Cliche Documentation

Release 0.0.0

Cliche.io

May 09, 2015

Contents

Cliche is an ontology of fictional works.

CHAPTER 1

Getting started

This tutorial covers how to get started with the Cliche codebase.

1.1 Prerequisites

Cliche is made with the following softwares:

Python 3.3 or higher Cliche is mostly written in Python language. It's a high-level scripting language for general purpose.

Git We use Git for version control. We can track changes of Cliche source code using it.

One of its main downsides is Windows support. If you're not using Mac or Linux, it's hard to setup. We recommend you to simply use GitHub for Windows.

If you're using Mac you can find installers of these softwares in their official websites.

If you're using Windows you can find CPython installer in Python website's download page. For Git, install GitHub for Windows as mentioned above.

You can install these softwares using APT if you're on Ubuntu or Debian Linux:

```
$ sudo apt-get install python3 git-core
```

There are other several third-party Python libraries as well, but you don't have to install it by yourself. These can be automatically resolved.

1.2 Check out the source code

Cliche codebase is managed under GitHub, so you can clone it using Git:

```
$ git clone git@github.com:clicheio/cliche.git
$ cd cliche/
```

If you're using Windows, you can clone it using GitHub for Windows as well.

1.3 Create an environment

The next step is creating an environment for Cliche. This step should be done first, and only once.

Each environment has its own directory for storing the `site-packages` folder, executable scripts, etc. Here we assume that the name of environment folder is `cliche-env`:

```
$ pyvenv cliche-env
```

1.4 Enter the environment

Each time you work on Cliche, you have to enter the created environment. It's applied to each terminal session.

Note: If you're on Windows, you should *not* run the command prompt as administrator.

We assume here that the environment name is `cliche-env` and the repository name is `cliche`.

```
$ . cliche-env/bin/activate
(cliche-env) $
```

Note: On Windows execute **`Scripts\activate.bat`** instead:

```
C:\Users\John Doe> cliche-env\Scripts\activate.bat
(cliche-env) C:\Users\John Doe>
```

The prefix `(cliche-env)` of the prompt indicates you're in the environment. And then install Cliche's dependencies in the environment (instead of system-wide `site-packages`).

```
(cliche-env) $ cd cliche/
(cliche-env) cliche$ pip install -e .
```

1.5 Resolve dependencies

Cliche depends on several third-party Python libraries, and these can be automatically resolved through **`pip`** command:

```
$ pip install -e .
Finished processing dependencies for Cliche==0.0.0
```

1.6 Configuration

To run web server a configuration file is required. Here we assume the filename we'll use is `dev.cfg.py`. Configuration file is an ordinary Python script. Create a new file and save:

```
DATABASE_URL = 'postgresql:///cliche_db'
```

1.7 Relational database

Cliche stores data into a relational database through SQLAlchemy. In production we use PostgreSQL and it also works well with SQLite.

Python ships with SQLite. There is no need to install it. To use PostgreSQL, read Setup PostgreSQL.

Schema has to be created on the database. Use **`cliche upgrade`** command:

```
$ cliche upgrade -c dev.cfg.py
```

CHAPTER 2

Setup PostgreSQL

We use PostgreSQL for Cliche in the production. PostgreSQL is the most powerful open-source relational database system in the world.

To build up your local development environment near production, we recommend you also to install PostgreSQL in your local box.

2.1 Install PostgreSQL

2.1.1 Mac

Postgres.app must be the easiest way to install PostgreSQL on your Mac. Go, download, and install.

Then, create a role and a database.

```
$ createuser -s `whoami`
$ createdb cliche_db -E utf8 -T postgres
```

2.1.2 Ubuntu/Debian Linux

Install it using APT, and then create a role and a database.

```
$ sudo apt-get install postgresql
$ sudo -u postgres createuser -s `whoami`
$ createdb cliche_db -E utf8 -T postgres
```

2.2 Connect from Python

Install `psycopg2` using APT (if you're on Ubuntu/Debian):

```
$ sudo apt-get install python-psycopg2
```

Or you can install it using **`pip`**, of course. However you have to install libpq (including its headers) and CPython headers to link first. These things can be installed using APT as well if you're on Ubuntu or Debian:

```
$ sudo apt-get install libpq libpq-dev python-dev
```

Or **`yum`** if you're CentOS/Fedora/RHEL:

```
$ sudo yum install postgresql-libs postgresql-devel python-devel
```

And then run **pip**:

```
$ pip install psycopg2
```

2.3 Configuration

```
DATABASE_URL = 'postgresql:///cliche_db'
```

CHAPTER 3

Database migration

We are using Alembic for database migration.

