hobbit-core Documentation

Legolas Bloom

2023 年 08 月 15 日

Contents

changelog // github // pypi // issues // API 文档 // EN version

基于 Flask + SQLAlchemy + marshmallow + webargs 的 flask 项目生成器。

包含 RESTful API、celery 集成、单元测试、gitlab-ci/cd、docker compose 一套解决方案。后续考虑更好的自动文档工具（目前有 apispec ）。

为什么我开发了这个项目？可以参考这一设计范式：Convention over configuration 。

CHAPTER 1

简易教程

1.1 快速安装

```
pip install "hobbit-core[hobbit,hobbit_core]"  # 安装全部功能
pip install "hobbit-core[hobbit,hobbit_core]" --pre  # 安装pre release版本
# 仅安装命令依赖，不安装库依赖（安装命令到全局时推荐使用）
pip install "hobbit-core[hobbit]"
```

1.2 快速生成项目

使用 hobbit 命令自动生成你的 flask 项目:

```
hobbit --echo new -n demo -d . -p 5000 --celery  # 建议试用 -t rivendell 新模版
```

建议使用 pipenv 创建虚拟环境:

```
pipenv install -r requirements.txt --pre && pipenv install --dev pytest pytest-cov␣
↪pytest-env ipython flake8 ipdb
```

该命令会生成一个完整的 api 及其测试范例，使用以下项目启动 server:

```
pipenv shell  # 使用虚拟环境
FLASK_APP=app/run.py flask run
```

你可以在控制台看到类似如下信息:

```
* Serving Flask app "app/run.py"
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
```

(续下页)

(接上页)

```
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

访问 http://127.0.0.1:5000/api/ping/

1.3 自动补全

```
# bash users add this to your .bashrc
eval "$(_HOBBIT_COMPLETE=source hobbit)"
# zsh users add this to your .zshrc
eval "$(_HOBBIT_COMPLETE=source_zsh hobbit)"
```

CHAPTER 2

项目结构

```
.
├── Dockerfile
├── app
│   ├── __init__.py
│   ├── configs
│   │   ├── __init__.py
│   │   ├── default.py
│   │   ├── development.py
│   │   ├── production.py
│   │   └── testing.py
│   ├── core
│   │   └── __init__.py
│   ├── exts.py
│   ├── models
│   │   └── __init__.py
│   ├── run.py
│   ├── schemas
│   │   └── __init__.py
│   ├── tasks
│   │   └── __init__.py
│   ├── utils
│   │   └── __init__.py
│   └── views
│       ├── __init__.py
│       └── ping.py
├── configs
│   └── gunicorn-logging.ini
├── deploy.sh
├── docker-compose.yml
├── docs
│   └── index.apib
├── pytest.ini
├── requirements.txt
└── tests
```

(续下页)

(接上页)

```
    ├── __init__.py
    ├── conftest.py
    └── test_ping.py
```

2.1 Dockerfile

使用 docker 来运行我们的 web 服务，基于同一个 docker image 运行测试，由此保证开发环境、测试环境、运行时环境一致。你可以在 Dockerfile reference 查看有关 Dockerfile 的语法。

2.2 app

app 文件夹保存了所有业务层代码。基于 **约定优于配置**范式，这个文件夹名字及所有其他文件夹名字 **禁止修改**。

2.3 configs

基于 flask 设计，我们使用环境变量 `FLASK_ENV` 加载不同环境的配置文件。例如 `FLASK_ENV=production`，会自动加载 `configs/production.py` 这个文件作为配置文件。

2.4 core

core 文件夹约定编写自定义的基础类库代码，或者临时扩展 hobbit_core 的基础组件（方便后续直接贡献到 hobbit_core）。

2.5 exts.py

flask 项目很容易产生循环引用问题，`exts.py` 文件的目的就是避免产生这个问题。你可以看下这个解释: Why use exts.py to instance extension?

2.6 models

所有数据库模型定义在这里。

2.7 services

使用 rivendell 模版事会有此模块，类比 java 结构，这时候约定 view 不访问 model 层而去访问 sevices 层，由 sevices 层去访问 model 层。

2.8 run.py

web 项目的入口。你将在这里注册路由、注册命令等等操作。

2.9 schemas

定义所有的 marshmallow scheams。我们使用 marshmallow 来序列化 api 输出，类似 `django-rest-framework` 的效果。

2.10 utils

定义所有的公用工具函数。

2.11 views

路由及简单业务逻辑。

2.11.1 deploy.sh

一个简易的部署脚本。配合 ci/cd 系统一起工作。

2.12 docker-compose.yml

基本的 docker compose 配置文件。考虑到单机部署需求，自动生成了一个简易的配置，启动项目:

```
docker-compose up
```

2.12.1 docs

API 文档、model 设计文档、架构设计文档、需求文档等等项目相关的文档。

2.12.2 logs

运行时的 log 文件。

2.12.3 tests

所有的测试 case. 推荐使用 pytest 测试，项目也会自动生成基本的 pytest 配置。

CHAPTER 3

配置

表 1: Configuration

Key	Value	Description
HOBBIT_USE_CODE_ORIGIN_TYPE	*True* or *False*	Return origin type of code in response. Default is *False*.
HOBBIT_RESPONSE_MESSAGE_MAPS	*dict*, *{code: message}*	Self-defined response message. Set to *{200: "success"}* will return *{"code": 200, "message": "success"}* in response.
HOBBIT_RESPONSE_DETAIL	*True* or *False*	Default return detail and must set to *False* in production env. Default is *True*. Only used in 500 server error response.

CHAPTER 4

Others

4.1 Change history

3.1.1 (2023-08-11)

- Hotfix: lock flask version < 2.3 and pyyaml verison(https://github.com/yaml/pyyaml/issues/724)

3.1.0 (2023-01-29)

- Support HOBBIT_RESPONSE_DETAIL config: Default return detail and must set to False in production env. Default is True. Only used in 500 server error response.

3.0.0 (2022-12-12)

- Upgrade deps: Flask 1.x -> 2.x, SQLAlchemy 1.3.x -> 1.4.x, Flask-SQLAlchemy 2.5.1 -> 3.x.
- **Notice:** [https://docs.sqlalchemy.org/en/14/tutorial/dbapi_transactions.html{]}(https://docs.sqlalchemy.org/en/14/tutorial/dbapi_transactions.html).

4.1.1 2.2.3 (2022-05-18)

- Support use nested=None(*@transaction(db.session, nested=None)*) to avoid bug from *flask_sqlalchemy.models_committed* signal.

4.1.2 2.2.2 (2022-02-17)

- Refactor tpl: Auto nested blueprint.
- Refactor tpl: ping and options api were merged into tools.
- Enhance teardown_method in test: auto handle deps when delete table.
- Fix some typo.

4.1.3 2.2.1 (2021-12-01)

- Add *err_handler.HobbitException*: Base class for all hobbitcore-related errors.

4.1.4 2.2.0 (2021-11-18)

- Support Python 3.10.

4.1.5 2.1.1 (2021-10-25)

- Add util *bulk_create_or_update_on_duplicate*, support MySQL and postgreSQL.

2.1.0 (2021-10-25, unused)

- This filename has already been used (Wrong file pushed to pypi.org).

4.1.6 2.0.4 (2021-07-13)

- Support set *HOBBIT_RESPONSE_MESSAGE_MAPS* to use self-defined response message.

4.1.7 2.0.3 (2021-07-08)

- Fix set response.xxxResult code = 0.

4.1.8 2.0.2 (2021-07-08)

- Fix response message err when code is 200 or 400.
- Support set *HOBBIT_USE_CODE_ORIGIN_TYPE = True* to return origin type of code in response.

4.1.9 2.0.1 (2021-06-21)

- Add data field for response.Result: return Real response payload.
- Bugfix: tests.BaseTest.teardown_method miss *app.app_context()*.

4.1.10 2.0.0 (2021-06-20)

- Upgrade webargs to version 8.x.x.
- Lock SQLAlchemy version less than 1.4.0 (session.autobegin feature doesn't look like a good idea).
- Lock Flask version less than 2.x.x (because some bugs).
- Upgrade and lock marshmallow>=3.0.0,<4.
- Remove hobbit gen cmd.

4.1.11 1.4.4 (2020-03-25)

- Fix webargs 6.x.x: limit version < 6.

4.1.12 1.4.3 (2019-07-24)

- Add CustomParser for automatically trim leading/trailing whitespace from argument values(*from hobbit_core.webargs import use_args, use_kwargs*).
- Add *HOBBIT_UPPER_SEQUENCE_NAME* config for upper db's sequence name.
- Fixs some err in template.

4.1.13 1.4.2 (2019-06-13)

- Add *db.BaseModel* for support Oracle id sequence.

4.1.14 1.4.1 (2019-05-23)

- Add template for 4-layers (view、schema、service、model).
- Add options api for query all consts defined in *app/models/consts*.
- Add *create* command to generate a csv file that defines some models to use in the *gen* command.
- Removed example code.
- Split hobbit cmd and hobbit_core lib, now install cmd should be *pip install "hobbit-core[hobbit,hobbit_core]"*.
- Remove flask_hobbit when import (*hobbit_core.flask_hobbit.db import transaction --> from hobbit_core.db import transaction*).
- Enhance gen cmd: now can auto create CRUD API and tests.
- Fix typo.
- Update some test cases.

4.1.15 1.4.0 (Obsolete version)

4.1.16 1.3.1 (2019-02-26)

- The strict parameter is removed in marshmallow >= 3.0.0.

4.1.17 1.3.0 (2019-01-14)

- Add import_subs util for auto import models、schemas、views in module/__init__.py file.
- Add index for created_at、updated_at cloumn and default order_by id.
- Add validate for PageParams.
- Add hobbit gen cmd for auto render views.py, models.py, schemas.py etc when start a feature dev.
- Add ErrHandler.handler_assertion_error.
- Add db.transaction decorator, worked either autocommit True or False.
- pagination return dict instead of class, order_by can set None for
- traceback.print_exc() --> logging.error.
- Foreign key fields support ondelete, onupdate.
- Hobbit startproject cmd support celery option.

4.1.18 1.2.5 (2018-10-30)

- Add ModelSchema(Auto generate load and dump func for EnumField).
- Add logging config file.
- Add EnumExt implementation.
- Fix use_kwargs with fileds.missing=None and enhanced.

4.1.19 1.2.4 (2018-10-18)

- Fix SuccessResult status arg not used.

4.1.20 1.2.3 (2018-10-18)

- Add utils.use_kwargs, fix webargs's bug.

4.1.21 1.2.2 (2018-10-16)

- Add SchemaMixin & ORMSchema use in combination with db.SurrogatePK.
- Now print traceback info when server 500.
- Fix miss hidden files when sdist.

4.1.22 1.2.1 (2018-10-12)

- secure_filename support py2 & py3.

4.1.23 1.2.0 (2018-10-11)

- Gitlab CI/CD support.
- Add secure_filename util.
- Enhance deploy, can deploy to multiple servers.
- Add --port option for startproject cmd.

4.1.24 1.1.0 (2018-09-29)

- Beta release.
- Fix hobbit create in curdir(.) err.
- Add dict2object util.
- Project tree confirmed.
- Add tutorial、project tree doc.
- Add example options for startproject cmd.

4.1.25 1.0.0 (2018-09-25)

- Alpha release.
- flask_hobbit release.

4.1.26 0.0.[1-9]

- hobbit cmd released.
- Incompatible production version.

4.2 Hobbit-core's API Documentation

4.2.1 hobbit cmd

hobbit - A flask project generator.

`hobbit.bootstrap.`**`new`**(**args: Any*, ***kwargs: Any*) → Any

Create a new flask project, render from different template.

Examples:

```
hobbit --echo new -n blog -d /tmp/test -p 1024
```

It is recommended to use pipenv to create venv:

```
pipenv install -r requirements.txt && pipenv install --dev pytest pytest-cov␣
↪pytest-env ipython flake8 ipdb
```

4.2.2 hobbit_core

A flask extension that take care of base utils.

hobbit_core

Common utils for flask app.

class `hobbit_core.`**`HobbitManager`**(*app=None*, *db=None*, ***kwargs*)

Customizable utils management.

`init_app`(*app*, *db*, ***kwargs*)

app: The Flask application instance.

db

class `hobbit_core.db.`**`BaseModel`**(***kwargs*)

Abstract base model class contains `id`、`created_at` and `updated_at` columns.

id: A surrogate biginteger 'primary key' column.

created_at: Auto save `datetime.now()` when row created.

updated_at: Auto save `datetime.now()` when row updated.

Support **oracle id sequence**, default name is `{class_name}_id_seq`, can changed by `sequence_name` and `HOBBIT_UPPER_SEQUENCE_NAME` config. Default value of app.config['HOBBIT_UPPER_SEQUENCE_NAME'] is False.

Examples:

```
from hobbit_core.db import Column, BaseModel

class User(BaseModel):
    username = Column(db.String(32), nullable=False, index=True)
```

(续下页)

(接上页)

```
print([i.name for i in User.__table__.columns])
# ['username', 'id', 'created_at', 'updated_at']
```

Can be blocked columns with **exclude_columns**:

```
class User(BaseModel):
    exclude_columns = ['created_at', 'updated_at']
    username = Column(db.String(32), nullable=False, index=True)

print([i.name for i in User.__table__.columns])
# ['username', 'id']
```

Can be changed primary_key's name using **primary_key_name**:

```
class User(BaseModel):
    primary_key_name = 'user_id'
    username = Column(db.String(32), nullable=False, index=True)

print([i.name for i in User.__table__.columns])
# ['username', 'user_id', 'created_at', 'updated_at']
```

Can be changed sequence's name using **sequence_name** (worked with oracle):

```
class User(BaseModel):
    sequence_name = 'changed'
    username = Column(db.String(32), nullable=False, index=True)

# print(User.__table__.columns['id'])
Column('id', ..., default=Sequence('changed_id_seq'))
```

__repr__() → str

You can set label property.

返回

<{classname}({pk}:{label!r})>

返回类型

str

class hobbit_core.db.**SurrogatePK**

A mixin that add `id`、`created_at` and `updated_at` columns to any declarative-mapped class.

id: A surrogate biginteger 'primary key' column.

created_at: Auto save `datetime.now()` when row created.

updated_at: Auto save `datetime.now()` when row updated.

It is not recommended. See hobbit_core.db.BaseModel.

__repr__() → str

You can set label property.

返回

<{classname}({pk}:{label!r})>

返回类型

str

class hobbit_core.db.**EnumExt**(*value*)

Extension for serialize/deserialize sqlalchemy enum field.

Be sure `type(key)` is `int` and `type(value)` is `str` (`label = (key, value)`).

Examples:

```
class TaskState(EnumExt):
    # label = (key, value)
    CREATED = (0, '新建')
    PENDING = (1, '等待')
    STARTING = (2, '开始')
    RUNNING = (3, '运行中')
    FINISHED = (4, '已完成')
    FAILED = (5, '失败')
```

classmethod strict_dump(*label: str*, *verbose: bool = False*) → Union[int, str]

Get key or value by label.

Examples:

```
TaskState.strict_dump('CREATED')  # 0
TaskState.strict_dump('CREATED', verbose=True)  # '新建'
```

返回

Key or value, If label not exist, raise `KeyError`.

返回类型

int|str

classmethod dump(*label: str*, *verbose: bool = False*) → Dict[str, Any]

Dump one label to option.

Examples:

```
TaskState.dump('CREATED')  # {'key': 0, 'value': '新建'}
```

返回

Dict of label's key and value. If label not exist, raise `KeyError`.

返回类型

dict

classmethod load(*val: Union[int, str]*) → str

Get label by key or value. Return val when val is label.

Examples:

```
TaskState.load('FINISHED')  # 'FINISHED'
TaskState.load(4)  # 'FINISHED'
TaskState.load('新建')  # 'CREATED'
```

返回

Label.

返回类型

str|None

classmethod to_opts(*verbose: bool = False*) → List[Dict[str, Any]]

Enum to options.

Examples:

```
opts = TaskState.to_opts(verbose=True)
print(opts)

[{'key': 0, 'label': 'CREATED', 'value': u'新建'}, ...]
```

返回

List of dict which key is *key*, *value*, label.

返回类型

list

classmethod strict_dump(*label: str*, *verbose: bool = False*) → Union[int, str]

Get key or value by label.

Examples:

```
TaskState.strict_dump('CREATED')  # 0
TaskState.strict_dump('CREATED', verbose=True)  # '新建'
```

返回

Key or value, If label not exist, raise `KeyError`.

返回类型

int|str

classmethod dump(*label: str*, *verbose: bool = False*) → Dict[str, Any]

Dump one label to option.

Examples:

```
TaskState.dump('CREATED')  # {'key': 0, 'value': '新建'}
```

返回

Dict of label's key and value. If label not exist, raise `KeyError`.

返回类型

dict

classmethod load(*val: Union[int, str]*) → str

Get label by key or value. Return val when val is label.

Examples:

```
TaskState.load('FINISHED')  # 'FINISHED'
TaskState.load(4)  # 'FINISHED'
TaskState.load('新建')  # 'CREATED'
```

返回

Label.

返回类型

str|None

classmethod to_opts(*verbose: bool = False*) → List[Dict[str, Any]]

Enum to options.

Examples:

```
opts = TaskState.to_opts(verbose=True)
print(opts)

