*) –

Errors

clink.error.http.**code_to_str**(*code*)

Convert HTTP error code to string

Parameters **code** (*int*) –

Raise HttpStatusError:

exception clink.error.http.**HttpStatusError**(*code*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**HttpError**(*status*, *req*, *msg=None*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http400Error**(*req*, *msg=None*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http401Error**(*req*, *msg=None*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http402Error**(*req*, *msg=None*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http403Error**(*req*, *msg=None*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http404Error**(*req*, *msg=None*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http405Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http406Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http407Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http408Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http409Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http410Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http411Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http412Error`**(*req*, *msg=None*)

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.error.http.`**`Http413Error`**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http414Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http415Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http416Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http417Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http500Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http501Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http502Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http503Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http504Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**Http505Error**(*req*, *msg=None*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.http.**HttpArgumentError**(*arg_str*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.error.wsgi.**WsgiResBodyError**(*body*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

clink.com

Injector

class clink.com.**Injector**
Object management

add_prim(*com_obj*)
Put an instance of component under management. Instance MUST be primitive

Parameters **com_obj** (*clink.com.Component*) –

Raises

- *ComExistError* –
- *ComDepedencyError* –

add_com(*com_type*)
Put an component under management

Parameters **com_type** (*class*) –

Raises *ComExistError* –

add_coms(*com_types*)
Put components under management

Parameters **com_types** (*list[class]*) –

load()
Start to create instances of all of components

ref(*com_type*)
Return reference to component

Parameters **com_type** (*class*) –

Return type object

sub_ref(*com_type*)
Return references to components which extends from com_type

Parameters **com_type** (*class*) –

Return type list[object]

Example

Listing 1.10: injector.py

```
from clink.com import Injector, Component, Primitive, stamp


@stamp()
class Engine(Primitive):
    def __init__(self, name):
        self.name = name


@stamp(Engine)
class Car(Component):
    def __init__(self, engine):
        self.engine = engine

    def info(self):
        return 'A car with %s engine' % self.engine.name


@stamp(Car)
class Human(Component):
    def __init__(self, car):
        self.car = car

    def drive(self):
        print('I drive %s' % self.car.info())


engine = Engine('Turbo v8')
injector = Injector()

injector.add_prim(engine)
injector.add_com(Human)
injector.load()

car = injector.ref(Car)
print(car.info())
```

```
human = injector.ref(Human)
human.drive()
```

Testing

```
$ python injector.py
A car with Turbo v8 engine
I drive A car with Turbo v8 engine
```

find()

`clink.com.find.`**`find`**`(`*module*, *com_type=<class 'clink.com.type.Component'>*`)`

Find class which extends from com_type in module

Parameters

- **`module`** (*`module`*) –
- **`com_type`** (*`class`*) –

Example

Listing 1.11: com_find.py

```
import clink
from clink.com import find
from clink import Service


coms = find(clink.service, Service)
for com in coms:
    print(com.__name__)
```

Testing

```
$ python com_find.py
AccService
AuthDbService
OAuthService
SmtpService
```

stamp()

`clink.com.label.`**`stamp`**`(`**args*`)`

Mark an class become component with depedencies

Parameters **`args`** (*`tuple`*) –

Example

Listing 1.12: com_stamp.py

```
from clink.com import stamp, Injector, Component


@stamp()
class Pump(Component):
    def run(self):
        print('Pump was started')


@stamp(Pump)
class Nozzle(Component):
    def __init__(self, pump):
        self._pump = pump

    def spray(self):
        self._pump.run()
        print('Nozzle is spraying')


injector = Injector()
injector.add_com(Nozzle)
injector.load()

nozzle = injector.ref(Nozzle)
nozzle.spray()
```

Testing

```
$ python com_stamp.py
Pump was started
Nozzle is spraying
```

write_stamp(), read_stamp()

`clink.com.label.`**`write_stamp`**(*target*, *key*, *value*)

Write metadata into target

Parameters

- **target** (*object*) –
- **key** (*str*) –
- **value** (*object*) –

`clink.com.label.`**`read_stamp`**(*target*, *key*)

Read metadata from target

Parameters

- **target** (*object*) –
- **key** (*str*) –

Example

Listing 1.13: com_rw_stamp.py

```python
from clink.com import write_stamp, read_stamp


class BrokenCar():
    def boot(self):
        print('Oop!, something went wrong.')


write_stamp(BrokenCar, 'note', 'This cars can not boot. It must be fixs')
note = read_stamp(BrokenCar, 'note')


print('class BrokenCar has a note:', note)
```

Testing

```
$ python com_rw_stamp.py
class BrokenCar has a note: This cars can not boot. It must be fixs
```

Types

class `clink.com.type.`**`Component`**

It doesn't contains anythings. It's use to mark an class become component

class `clink.com.type.`**`Primitive`**

Primitive components, it isn't create directly by injector. It must be add to injector by add_prim(). If other components depend on it, injector look up for it's instance, if not founds instance, raise error instead of create new one

Errors

exception `clink.com.error.`**`PrimError`**(*prim_type*, *msg*)

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.com.error.`**`ComDepedencyError`**(*com_type*, *msg*)

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.com.error.`**`InjectorLoadingError`**