3.1 Get up to date

To upgrade the schema to the latest revision, simply run **`cliche upgrade`** command:

```
$ cliche upgrade -c dev.cfg.py
INFO  [alembic.migration] Context impl PostgresqlImpl.
INFO  [alembic.migration] Will assume transactional DDL.
INFO  [alembic.migration] Running upgrade 256db34030b7 -> 2a1ebdf4c19e
```

3.2 Add a new revision

Each time you change the database schema, a new migration script has to be added. Use **`setup.py revision`** command:

```
$ python setup.py revision -c dev.cfg.py --autogenerate -m "add x to y"
```

Note: You should use `--autogenerate` option to automatically generate a new migration script from delta between the actual database schema and the head migration script. It does not 100% completely work well, you have to manually review a generated new migration script before commit it.

See also:

Operation Reference – Alembic The reference of operations Alembic provides.

3.3 List revision history

Use `history` command if you want to list all revisions of the database migration history:

```
$ python setup.py history -c dev.cfg.py
running history

Rev: 1ed82ef0071 (head)
Parent: 1344c33541b
Path: /.../cliche/migrations/versions/1ed82ef0071_add_dod_column_to_person.py
```

```
    Add dod column to Person

    Revision ID: 1ed82ef0071
    Revises: 1344c33541b
    Create Date: 2014-08-04 21:48:04.403449

Rev: 1344c33541b
Parent: 2d8b17e13d1
Path: /.../cliche/migrations/versions/1344c33541b_add_team_memberships_table.py

    Add team_memberships table

    Revision ID: 1344c33541b
    Revises: 2d8b17e13d1
    Create Date: 2014-02-27 03:05:00.853963

Rev: 2d8b17e13d1
Parent: 27e81ea4d86
Path: /.../cliche/migrations/versions/2d8b17e13d1_add_teams_table.py

    Add teams table

    Revision ID: 2d8b17e13d1
    Revises: 27e81ea4d86
    Create Date: 2014-02-27 02:00:25.694782

Rev: 27e81ea4d86
Parent: None
Path: /.../cliche/migrations/versions/27e81ea4d86_add_people_table.py

    Add people table

    Revision ID: 27e81ea4d86
    Revises: None
    Create Date: 2014-02-27 00:50:04.698519
```

3.4 Merge branches

The **`cliche upgrade`** script will refuse to run any migrations if there are two or more heads at a time:

```
$ cliche upgrade -c dev.cfg.py
INFO  [alembic.context] Context class PostgresqlContext.
INFO  [alembic.context] Will assume transactional DDL.
Exception: Only a single head supported so far...
```

If you want to see how it's going on, list the history. It would show you there are two heads:

```
$ python setup.py history -c dev.cfg.py
running history

Rev: 2d8e07def2 (head)
Parent: 1344c33541b
Path: /.../cliche/migrations/versions/2d8e07def2_add_nationality_column_to_people_table.py

    Add nationality column to people table
```

```
    Revision ID: 2d8e07def2
    Revises: 1ed82ef0071
    Create Date: 2014-08-08 02:38:45.072148


Rev: 1ed82ef0071 (head)
Parent: 1344c33541b
Path: /.../cliche/migrations/versions/1ed82ef0071_add_dod_column_to_person.py

    Add dod column to Person

    Revision ID: 1ed82ef0071
    Revises: 1344c33541b
    Create Date: 2014-08-04 21:48:04.403449

Rev: 1344c33541b (branchpoint)
Parent: 2d8b17e13d1
Path: /.../cliche/migrations/versions/1344c33541b_add_team_memberships_table.py

    Add team_memberships table

    Revision ID: 1344c33541b
    Revises: 2d8b17e13d1
    Create Date: 2014-02-27 03:05:00.853963

Rev: 2d8b17e13d1
Parent: 27e81ea4d86
Path: /.../cliche/migrations/versions/2d8b17e13d1_add_teams_table.py

    Add teams table

    Revision ID: 2d8b17e13d1
    Revises: 27e81ea4d86
    Create Date: 2014-02-27 02:00:25.694782

Rev: 27e81ea4d86
Parent: None
Path: /.../cliche/migrations/versions/27e81ea4d86_add_people_table.py

    Add people table

    Revision ID: 27e81ea4d86
    Revises: None
    Create Date: 2014-02-27 00:50:04.698519
```

In this case you have to rebase one side's `down_revisions` to another head:

```
"""Add nationality column to people table

Revision ID: 2d8e07def2
Revises: 1ed82ef0071  # changed from 1344c33541b
Create Date: 2014-08-08 02:38:45.072148

"""

# revision identifiers, used by Alembic.
revision = '2d8e07def2'
# changed from 1344c33541b
down_revision = '1ed82ef0071'
```

CHAPTER 4

Testing

4.1 tests/

The test suite for Cliche is in `tests/` directory. All Python source files with `_test.py` suffix are executed by pytest framework.

4.2 Running tests

You can run tests using **`setup.py`** script or **`py.test`** command which is provided pytest package:

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

two ways provide slightly different features:

`python setup.py test` It installs dependencies including testing libraries like pytest first if these are not completely resolved yet. It only supports very basic *Test settings* thorugh environment variables.

`py.test` You have to install packages for testing e.g. pytest by yourself to use this command. You can resolve these using **`pip`**:

```
$ pip install -e .[tests]
```

It provides more rich options like *Partial testing*.

4.3 Test settings

You can configure settings like database connection for unit testing. These can be set using command line options of **`py.test`**, or environment variables. Here's the short listing:

Command Option	Environment Variable	Meaning
`--database-url`	`CLICHE_TEST_DATABASE_URL`	Database URL for testing. SQLite 3 by default.
`--echo-sql`		Print all executed queries for failed tests. See *Query logging of failed tests*

4.4 Partial testing

The complete test suite is slow to run. Slow feedback loop decreases productivity. (You would see your Facebook timeline or chat in the IRC to wait the long time of testing.) To quickly test what you're workin on, you can run only part of complete test suite.

If you set `-k` test will run only particular matched suites and others get skipped.

```
$ py.test -k work_test
$ py.test -k test_work_has_awards
```

The option `--maxfail` is useful as well, it exits after the specified number of failures/errors. The `-x` option is equivalent to `--maxfail=1`, it exits instantly on the first failure/error.

See also:

Excluding tests with py.test 2.3.4 using -k selection Since pytest version 2.3.4, the `-k` keyword supports expressions.

4.5 Query logging of failed tests

Sometimes you would want to see logs of database queries to debug failed tests. You can see these using `--echo-sql` option.

```
$ py.test --echo-sql
```

CHAPTER 5

How to run crawler

This tutorial covers how to run the cliche crawlers.

5.1 Running TVTropes crawler

You can run TVTropes crawler using **`cliche crawler`** command with **`celery worker`**:

```
$ celery worker -A cliche.services.tvtropes.crawler \
  --config CONFIG_FILENAME_WITHOUT_EXT
$ cliche crawler
```

with subcommands you can provide options:

`celery worker` It runs celery worker to crawl links. You can supply `--purge` option for purging pending work queue, and `-f LOG_FILE` to save logs into a file.

`cliche crawler` You have to provide config file with `-c CONFIG_FILE` option or `CLICHE_CONFIG` environmental variable. Config option must be provided before `crawler` subcommand.

when the crawler is first run, it will fetch and populate the celery queue with links from TVTropes Index Report. If there is already some crawled links in the database, the crawler will skip this step and populate the queue from the database.

5.2 Running Wikipedia crawler

You can run Wikipedia crawler in the same way using **`cliche crawler`** command with **`celery worker`**:

```
$ celery worker -A cliche.services.wikipedia.crawler \
  --config dev.py
$ cliche sync wikipedia -c CONFIG_FILENAME_WITHOUT_EXT
```

It also provides same options.

CHAPTER 6

cliche — Ontology of fictional works

6.1 cliche.celery — Celery-backed task queue worker

Sometimes web app should provide time-consuming features that cannot immediately respond to user (and we define "immediately" as "shorter than a second or two seconds" in here). Such things should be queued and then processed by background workers. Celery does that in natural way.

We use this at serveral points like resampling images to make thumbnails, or crawling ontology data from other services. Such tasks are definitely cannot "immediately" respond.

See also:

What kinds of things should I use Celery for? — Celery FAQ Answer to what kinds of benefits are there in Celery.

Queue everything and delight everyone This article describes why you should use a queue in a web application.

6.1.1 How to define tasks

In order to defer some types of tasks, you have to make these functions a task. It's not a big deal, just attach a decorator to them:

```
@celery.task(ignore_result=True)
def do_heavy_work(some, inputs):
    '''Do something heavy work.'''
    ...
```

6.1.2 How to defer tasks

It's similar to ordinary function calls except it uses `delay()` method (or `apply_async()` method) instead of calling operator:

```
do_heavy_work.delay('some', inputs='...')
```

That command will be queued and sent to one of distributed workers. That means these argument values are serialized using `json`. If any argument value isn't serializable it will error. Simple objects like numbers, strings, tuples, lists, dictionaries are safe to serialize. In the other hand, entity objects (that an instance of *`cliche.orm.Base`* and its subtypes) mostly fail to serialize, so use primary key values like entity id instead of object itself.

6.1.3 What things are ready for task?

Every deferred call of task share equivalent inital state:

- You can get a database session using *get_session()*.
- You also can get a database engine using *get_database_engine()*.