`args`

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.com.error.**ComTypeError**(*com_type*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.com.error.**ComAttrError**(*com_type*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.com.error.**CircleComError**(*com_types*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.com.error.**ComExistError**(*com_type*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.com.error.**ComCreationError**(*com_type*, *args*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

clink.iface

class clink.iface.pipe.**IPipeHandler**

class clink.iface.pipe.**ILv0Handler**
Receive handling

handle(*req*, *res*, *env*)

Parameters

- **req** (Request) –
- **res** (Response) –
- **env** (*dict*) –

class clink.iface.pipe.**ILv1Handler**
Pre-Routing handling

handle(*req*, *res*)

Parameters

- **req** (Request) –
- **res** (Response) –

class clink.iface.pipe.**ILv2Handler**

Routing

handle(*req*)

Parameters **req** (Request) –

Return type function

class clink.iface.pipe.**ILv3Handler**

Pre-Main handling

handle(*req*, *res*)

Parameters

- **req** (Request) –
- **res** (Response) –

class clink.iface.pipe.**ILv4Handler**

Main handling. It must be function, but we can't define interface for functions. Here are symbolic interface.

class clink.iface.pipe.**ILv5Handler**

Responding handling

handle(*req*, *res*)

Parameters

- **req** (Request) –
- **res** (Response) –

class clink.iface.pipe.**ILv6Handler**

Sending handling

handle(*req*, *res*, *wsgi_send*)

Parameters

- **req** (Request) –
- **res** (Response) –
- **wsgi_send** (*function*) –

class clink.iface.pipe.**ILv7Handler**

Error handling

handle(*req*, *res*, *e*)

Parameters

- **req** (Request) –
- **res** (Response) –
- **e** (*Exception*) –

class clink.iface.pipe.**ILv8Handler**

Error logging handling

handle(*req*, *res*, *e*)

Parameters

- **req** (Request) –
- **res** (Response) –
- **e** (*Exception*) –

class clink.iface.pipe.**ILv9Handler**
Sending error handling

handle(*req*, *res*, *wsgi_send*)

Parameters

- **req** (Request) –
- **res** (Response) –
- **wsgi_send** (*function*) –

class clink.iface.wsgi.**IWsgi**

__call__(*wsgi_env*, *wsgi_send*)
WSGI inteface

Parameters

- **wsgi_env** (*dict*) –
- **wsgi_send** (*function*) –

clink.routing

Router

class clink.routing.**Router**(*routes=[]*)
Store and find routes

Parameters **routes** (*list<Router>*) –

add_ctl(*ctl*)
Collect routing information from controller

Parameters **ctl** (*object*) –

add_route(*route*)
Put route into map

Parameters **route** (Route) –

Raises ***RouteExistError*** –

add_routes(*routes*)
Put routes into map

Parameters **routes** (*list*[Route]) –

handle(*req*)
Find handle which match with request

Parameters **req** (Request) –

Return type function

Raises

- ***Http400Error*** –
- ***Http404Error*** –
- ***Http405Error*** –

Example

Listing 1.14: router.py

```
from clink import Router, Route, Request


def book_handle(name):
    print('This is %s book' % name)


def hat_handle(name):
    print('This is %s hat' % name)


router = Router()

book_route = Route('/api/book', 'get', 'text/plain', book_handle)
hat_route = Route('/api/hat', 'get', 'text/plain', hat_handle)

router.add_route(book_route)
router.add_route(hat_route)

req = Request()
req.method = 'get'
req.content_type = 'text/plain'

req.path = '/api/book'
req_handle = router.handle(req)
req_handle('story')

req.path = '/api/hat'
req_handle = router.handle(req)
req_handle('baseball')
```

Testing

```
$ python router.py
This is story book
This is baseball hat
```

mapper

`clink.routing.mapper.`**`path`**(*path*)

Specify a path to controller

Parameters **path** (*str*) –

`clink.routing.mapper.`**`map`**(*path*, *method*, *req_type*)
Specify a controller's method

Parameters

- **path** (*str*) –
- **method** (*str*) –
- **req_type** (*str*) –

`clink.routing.mapper.`**`get`**(*path*)
Specify a GET controller's method

Parameters **path** (*str*) –

`clink.routing.mapper.`**`post`**(*path*, *req_type='application/json'*)
Map a POST controller's method

Parameters

- **path** (*str*) –
- **req_type** (*str*) –

`clink.routing.mapper.`**`put`**(*path*, *req_type='application/json'*)
Map a PUT controller's method

Parameters

- **path** (*str*) –
- **req_type** (*str*) –

`clink.routing.mapper.`**`patch`**(*path*, *req_type='application/json'*)
Map a PATCH controller's method

Parameters

- **path** (*str*) –
- **req_type** (*str*) –

`clink.routing.mapper.`**`delete`**(*path*, *req_type='application/json'*)
Map a DELETE controller's method

Parameters **path** (*str*) –

Example

See *Example*

Types

class `clink.routing.route.`**`Route`**(*path*, *method*, *req_type*, *handle*)
Specify a route in a map

Parameters

- **path** (*str*) –
- **method** (*str*) –

- **req_type** (*str*) –
- **handle** (*function*) –

class `clink.routing.type.`**`CtlMethod`**(*path*, *method*, *req_type*)