While there are several things not ready either:

- Flask's request context isn't ready for each task. You should explicitly deal with it using `request_context()` method to use context locals like `flask.request`. See also The Request Context.
- Physical computers would differ from web environment. Total memory, CPU capacity, the number of processors, IP address, operating system, Python VM (which of PyPy or CPython), and other many environments also can vary. Assume nothing on these variables.
- Hence global states (e.g. module-level global variables) are completely isolated from web environment which called the task. Don't depend on such global states.

6.1.4 How to run Celery worker

celery worker (formerly **celeryd**) takes Celery app object as its endpoint, and Cliche's endpoint is `cliche.celery.celery`. You can omit the latter variable name and module name: *cliche*. Execute the following command in the shell:

```
$ celery worker -A cliche --config dev.cfg.yml
 -------------- celery@localhost v3.1.13 (Cipater)
---- **** -----
--- * ***  * -- Darwin-13.3.0-x86_64-i386-64bit
-- * - **** ---
- ** ---------- [config]
- ** ---------- .> app:         cliche.celery:0x1... (cliche.celery.Loader)
- ** ---------- .> transport:   redis://localhost:6379/5
- ** ---------- .> results:     disabled
- *** --- * --- .> concurrency: 4 (prefork)
-- ******* ----
--- ***** ----- [queues]
 -------------- .> celery           exchange=celery(direct) key=celery


[2014-09-12 00:31:25,150: WARNING/MainProcess] celery@localhost ready.
```

Note that you should pass the same configuration file (`--config` option) to the WSGI application. It should contain `DATABASE_URL` and so on.

6.1.5 References

class `cliche.celery.`**Loader**(*app*, ***kwargs*)
The loader used by Cliche app.

`cliche.celery.`**get_database_engine**() → sqlalchemy.engine.base.Engine
Get a database engine.

Returns a database engine

Return type `sqlalchemy.engine.base.Engine`

cliche.celery.**get_session**() -> *sessionmaker(class_='Session', expire_on_commit=True, bind=None, autocommit=True, autoflush=True)*

Get a database session.

Returns a database session

Return type Session

cliche.celery.**get_raven_client**() → raven.base.Client

Get a raven client.

Returns a raven client

Return type raven.Client

6.2 cliche.cli — Command-line interfaces

cliche.cli.**initialize_app**(*config=None*)

Initialize celery/flask app.

Parameters **config** – a config file path. accept .py, .yml file. default value is None

cliche.cli.**config**(*func*)

Provide --config or -c option and run *initialize_app()* automatically.

Parameters **func** (collections.abc.Callable) – a command function to decorate

Returns decorated func

cliche.cli.**main = <click.core.Group object>**

(collections.abc.Callable) The CLI entry point.

6.3 cliche.config — Reading configurations

class cliche.config.**ConfigDict**

Almost the same to the built-in dict except it raises *ConfigKeyError* instead of KeyError with finer error message.

exception cliche.config.**ConfigKeyError**

The exception raised when there's no such configured key, that is a subtype of built-in KeyError.

cliche.config.**read_config**(*filename*)

Read Cliche app configuration from the given filename.

```
config = read_config(filename='dev.cfg.yml')
```

Note that it takes only one keyword argument at a time. All parameters are mutually exclusive for each other.

Parameters **filename** (pathlib.Path) – read config from a *filename* of yaml or python source code

Returns the parsed dictionary with uppercase keys

Return type *ConfigDict*

cliche.config.**read_config_from_python**(*, *string=None*, *file=None*, *filename=None*)

Read Cliche app configuration from Python code i.e. Flask-style configuration:

```
config = read_config_from_python(filename='dev.cfg.py')
```

Note that it takes only one keyword argument at a time. All parameters are mutually exclusive for each other.

Parameters

- **string** (str) – read config from a python source code string
- **file** – read config from a *file object* of python source code
- **filename** (pathlib.Path) – read config from a *filename* of python source code

Returns the parsed dictionary with uppercase keys

Return type collections.abc.Mapping

cliche.config.**read_config_from_yaml**(*, *string=None*, *file=None*, *filename=None*)

Read Cliche app configuration from YAML.

```
config = read_config_from_yaml(filename='dev.cfg.yml')
```

Note that it takes only one keyword argument at a time. All parameters are mutually exclusive for each other.

Parameters

- **string** (str) – read config from a yaml string
- **file** – read config from a *file object* of yaml
- **filename** (pathlib.Path) – read config from a *filename* of yaml

Returns the parsed dictionary with uppercase keys

Return type *ConfigDict*

6.4 cliche.credentials — Authentication methods

class cliche.credentials.**TwitterCredential**(**kwargs*)

Information about Twitter User

id
(int) The primary key from Credential.id.

identifier
(int) Twitter user id

token
(str) The oauth token.

token_secret
(str) The oauth secret token.

6.5 cliche.orm — Object-relational mapping

Cliche uses the relational database and data on the database are mapped to objects. It widely uses SQLAlchemy as its ORM (object-relational mapping) framework.