Specify controller method

Parameters

- **path** (*str*) –
- **method** (*clink.type.HttpMethod*) –
- **req_type** (*str*) –

Errors

exception `clink.routing.error.`**`CtlSpecError`**(*ctl*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.routing.error.`**`PathNotFoundError`**(*path*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.routing.error.`**`HandleNotFoundError`**(*method*, *content_type*, *path*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.routing.error.`**`RouteExistError`**(*route*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.routing.error.`**`RouteMethodError`**(*method*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.routing.error.`**`RoutePathError`**(*path*, *regex*)

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.routing.error.`**`RouteHandleError`**(*handle*)

> **`args`**
>
> **`with_traceback`**()
>
> > Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

clink.handler

class `clink.handler.err_http.`**`ErrorHttpHandler`**

> Catch HTTP error and make response message correspond with error
>
> **`handle`**(*req*, *res*, *e*)

class `clink.handler.err_log.`**`ErrorLogHandler`**(*app_conf*)

> Catch error, write information to file at /var/tmp/<app-name>/error.log
>
> **`handle`**(*req*, *res*, *e*)

class `clink.handler.recv.`**`RecvHandler`**

> Receive HTTP message, construct an isinstance of Request
>
> **Parameters**
>
> - **req** (*clink.Request*) –
> - **res** (*clink.Response*) –
> - **env** (*dict*) –
>
> **`handle`**(*req*, *res*, *env*)

class `clink.handler.req_json.`**`ReqJsonHandler`**

> Map JSON string from body message to Python object
>
> **`handle`**(*req*, *res*)

class `clink.handler.req_log.`**`ReqLogHandler`**(*app_conf*)

> Catch HTTP request, write information of request to file in /var/tmp/<app-name>/request.log
>
> **`handle`**(*req*, *res*)

class `clink.handler.req_url_encode.`**`ReqUrlEncodeHandler`**

> Map URL Encode string to Python object
>
> **`handle`**(*req*, *res*)

class `clink.handler.res_cors.`**`ResCorsHandler`**

> Inform client to know that server allow CORS
>
> **`handle`**(*req*, *res*)

class `clink.handler.res_json.`**`ResJsonHandler`**

> Serialize Python object to JSON string
>
> **`handle`**(*req*, *res*)

class `clink.handler.send.`**`SendHandler`**

> Send response message to client
>
> **`handle`**(*req*, *res*, *wsgi_send*)

class `clink.handler.dflow_err.`**`DflowErrorHandler`**

> Catch Data Flow error and make response message correspond with error

handle (*req*, *res*, *e*)

clink.dflow

verify()

clink.dflow.verify.**verify** (**schemas*)

Decorator, verify formating of input arguments

Parameters **schemas** (*tuple[dict]*) –

Return type function

Example

Listing 1.15: dflow_verify.py

```
from clink.dflow import verify, FormatError


road_schema = {'type': 'string', 'pattern': '^[a-zA-Z0-9 ]{6,32}$'}


@verify(road_schema)
def run_car(road):
        print('Car is running on the %s' % road)


class Car():
    @verify(None, road_schema)
    def run(self, road):
        print('Car is running on the %s' % road)


try:
    car = Car()
    car.run('1st Hell Street')
    run_car('1st Hell Street @@@')
except FormatError as e:
    print('Error: ', e)
```

Testing

```
$ python dflow_verify.py
Car is running on the 1st Hell Street
Error:  name=, value=1st Hell Street @@@, schema={'pattern':
'^[a-zA-Z0-9 ]{6,32}$', 'type': 'string'}
```

Errors

exception clink.dflow.error.**ExistError** (*indexes*)

Specify for indexes is early exist

Parameters **indexes** (*dict*) –

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.dflow.error.**NonExistError**(*indexes*)
Specify for indexes doesn't exist

Parameters **indexes** (*dict*) –

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.dflow.error.**FormatError**(*name*, *value*, *schema*)
Specify for data formating is invalid

Parameters

- **name** (*str*) –
- **value** (*object*) –
- **req** (*object*) –

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception clink.dflow.error.**ExpiredError**(*indexes*)
Specify expirision error

Parameters **indexes** (*dict*) –

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

clink.mime

clink.service

MongoSv

class clink.service.**MongoSv**(*conf*)
MongoDB interface in application layer

Parameters **conf** (MongoConf) –

use_docspec(*doc_spec*)
Put a document's specification under management

Parameters **doc_spec** (MongoDocSpec) –

use_docspecs(*doc_specs*)
Put document's specifications under management

Parameters **doc_specs** (*list* [MongoDocSpec]) –

doc (*name*)

Return document object

Parameters **name** (*str*) –

Return type pymongo.collection.Collection

Raises ***DocumentNotExist*** –

docs (**args*)

Return document objects

Parameters **args** (*tuple[str]*) –

Return type tuple[pymongo.collection.Collection]

Raises ***DocumentNotExist*** –

close ()

Close connection to server

clear ()

Clear all of data in database

Example

Listing 1.16: mongo_sv.py

```python
from clink.service import MongoSv, MongoConf, MongoDocSpec
from pymongo import IndexModel, ASCENDING


conf = MongoConf('mongodb://localhost', 'book-db')
mongo_sv = MongoSv(conf)

book_doc_name = 'book'
book_doc_index_1 = IndexModel([('name', ASCENDING)], unique=True)
book_doc_spec = MongoDocSpec(book_doc_name, [book_doc_index_1])

mongo_sv.use_docspec(book_doc_spec)

book_doc = mongo_sv.doc(book_doc_name)
book_doc.insert_one({'name': 'How to Die', 'author': 'Death'})

book = book_doc.find_one({'name': 'How to Die'})
print(book)
```

Testing

```
$ python mong_sv.py
{'author': 'Death', '_id': ObjectId('592403e2e7798911eb606720'),
'name': 'How to Die'}
```

SmtpSv

class `clink.service.`**`SmtpSv`**(*app_conf*, *auth_conf*)

Send mail message on SMTP

Parameters

- **`app_conf`** (`AppConf`) –
- **`auth_conf`** (`AuthConf`) –

`send`(*dest_email*, *subject*, *text_body*)

Send plain text message

Parameters

- **`dest_email`** (*`str`*) –
- **`subject`** (*`str`*) –
- **`text_body`** (*`str`*) –

Example

Listing 1.17: smtp_sv.py

```
from clink.service import SmtpSv, AuthConf
from clink import AppConf


app_conf = AppConf('Awesome Application')

sender_email = 'you@email.com'
sender_email_password = 'secret-words'
email_server_address = 'smtp.email.com'
email_server_port = 587
auth_conf = AuthConf(
    'root-pwd', sender_email, sender_email_password,
    email_server_address, email_server_port, 'jwt-key'
)

smtp = SmtpSv(app_conf, auth_conf)

receiver = 'someone@email.com'
subject = 'Let be friends'
content = 'Hello there'
smtp.send(receiver, subject, content)
```

Testing

```
# replace your email, password and server then run
$ python smtp_sv.py
```

AuthDbSv

class `clink.service.`**`AuthDbSv`**`(`*mongo_sv*`)`

Ensure that database is compative with both AccSv and OAuthSv

Parameters **`mongo_sv`** (`MongoSv`) –

`acc_doc``()`

Return account collection

Return type pymongo.collection.Collection

`grp_doc``()`

Return group collection

Return type pymongo.collection.Collection

`rpwd_doc``()`

Return refresh token collection

Return type pymongo.collection.Collection

`acctmp_doc``()`

Return temporary account collection

Return type pymongo.collection.Collection

Example

Listing 1.18: authdb_sv.py

```
from clink.service import AuthDbSv, MongoSv, MongoConf


mongo_conf = MongoConf('mongodb://localhost', 'book-db')
mongo_sv = MongoSv(mongo_conf)

authdb_sv = AuthDbSv(mongo_sv)
print(authdb_sv.acc_doc())
print(authdb_sv.grp_doc())
print(authdb_sv.rpwd_doc())
print(authdb_sv.acctmp_doc())
```

Testing

```
$ python authdb_sv.py
Collection(Database(MongoClient(host=['localhost:27017'],...
Collection(Database(MongoClient(host=['localhost:27017'],...
Collection(Database(MongoClient(host=['localhost:27017'],...
Collection(Database(MongoClient(host=['localhost:27017'],...
```

AccSv

class `clink.service.`**`AccSv`**`(`*authdb_sv*, *auth_conf*`)`

Manage accounts and related concepts

Parameters

- **`authdb_sv`** (`AuthDbSv`) –
- **`auth_conf`** (`AuthConf`) –

`mk_acc`(*name*, *password*, *email*, *phone=None*)

Create new account

Parameters

- **`name`** (*`str`*) –
- **`password`** (*`str`*) –
- **`email`** (*`str`*) –
- **`phone`** (*`str`*) –

Return type bson.objectid.ObjectId

Raises **`TypeError`** –

`mk_reg_code`(*name*, *password*, *email*, *phone=None*)

Create a registration code. Use returned code with cf_reg_code() to create account

Parameters

- **`name`** (*`str`*) –
- **`password`** (*`str`*) –
- **`email`** (*`str`*) –
- **`phone`** (*`str`*) –

Return type *ConfirmCodeSpec*

Raises

- **`TypeError`** –
- ***`ExistError`*** –

`cf_reg_code`(*code*)

Use registration code to create account

Parameters **`code`** (*`str`*) –

Return type dict

Raises

- ***`ExistError`*** –
- ***`ExpiredError`*** –

`find_id`(*id*)

Find account by identity

Parameters **`id`** (*`bson.objectid.ObjectId`*) –

Return type dict

`find_name`(*name*)

Find account by name

Parameters **`name`** (*`str`*) –

Return type dict

Raises `TypeError` –

`find_email`(*email*)
Find account by email

Parameters **`email`** (*str*) –

Return type dict

Raises `TypeError` –

`find_phone`(*phone*)
Find account by phone number

Parameters **`phone`** (*str*) –

Return type dict

Raises `TypeError` –

`find_pwd`(*name*, *pwd*)
Find account by name and password

Parameters

- **`name`** (*str*) –
- **`pwd`** (*str*) –

Return type dict

Raises `TypeError` –

`rm_acc`(*id*)
Remove account by identity

Parameters **`id`** (*bson.objectid.ObjectId*) –

`ch_pwd`(*id*, *new_pwd*)
Change password of account by identity

Parameters

- **`id`** (*bson.objectid.ObjectId*) –
- **`new_pwd`** (*str*) –

Raises `TypeError` –

`mk_rpwd_code`(*email*)
Create reset password code from email. Use returned code with cf_rpwd_code() to reset to new password