In order to define a persist model class, just subclass *Base*:

```
from sqlalchemy import Column, Integer, UnicodeText

from .orm import Base
```

```
class Thing(Base):
    '''A something object-relationally mapped.'''

    id = Column(Integer, primary_key=True)
    value = Column(UnicodeText, nullable=False)
    __repr_columns = id, value
    __tablename__ = 'things'
```

class cliche.orm.**Base**(***kwargs*)
SQLAlchemy declarative base class.

__repr_columns__
(collections.abc.Sequence) This columns will be printed to repr() string of its instances if *__repr_columns__* is defined.

See also:

SQLAlchemy — Declarative Declarative allows all three to be expressed at once within the class declaration.

cliche.orm.**Session = sessionmaker(class_='Session',expire_on_commit=True, bind=None, autocommit=True, autoflush=**
SQLAlchemy session class.

See also:

SQLAlchemy — Using the Session Session is the primary usage interface for persistence operations.

cliche.orm.**downgrade_database**(*engine*, *revision*)
Reverts to a previous revision.

Parameters

- **engine** (sqlalchemy.engine.base.Engine) – the database engine to revert
- **revision** (str) – the previous revision to revert to

cliche.orm.**get_alembic_config**(*engine*)
Creates a configuration for alembic. You can pass an Engine object or a string of database url either. So:

```
from sqlalchemy import create_engine
engine = create_engine('postgresql://localhost/cliche')
alembic_cfg = get_alembic_config(engine)
```

is equivalent to:

```
db_url = 'postgresql://localhost/cliche'
alembic_cfg = get_alembic_config(db_url)
```

Parameters **engine** (sqlalchemy.engine.base.Engine, str) – the database engine to use

Returns an alembic config

Return type alembic.config.Config

cliche.orm.**get_database_revision**(*engine*)
Gets the current revision of the database.

Parameters **engine** (sqlalchemy.engine.base.Engine) – the database engine to get the current revision

Returns the script of the current revision

Return type alembic.script.Script

cliche.orm.**import_all_modules**(*dry_run=False*)
Import all submodules of *cliche* to ensure every ORM entity classes are ready to use. It's useful for being ready to auto-generate a migration script.

Returns the set of module names

Return type collections.abc.Set

cliche.orm.**initialize_database**(*engine*)
Creates all database schemas and stamps it as the head of versions.

Parameters **engine** (sqlalchemy.engine.base.Engine) – the database engine to initialize

cliche.orm.**upgrade_database**(*engine*, *revision='head'*)
Upgrades the database schema to the chosen `revision` (default is head).

Parameters

- **engine** (sqlalchemy.engine.base.Engine) – the database engine to upgrade
- **revision** (str) – the revision to upgrade to. default is `'head'`

6.6 cliche.people — Artists, teams, and editors

class cliche.people.**Person**(***kwargs*)
People i.e. artists, editors.

created_at
(datetime.datetime) The created time.

credits
(collections.abc.MutableSet) The set of *cliche.work.Credit*s that the person has.

dob
(datetime.date) The date of birth.

dod
(datetime.date) The date of death.

id
(int) The primary key integer.

memberships
(collections.abc.MutableSet) The set of *TeamMembership*s he/she has.

teams
(collections.abc.MutableSet) The set of *Team*s he/she belongs to.

class cliche.people.**Team**(***kwargs*)
Teams (including ad-hoc teams).

created_at
(datetime.datetime) The created time.

credits
(collections.abc.MutableSet) The set of *cliche.work.Credit*s in which the team was involved.

id
(int) The primary key integer.

members
(collections.abc.MutableSet) The members *Person* set.

memberships
(collections.abc.MutableSet) The set of *TeamMembership*s that the team has.

class cliche.people.**TeamMembership**(***kwargs*)
Team memberships to people.

created_at
(datetime.datetime) The added time.

member
(*Person*) The member who is a member of the *team*.

member_id
(int) *Person.id* of *member*.

team
(*Team*) The team that the *member* belongs to.

team_id
(int) *Team.id* of *team*.

6.7 cliche.services — Interfacing external services

6.7.1 How to add a new external service

In order to add a new service to cliche, you must create a subpackage under *cliche.services* and expose some methods referring to interfaces, using __init__.py.

6.7.2 Interfaces needed to be exposed

- sync(): Method to delay a main crawling task to the queue. It should be decorated with @app.task to be defined as a celery app worker task. It should have no arguments and no return. Every output should be made as a log to celery logger.