Parameters **`email`** (*str*) –

Return type *ConfirmCodeSpec*

Raises `TypeError` –

`cf_rpwd_code`(*code*, *new_pwd*)
Reset password from code

Parameters

- **`code`** (*str*) –
- **`new_pwd`** (*str*) –

Return type bson.objectid.ObjectId

Raises `TypeError` –

`mk_group`(*group_name*)
Create new account group

Parameters **`group_name`** (*`str`*) –

`rm_group`(*group_name*)
Remove account group

Parameters **`group_name`** (*`str`*) –

`add_to_group`(*acc_id*, *group_name*)
Put an account into group

Parameters

- **`acc_id`** (*`bson.objectid.ObjectId`*) –
- **`group_name`** (*`str`*) –

`del_fm_group`(*acc_id*, *group_name*)
Remove an account from group

Parameters

- **`acc_id`** (*`bson.objectid.ObjectId`*) –
- **`group_name`** (*`str`*) –

Example

Listing 1.19: acc_sv.py

```python
from clink.service import AccSv, AuthDbSv, MongoSv, MongoConf
from clink import AuthConf


mongo_conf = MongoConf('mongodb://localhost', 'book-db')
mongo_sv = MongoSv(mongo_conf)

authdb_sv = AuthDbSv(mongo_sv)

root_pwd = 'root-pwd'
root_email = 'root@mail.com'
root_email_pwd = 'root-email-pwd'
root_email_server = 'smtp.email.com:587'
auth_conf = AuthConf(
    root_pwd, root_email, root_email_pwd, root_email_server,
    'jwt-key'
)

acc_sv = AccSv(authdb_sv, auth_conf)

root_acc = acc_sv.find_pwd('root', root_pwd)
print(root_acc['name'], root_acc['email'], root_acc['last_action'])
```

Testing

```
$ python acc_sv.py
root root@mail.com REGISTERED
```

OAuthSv

class `clink.service.`**`OAuthSv`**(*authdb_sv*, *acc_sv*, *auth_conf*)

Limited OAuth2 implementation

Parameters

- **`authdb_sv`** (`AuthDbSv`) –
- **`acc_sv`** (`AccSv`) –
- **`auth_conf`** (`AuthConf`) –

`mktoken_pwd`(*name*, *password*)

Create an token from account name and password

Parameters

- **`name`** (*`str`*) –
- **`password`** (*`str`*) –

Return type dict

Raises ***`NonExistError`*** –

`mktoken_rtoken`(*rtoken*)

Create an token from refresh token

Parameters **`rtoken`** (*`str`*) –

Return type dict

Raises

- **`TypeError`** –
- ***`ExpiredError`*** –

`authen`(*access_token*)

Authenticate access token

Parameters **`access_token`** (*`str`*) –

Return type bson.objectid.ObjectId

Raises

- ***`FormatError`*** –
- ***`ExpiredError`*** –

`authen_req`(*req*)

Authenticate HTTP request

Parameters **`req`** (`Request`) –

Rtype mongo.objectid.ObjectId

Raises ***`Http400Error`*** –

Notes

It doesn't support all of OAuth2 specification, here are supported features:

- RFC 6749, section 4.3 - Resource Owner Password Credentials Grant
- RFC 6749, section 6 - Refreshing an Access Token
- RFC 7519 - JSON Web Token

Other specifications isn't supported because it's complicated without browsers. For example, mobile device need polls auth server to gets token instead of gets it from auth provider directly.

Use this limited OAuth specification, you can't perform external login with other OAuth Providers, you can only use name-password to get token and refresh that token. However, it work in simply on all of platform.

It also ignore authorization 'scope'. Authorization is perform by query database, not by information in access_token.

Example

Listing 1.20: oauth_sv.py

```python
from clink.service import OAuthSv, MongoSv, AuthDbSv, AccSv, \
                          AuthConf, MongoConf


mongo_conf = MongoConf('mongodb://localhost', 'book-db')
mongo_sv = MongoSv(mongo_conf)

root_pwd = 'root-pwd'
root_email = 'root@email.com'
root_email_pwd = 'root-email-pwd'
root_email_server = 'smtp.email.com'
root_email_server_port = 587
auth_conf = AuthConf(
    root_pwd, root_email, root_email_pwd, root_email_server,
    root_email_server_port, 'jwt-key'
)

authdb_sv = AuthDbSv(mongo_sv)
acc_sv = AccSv(authdb_sv, auth_conf)
oauth_sv = OAuthSv(authdb_sv, acc_sv, auth_conf)

token = oauth_sv.mktoken_pwd('root', root_pwd)
for k, v in token.items():
    str_v = str(v)
    print('%s: %s...' % (k, str_v[0:32]))
```

```
$ python oauth_sv.py
```

Testing

```
$ python oauth_sv.py
token_type: Bearer...
refresh_token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUx...
```

```
expires_in: 1495561355.3989384...
access_token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUx...
```

TemplateSv

class `clink.service.`**`TemplateSv`**(*app_conf*, *auth_conf*)

Simple template engine

Parameters

- **`app_conf`** (`AppConf`) –
- **`auth_conf`** (`AuthConf`) –

`build_file`(*file_path*, *values*)

Read data from file then build it as a template

Parameters

- **`file_path`** (*`str`*) –
- **`values`** (*`dict`*) –

Return type str

`build_str`(*data*, *values*)

Build string template

Parameters

- **`data`** (*`str`*) –
- **`values`** (*`dict`*) –

Return type str

Example

Listing 1.21: template_sv.py

```python
from clink.service import TemplateSv
from clink.type import AppConf, AuthConf


app_conf = AppConf('dead-book', 'Hell Corporation', '1st Hell Street')

root_pwd = 'root-pwd'
root_email = 'root@email.com'
root_email_pwd = 'root-email-pwd'
root_email_server = 'smtp.email.com'
root_email_server_port = 587
auth_conf = AuthConf(
    root_pwd, root_email, root_email_pwd, root_email_server,
    root_email_server_port, 'jwt-key'
)

template_sv = TemplateSv(app_conf, auth_conf)

report_tpl = (
    'REPORT: MOTOCYCLE SPEC\n'
    'AUTHOR: $author\n\n'
    'Name       : $name\n'
    'Price      : $price\n'
    'Color      : $color\n'
    'Power      : $power'
)
values = {
    'author': 'Johnny Blaze',
    'name': 'Hell Cycle',
    'price': 'Not for Sale',
    'color': 'Black - Red Fire',
    'power': 'Unlimited'
}
report = template_sv.build_str(report_tpl, values)
print(report)
```

Testing

```
$ python template_sv.py
REPORT: MOTOCYCLE SPEC
AUTHOR: Johnny Blaze

Name       : Hell Cycle
Price      : Not for Sale
Color      : Black - Red Fire
Power      : Unlimit
```

Types

class `clink.service.mongo.type.`**`MongoDocSpec`**(*name*, *indexes*)

Specify an mongo document in database

Parameters

- **`name`** (*`str`*) –
- **`indexes`** (*`list`*[*`pymongo.IndexModel`*]) –

`name`

Name of document

`indexes`

List of indexes of document

class `clink.service.mongo.type.`**`MongoConf`**(*dburl*, *dbname*)

Specify information of mongo server

Parameters

- **`dburl`** (*`str`*) –
- **`dbname`** (*`str`*) –

`dburl`

URL to server, it follows forms

- mongodb://domain/path

•mongodb://<username>:<password>@domain/path

dbname
Name of database will be use

class `clink.service.auth.type.`**`ConfirmCodeSpec`**(*code*, *exp_date*)
Specify confirm code and expired date

Parameters

- **code** (*str*) –
- **exp_date** (*datetime*) –

Errors

exception `clink.service.mongo.error.`**`DocSpecExit`**(*name*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.service.mongo.error.`**`DocumentNotExist`**(*doc_name*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `clink.service.mongo.error.`**`DocumentIndexError`**(*doc_name*, *req_index*)

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

clink.ctl

AccCtl

class `clink.ctl.`**`AccCtl`**(*app_conf*, *auth_conf*, *acc_sv*, *oauth_sv*, *smtp_sv*, *tpl_sv*)
Manage accounts and related concepts

POST /acc/reg/code

Invoke registration code.

On successfully, an message will be send to user's email to inform about action. It contains registration code and expired date.

If code was expired, temporary account's information can be remove and other users can register with that information.

Use that code to active account in *POST /acc/reg*.

```
POST /acc/reg/code
Content-Type: application/json

{
    "properties": {
        "email": {
            "pattern": "^[a-zA-Z0-9-._]{1,64}\\@[a-zA-Z0-9\\[]{1}[a-zA-Z0-9.-:]{1,61}
↪[a-zA-Z0-9\\]]{1}$",
            "type": "string"
        },
        "name": {
            "pattern": "^[a-z0-9-]{2,32}$",
            "type": "string"
        },
        "phone": {
            "pattern": "^\\+[0-9]{2}\\s[0-9]{3}\\s([0-9]{3}|[0-9]{4})\\s[0-9]{4}$",
            "type": [
                "string",
                "null"
            ]
        },
        "pwd": {
            "pattern": "^.{6,32}$",
            "type": "string"
        }
    },
    "required": [
        "name",
        "email",
        "pwd"
    ],
    "type": "object"
}

HTTP/1.1 204 No Content
```

POST /acc/reg

Active account with registration code.

On successfully, an message will be send to user's email to inform about action. Then account becomes avaiable on system and user can use that account.