6.7.3 Example __init__.py

```
from .crawler import crawl as sync  # noqa


__all__ = 'sync',
```

Note that you will need the import lines annotated with # noqa because otherwise **flake8** will consider it as unused import.

cliche.services.align — String matching to align

class cliche.services.align.**ExternalId**(**kwargs)
Relationship between two kinds of external works This class can be replaced based on the probability of equality.

id
(int) The primary key integer.

tvtropes
(collections.abc.MutableSet) The set of *cliche.services.tvtropes.entities.Entity*.

tvtropes_name
(str) The name of the trope.

tvtropes_namespace
(str) The namespace of the trope, both namespace and name determines one trope.

wikipedia
(collections.abc.MutableSet) The set of *cliche.services.wikipedia.work.Entity*.

wikipedia_id
(str) The namespace of the trope.

work
(collections.abc.MutableSet) The set of cliche.work.Entity.

work_id
(int) foreignkey for works.id

cliche.services.tvtropes — Interfacing TVTropes

cliche.services.tvtropes.crawler — TVTropes crawler

cliche.services.tvtropes.crawler.**fetch_link**(*url*, *session*, *, *log_prefix=''*)
Returns result, tree, namespace, name, final_url.

cliche.services.tvtropes.entities — Data entities for TVTropes

class cliche.services.tvtropes.entities.**ClicheTvtropesEdge**(**kwargs)
Correspondence between Works of Cliche and TV Tropes

class cliche.services.tvtropes.entities.**Entity**(**kwargs)
Representation of a TVTropes page.

class cliche.services.tvtropes.entities.**Redirection**(**kwargs)
Representation of an alias of *Entity*.

class cliche.services.tvtropes.entities.**Relation**(**kwargs)
Associate *Entity* to other *Entity*.

cliche.services.wikipedia — Crawl data from Wikipedia via DBpedia

cliche.services.wikipedia.crawler — Wikipedia crawler

Crawling DBpedia tables into a relational database

See also:

The list of dbpedia classes This page describes the structure and relation of DBpedia classes.

References

cliche.services.wikipedia.crawler.**count_by_class**(*class_list*)

Get count of a ontology class

Parameters **class_list** (*list*) – List of properties

Return type int

cliche.services.wikipedia.crawler.**count_by_relation**(*p*)

Get count of all works

Parameters **p** (*list*) – List of properties

Return type int

cliche.services.wikipedia.crawler.**select_by_class**(*s*, *s_name='subject'*, *p={}*, *entities=[]*, *page=1*)

List of **s** which as property as **entities**

Parameters

- **s** (*str*) – Ontology name of subject.
- **s_name** (*str*) – Name of subject. It doesn't affect the results.
- **entities** (*list*) – List of property ontologies.
- **page** (int) – The offset of query, each page will return 100 entities.

Returns list of a dict mapping keys which have 'entities' as property.

Return type list

For example:

```
select_by_class (
    s_name='author',
    s=['dbpedia-owl:Artist', 'dbpedia-owl:ComicsCreator'],
    p=['dbpedia-owl:author', 'dbpprop:author', 'dbpedia-owl:writer'],
    entities=['dbpedia-owl:birthDate', 'dbpprop:shortDescription']
)
```

```
[{
    'author': 'http://dbpedia.org/page/J._K._Rowling',
    'name': 'J. K. Rowling',
    'dob' : '1965-07-31',
    'shortDescription' : 'English writer. Author of the Harry ...'
    },{
    'author': ...
}]
```

cliche.services.wikipedia.crawler.**select_by_relation**(*p*, *revision*, *s_name='subject'*, *o_name='object'*, *page=1*)

Find author of something

Retrieves the list of s_name and o_name, the relation is a kind of ontology properties.

Parameters

- **p** (*list*) – List of properties between s_name and o_name.

- **s_name** (*str*) – Name of subject. It doesn't affect the results.
- **o_name** (*str*) – Name of object. It doesn't affect the results.
- **page** (int) – The offset of query, each page will return 100 entities.

Returns list of a dict mapping keys to the matching table row fetched.

Return type list

For example:

```
select_by_relation(s_name='work',
p=['dbpprop:author', 'dbpedia-owl:writer', 'dbpedia-owl:author'],
o_name='author', page=0)
```

```
[{
    'work':'http://dbpedia.org/resource/The_Frozen_Child',
    'author': 'http://dbpedia.org/resource/József_Eötvös
    http://dbpedia.org/resource/Ede_Sas'
    },{
    'work':'http://dbpedia.org/resource/Slaves_of_Sleep',
    'author': 'http://dbpedia.org/resource/L._Ron_Hubbard'
}]
```

When the row has more than two items, the items are combined by EOL.

cliche.services.wikipedia.crawler.**select_property**(*s*, *s_name='property'*, *return_json=False*)

Get properties of a ontology.

Parameters **s** (*str*) – Ontology name of subject.

Returns list of objects which contain properties.