Get registration from *POST /acc/reg/code*.

```
POST /acc/reg
Content-Type: application/json

{
    "properties": {
        "code": {
            "pattern": "^([a-zA-Z]{4}-){3}[a-zA-Z]{4}$",
            "type": "string"
        }
    },
    "required": [
        "code"
    ],
    "type": "object"
}

HTTP/1.1 204 No Content
```

GET /acc/me

Retrieve account information.

```
GET /acc/me
Authorization: <bearer-token>

HTTP/1.1 200 OK

{
    "properties": {
        "_id": {
            "pattern": "^[a-fA-F0-9]{24}$",
            "type": "string"
        },
        "created_date": {
            "type": "number"
        },
        "email": {
            "pattern": "^[a-zA-Z0-9-._]{1,64}\\@[a-zA-Z0-9\\[]{1}[a-zA-Z0-9.-:]{1,61}
↪[a-zA-Z0-9\\]]{1}$",
            "type": "string"
        },
        "last_action": {
            "type": "string"
        },
        "modified_date": {
            "type": "number"
        },
        "name": {
            "pattern": "^[a-z0-9-]{2,32}$",
            "type": "string"
        },
        "phone": {
            "pattern": "^\\+[0-9]{2}\\s[0-9]{3}\\s([0-9]{3}|[0-9]{4})\\s[0-9]{4}$",
            "type": [
                "string",
                "null"
            ]
        }
    },
    "required": [
        "_id",
        "name",
        "email",
        "phone",
        "created_date",
        "modified_date",
        "last_action"
    ],
    "type": "object"
}
```

PUT /acc/me/pwd

Change password.

On successfully, an email will be send to account's email to inform about action.

```
PUT /acc/me/pwd
Authorization: <bearer-token>
Content-Type: application/json

{
    "properties": {
        "new_pwd": {
            "pattern": "^.{6,32}$",
            "type": "string"
        },
        "old_pwd": {
            "pattern": "^.{6,32}$",
            "type": "string"
        }
    },
    "required": [
        "old_pwd",
        "new_pwd"
    ],
    "type": "object"
}

HTTP/1.1 204 No Content
```

POST /acc/pwd/code

Invoke reset password code.

On successfully, an email will be send to account's email to inform about action. It contains code and expired date.

If code was expired, it can use to reset password.

Use that code to reset password in *POST /acc/pwd*.

```
POST /acc/pwd/code
Content-Type: application/json

{
    "properties": {
        "email": {
            "pattern": "^[a-zA-Z0-9-._]{1,64}\\@[a-zA-Z0-9\\[]{1}[a-zA-Z0-9.-:]{1,61}
↪[a-zA-Z0-9\\]]{1}$",
            "type": "string"
        }
    },
    "required": [
        "email"
    ],
    "type": "object"
}

HTTP/1.1 204 No Content
```

POST /acc/pwd

Set current password to new password by reset password code.

On successfully, an message will send to account's email to inform about action. It contains reset password code and expired date.

Get reset password code from *POST /acc/pwd/code*.

```
POST /acc/pwd
Content-Type: application/json

{
    "properties": {
        "code": {
            "pattern": "^([a-zA-Z]{4}-){3}[a-zA-Z]{4}$",
            "type": "string"
        },
        "new_pwd": {
            "pattern": "^.{6,32}$",
            "type": "string"
        }
    },
    "required": [
        "code",
        "new_pwd"
    ],
    "type": "object"
}

HTTP/1.1 204 No Content
```

AuthCtl

class `clink.ctl.`**`AuthCtl`**(*oauth_sv*, *acc_sv*, *smtp_sv*, *tpl_sv*)

POST /auth/token

Invoke token.

Use password

```
POST /auth/token
Content-Type: application/json

{
    "properties": {
        "grant_type": {
            "pattern": "^password$",
            "type": "string"
        },
        "password": {
            "pattern": "^.{6,32}$",
            "type": "string"
        },
        "username": {
            "pattern": "^[a-z0-9-]{2,32}$",
            "type": "string"
        }
    },
    "required": [
        "grant_type",
        "username",
        "password"
    ],
    "type": "object"
}

HTTP/1.1 200 OK
Content-Type: application/json

{
    "properties": {
        "access_token": {
            "type": "string"
        },
        "expires_in": {
            "type": "number"
        },
        "refresh_token": {
            "type": "string"
        },
        "token_type": {
            "pattern": "Bearer",
            "type": "string"
        }
    },
    "required": [
        "token_type",
        "expires_in",
        "access_token",
        "refresh_token"
    ],
    "type": "object"
}
```

Use refresh token

```
POST /auth/token
Content-Type: application/json

{
    "properties": {
        "grant_type": {
            "pattern": "^refresh_token$",
            "type": "string"
        },
        "refresh_token": {
            "type": "string"
        }
    },
    "required": [
        "grant_type",
        "refresh_token"
    ],
    "type": "object"
}

HTTP/1.1 200 OK
Content-Type: application/json

{
    "properties": {
        "access_token": {
            "type": "string"
        },
        "expires_in": {
            "type": "number"
        },
        "refresh_token": {
            "type": "string"
        },
        "token_type": {
            "pattern": "Bearer",
            "type": "string"
        }
    },
    "required": [
        "token_type",
        "expires_in",
        "access_token",
        "refresh_token"
    ],
    "type": "object"
}
```

clink.model

std

Standard data models

`clink.model.std.`**`unix_time`** **= {'type': 'number'}**

`clink.model.std.`**`objectid`** **= {'type': 'string', 'pattern': '^[a-fA-F0-9]{24}$'}**