Return type list

For example:

```
select_property(s='dbpedia-owl:Writer', json=True)
```

```
[{
    'property' : 'rdf:type'
    },{
    'property' : 'owl:sameAs'
}]
```

cliche.services.wikipedia.work — Data entities for Wikipedia

All classes in this file are rdfs:domain of its columns.

class cliche.services.wikipedia.work.**ClicheWikipediaEdge**(***kwargs*)
Correspondence between Works of Cliche and Wikipedia

class cliche.services.wikipedia.work.**Entity**(***kwargs*)
Representation of entities.

class cliche.services.wikipedia.work.**Relation**(***kwargs*)
Representation of relations.

class cliche.services.wikipedia.work.**Artist**(***kwargs*)
Representation of artist as an ontology.

class cliche.services.wikipedia.work.**Work**(***kwargs*)
Representation of work as an ontology.

class cliche.services.wikipedia.work.**Film**(***kwargs*)
Representation of film as an ontology.

class cliche.services.wikipedia.work.**Book**(***kwargs*)
Representation of book as an ontology.

6.8 cliche.sqltypes — Collection of custom types for SQLAlchemy.

class cliche.sqltypes.**EnumType**(*enum_class: enum.Enum*, ***kw*)
Custom enum type to be used as `enum.Enum`in Python standard library. It inherits :class:`sqlalchemy.types.SchemaType` since it requires schema-level DDL. PostgreSQL ENUM type defined in an Alembic script must be explicitly created/dropped.

impl
alias of `Enum`

class cliche.sqltypes.**HashableLocale**(*language*, *territory=None*, *script=None*, *variant=None*)
Hashable Locale

class cliche.sqltypes.**LocaleType**(**args*, ***kwargs*)
Custom locale type to be used as `babel.Locale`.

impl
alias of `String`

class cliche.sqltypes.**UuidType**(**args*, ***kwargs*)
Custom UUID type to be used as `uuid.UUID`.

impl
alias of `CHAR`

6.9 cliche.user — Users

class cliche.user.**User**(***kwargs*)
Registered user in cliche.io with social providers.

created_at
(`datetime.datetime`) The date and time on which the record was created.

credentials
(`collections.abc.MutableSet`) The credentials matched.

id
(`int`) The primary key integer.

name
(`str`) The name for using in user list.

6.10 cliche.web — Web application

6.10.1 cliche.web.app — Flask application object

cliche.web.app.**app = <Flask 'cliche.web.app'>**
(flask.Flask) The Flask application object.

cliche.web.app.**check_login_status**()
Check he/she logged and expired.

cliche.web.app.**index**()
Cliche.io web index page.

6.10.2 cliche.web.db — Database connections

Use *session* in view functions.

cliche.web.db.**close_session**(*exception=None*)
Close an established session.

cliche.web.db.**get_database_engine**()
Get a database engine.

Returns a database engine

Return type sqlalchemy.engine.base.Engine

cliche.web.db.**get_database_engine_options**()
Get a dictionary of options for SQLAlchemy engine. These options are used by *get_database_engine()* and passed to sqlalchemy.create_engine() function.

cliche.web.db.**get_session**()
Get a session. If there's no yet, create one.

Returns a session

Return type *Session*

cliche.web.db.**session = <LocalProxy unbound>**
(LocalProxy of *Session*) The context local session. Use this.

cliche.web.db.**setup_session**(*app*)
Setup the app to be able to use *session*.

Parameters **app** (Flask) – the Flask application to setup

6.10.3 cliche.web.ontology — Ontology web views

Cliche provides ontology web pages to use our database. It widely uses Flask as its web framework.

cliche.web.ontology.**trope_list**()
A list of name of tropes.

cliche.web.ontology.**trope_page**(*name*)
More detailed data of a work.

cliche.web.ontology.**work_list**()
A list of id-name pairs of works.

cliche.web.ontology.**work_page**(*title*)
More detailed data of a work.

6.10.4 cliche.web.social.provider — Social Support

Cliche provides Twitter login/join to use our service. It widely uses Flask-OAuthlib as its OAuth framework.

cliche.web.social.provider.**oauth = <flask_oauthlib.client.OAuth object>**
(flask_oauthlib.client.OAuth) OAuth client.

cliche.web.social.provider.**twitter = <flask_oauthlib.client.OAuthRemoteApp object>**
Twitter OAuth client.

6.10.5 cliche.web.social.twitter — Twitter Support

Cliche provides Twitter login/join to use our service. It widely uses Flask-OAuthlib as its OAuth framework.

class cliche.web.social.oauth.**Vendor**(*name*, *credential_table*, *oauth_version*, *oauth_client*, *key_names*)

credential_table
Alias for field number 1

key_names
Alias for field number 4

name
Alias for field number 0

oauth_client
Alias for field number 3

oauth_version
Alias for field number 2

cliche.web.social.oauth.**login**(*vendor*)
Login.

cliche.web.social.oauth.**oauth_authorized**(*vendor*)
Authorized OAuth and login or join with social account.

6.10.6 cliche.web.user — User web views

Cliche provides user web pages to use our database. It widely uses Flask as its web framework.

cliche.web.user.**logout**()
Logout.

6.11 cliche.work — Things associated with a creative work.

class cliche.work.**Character**(***kwargs*)
Fictional character that appears in creative work.

derived_characters
(:class:'collections.abc.MutableSet') The set of *Character*s which is derived from this character

id
(int) The primary key integer.

original_character
(:class:'Character') The original character from which this character is derived.

original_character_id
(:class:'int') *Character.id* of *original_character*.

works
(collections.abc.MutableSet) The set of *Work*s in which the character appeared.

class cliche.work.**Credit**(***kwargs*)
Relationship between the work, the person, and the team. Describe that the person participated in making the work.

created_at
(datetime.datetime) The date and time on which the record was created.

person
(*cliche.people.Person*) The person who made the *work*.

person_id
(int) *cliche.people.Person.id* of *person*.

role
The person's role in making the work.

team
The team which the person belonged when work had been made.

team_id
(int) Team.id of *team*. (optional)

work
(*Work*) The work which the *person* made.

work_id
(int) *Work.id* of *work*.

class cliche.work.**Franchise**(***kwargs*)
Multimedia franchise that is a franchise for which installments exist in multiple forms of media, such as books, comic books, and films, for example *The Lord of the Rings* and *Iron Man*.

created_at
(datetime.datetime) The date and time on which the record was created.

id
(int) The primary key integer.

work_franchises
(collections.abc.MutableSet) The set of *WorkFranchise*s that the franchise has.

works
(collections.abc.MutableSet) The set of *Work*s that belongs to the franchise.

world
(*World*) The world which the franchise belongs to.

world_id
(int) *World.id* of *world*.

class cliche.work.**Genre**(***kwargs*)
Genre of the creative work

created_at
(datetime.datetime) The date and time on which the record was created.

id
(int) The primary key integer.

name
(str) The name of the genre.

work_genres
(collections.abc.MutableSet) The set of *WorkGenre*s that the genre has.

works
(collections.abc.MutableSet) The set of *Work*s that fall into the genre.

class cliche.work.**Role**
Python enum type to describe role of him/her in making a work.

class cliche.work.**Work**(***kwargs*)
Creative work(s) that could be a single work like a film, or a series of works such as a combic book series and a television series.

characters
(collections.abc.MutableSet) The set of *Character*s that appeared in the work.

created_at
(datetime.datetime) The date and time on which the record was created.

credits
(collections.abc.MutableSet) The set of *Credit*s that the work has.

franchises
(collections.abc.MutableSet) The set of *Franchise*s that the work belongs to.

genres
(collections.abc.MutableSet) The set of *Genre*s that the work falls into.

id
(int) The primary key integer.

media_type
(str) Work media type.

published_at
(datetime.date) The publication date.

tropes
(collections.abc.MutableSet) The set of Trope.

work_franchises
(collections.abc.MutableSet) The set of *WorkFranchise*s that the work has.

work_genres
(collections.abc.MutableSet) The set of *WorkGenre*s that the work has.

work_tropes
(collections.abc.MutableSet) The set of WorkTrope.

class cliche.work.**WorkCharacter**(***kwargs*)
Relationship between the character and the work. Describe that the character appeared in the work.

character
(*Character*) The character that appeared in the *work*.

character_id
(int) *Character.id* of *character*.

created_at
(datetime.datetime) The date and time on which the record was created.

work
(*Work*) The work in which the *character* appeared.

work_id
(int) *Work.id* of *work*.

class cliche.work.**WorkFranchise**(***kwargs*)
Relationship between the work and the Franchise.

created_at
(datetime.datetime) The date and time on which the record was created.

franchise
(*Franchise*) The franchise that the *work* belongs to.

franchise_id
(int) *Franchise.id* of *franchise*.

work
(*Work*) The work that belongs to the *franchise*.

work_id
(int) *Work.id* of *work*.

class cliche.work.**WorkGenre**(***kwargs*)
Relationship between the work and the genre.

created_at
(datetime.datetime) The date and time on which the record was created.

genre
(*Genre*) The genre into which the *work* falls.

genre_id
(int) *Genre.id* of *genre*.

work
(*Work*) The work that falls into the *genre*.

work_id
(int) *Work.id* of *work*.

class cliche.work.**World**(***kwargs*)
Fictional universe that is a self-consistent fictional setting with elements that differ from the real world, for example *Middle-earth* and *Marvel Cinematic Universe*.

created_at
(datetime.datetime) The date and time on which the record was created.

franchises
(collections.abc.MutableSet) The set of *Franchise*s that belong the world.

id
(int) The primary key integer.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

C

Symbols

A

B

C

D