`clink.model.std.`**`acc_name`** **= {'type': 'string', 'pattern': '^[a-z0-9-]{2,32}$'}**

`clink.model.std.`**`acc_pwd`** **= {'type': 'string', 'pattern': '^.{6,32}$'}**

`clink.model.std.`**`email`** **= {'type': 'string', 'pattern': '^[a-zA-Z0-9-._]{1,64}\\@[a-zA-Z0-9\\[]{1}[a-zA-Z0-9.-:]{1,61}[a-zA**

`clink.model.std.`**`phone`** **= {'type': ['string', 'null'], 'pattern': '^\\+[0-9]{2}\\s[0-9]{3}\\s([0-9]{3}|[0-9]{4})\\s[0-9]{4}$'}**

acc

Models is related with account management

`clink.model.acc.`**`confirm_code`** **= {'type': 'string', 'pattern': '^([a-zA-Z]{4}-){3}[a-zA-Z]{4}$'}**

`clink.model.acc.`**`get_me_body`** **= {'type': 'object', 'required': ['_id', 'name', 'email', 'phone', 'created_date', 'modified_**

`clink.model.acc.`**`post_reg_code_body`** **= {'type': 'object', 'required': ['name', 'email', 'pwd'], 'properties': {'email':**

`clink.model.acc.`**`post_reg`** **= {'type': 'object', 'required': ['code'], 'properties': {'code': {'type': 'string', 'pattern': '^([**

`clink.model.acc.`**`put_pwd`** **= {'type': 'object', 'required': ['old_pwd', 'new_pwd'], 'properties': {'old_pwd': {'type': 'stri**

`clink.model.acc.`**`post_rpwd_code`** **= {'type': 'object', 'required': ['email'], 'properties': {'email': {'type': 'string', 'pat**

`clink.model.acc.`**`post_rpwd`** **= {'type': 'object', 'required': ['code', 'new_pwd'], 'properties': {'code': {'type': 'string', '**

auth

Authentication model

`clink.model.auth.`**`post_token_pwd`** **= {'type': 'object', 'required': ['grant_type', 'username', 'password'], 'properties':**

`clink.model.auth.`**`post_token_rtoken`** **= {'type': 'object', 'required': ['grant_type', 'refresh_token'], 'properties': {'r**

`clink.model.auth.`**`res_bearer_token`** **= {'type': 'object', 'required': ['token_type', 'expires_in', 'access_token', 'refres**

clink.util

fs

`clink.util.fs.`**`asset_path`**(*rel_path*)

Return absolute path to Clink's asset directory

Parameters **`path`** (*`str`*) –

Return type str

shell

`clink.util.shell.`**`call`**(*args*, *cwd=None*)

Perform new processing

Parameters

- **`args`** (*`list[str]`*) –
- **`cwd`** (*`str`*) –

`clink.util.shell.`**`cp`**(*src*, *dest*, *exist_ignore=True*)

Copy file or directory

Parameters

- **`src`** (*`str`*) –

- **dest** (*str*) –
- **exist_ignore** (*bool*) –

`clink.util.shell.`**`rm`**(*path*, *exist_ignore=True*)
Remove files

Parameters

- **path** (*str*) –
- **exist_ignore** (*bool*) –

`clink.util.shell.`**`mkdir`**(*path*)
Create new directory

Parameters **path** (*str*) –

`clink.util.shell.`**`chmod`**(*path*, *mode*)
Change mode of file

Parameters

- **path** (*str*) –
- **mode** (*int*) –

`clink.util.shell.`**`touch`**(*path*, *mode*)
Create new empty file

Parameters

- **path** (*str*) –
- **mode** (*int*) –

Development

Installation

Clink is not depends on mongodb server but it's testing do. Mongodb server is not in standard package repository of Linux distros, install it by hand here https://docs.mongodb.com/manual/administration/install-on-linux/.

Then follow instructions:

```
# essential tools
$ apt-get install python3 git

# clone source code
$ git clone https://github.com/kevin-leptons/clink
$ cd clink

# enter virtual environment
$ ./env init
$ . venv/bin/active

# install dependency packages
$ ./env install

# create test configuration
export CLINK_TEST_ROOT_EMAIL='test-mail@gmail.com'
export CLINK_TEST_ROOT_EMAIL_PWD='test-mail-pwd'
export CLINK_TEST_ROOT_EMAIL_SERVER='smtp.gmail.com'
export CLINK_TEST_ROOT_EMAIL_SERVER_PORT='587'
```

Develop

```
# build document
$ ./ctl doc

# view document
$ ./ctl doc --view

# test
$ ./ctl test

# build and push pip package to pypi
# it required authentication
$ ./ctl release
```

References

Advanced Packaging Tool https://en.wikipedia.org/wiki/Advanced_Packaging_Tool

Python https://en.wikipedia.org/wiki/Python_(programming_language)

curl https://en.wikipedia.org/wiki/CURL

pip https://en.wikipedia.org/wiki/Pip_(package_manager)

wget https://en.wikipedia.org/wiki/Wget

Python Module Index

Index