[{'key': 0, 'label': 'CREATED', 'value': u'新建'}, ...]
```

返回

List of dict which key is *key*, *value*, label.

返回类型

list

class hobbit_core.db.**BaseModelMeta**(*name*, *bases*, *attrs*)

metadata: sa.MetaData

class hobbit_core.db.**BaseModel**(***kwargs*)

Abstract base model class contains `id`、`created_at` and `updated_at` columns.

id: A surrogate biginteger 'primary key' column.

created_at: Auto save `datetime.now()` when row created.

updated_at: Auto save `datetime.now()` when row updated.

Support **oracle id sequence**, default name is `{class_name}_id_seq`, can changed by `sequence_name` and `HOBBIT_UPPER_SEQUENCE_NAME` config. Default value of app.config['HOBBIT_UPPER_SEQUENCE_NAME'] is False.

Examples:

```
from hobbit_core.db import Column, BaseModel

class User(BaseModel):
    username = Column(db.String(32), nullable=False, index=True)

print([i.name for i in User.__table__.columns])
# ['username', 'id', 'created_at', 'updated_at']
```

Can be blocked columns with **exclude_columns**:

```
class User(BaseModel):
    exclude_columns = ['created_at', 'updated_at']
    username = Column(db.String(32), nullable=False, index=True)

print([i.name for i in User.__table__.columns])
# ['username', 'id']
```

Can be changed primary_key's name using **primary_key_name**:

```
class User(BaseModel):
    primary_key_name = 'user_id'
    username = Column(db.String(32), nullable=False, index=True)

print([i.name for i in User.__table__.columns])
# ['username', 'user_id', 'created_at', 'updated_at']
```

Can be changed sequence's name using **sequence_name** (worked with oracle):

```
class User(BaseModel):
    sequence_name = 'changed'
    username = Column(db.String(32), nullable=False, index=True)

# print(User.__table__.columns['id'])
Column('id', ..., default=Sequence('changed_id_seq'))
```

query: t.ClassVar[Query]

A SQLAlchemy query for a model. Equivalent to `db.session.query(Model)`. Can be customized per-model by overriding `query_class`.

> **警告:** The query interface is considered legacy in SQLAlchemy. Prefer using `session.execute(select())` instead.

hobbit_core.db.**reference_col**(*tablename: str*, *nullable: bool = False*, *pk_name: str = 'id'*, *onupdate: Optional[str] = None*, *ondelete: Optional[str] = None*, ***kwargs: Any*) → Column

Column that adds primary key foreign key reference.

参数

- **tablename** (*str*) -- Model.__table_name__.
- **nullable** (*bool*) -- Default is False.
- **pk_name** (*str*) -- Primary column's name.
- **onupdate** (*str*) -- If Set, emit ON UPDATE <value> when issuing DDL for this constraint. Typical values include CASCADE, DELETE and RESTRICT.
- **ondelete** (*str*) -- If set, emit ON DELETE <value> when issuing DDL for this constraint. Typical values include CASCADE, DELETE and RESTRICT.

Others:

See `sqlalchemy.Column`.

Examples:

```
from sqlalchemy.orm import relationship

role_id = reference_col('role')
role = relationship('Role', backref='users', cascade='all, delete')
```

class hobbit_core.db.**OptType**(**args*, ***kwargs*)

key: int

value: str

label: str

hobbit_core.db.**transaction**(*session: Session*, *nested: bool = False*)

SQLAlchemy 1.4 deprecates "autocommit mode. See more: https://docs.sqlalchemy.org/en/14/orm/session_transaction.html

2022-05-18 Updated: Use *nested=None* to prevent signal bug, See more: https://github.com/pallets-eco/flask-sqlalchemy/issues/645

Tips:

- **Can't** do `session.commit()` **in func**,

otherwise unknown beloved.

- **Must use the same session in decorator and decorated function**.
- **We can use nested** if keep top decorated by `@transaction(session, nested=False)` and all subs decorated by `@transaction(session, nested=True)`.

Examples:

```
from hobbit_core.db import transaction

from app.exts import db


@bp.route('/users/', methods=['POST'])
@transaction(db.session)
def create(username, password):
    user = User(username=username, password=password)
    db.session.add(user)
    # db.session.commit() error
```

We can nested use this decorator. Must set `nested=True` otherwise raise `ResourceClosedError` (session.autocommit=False) or raise `InvalidRequestError` (session.autocommit=True):

```
@transaction(db.session, nested=True)
def set_role(user, role):
    user.role = role
    # db.session.commit() error


@bp.route('/users/', methods=['POST'])
@transaction(db.session)
def create(username, password):
    user = User(username=username, password=password)
    db.session.add(user)
    db.session.flush()
    set_role(user, 'admin')
```

pagination

hobbit_core.pagination.**PageParams = {'order_by': <fields.DelimitedList(dump_default=<marshmallow.missing>, attribute=None, validate=None, required=False, load_only=False, dump_only=False, load_default=['-id'], allow_none=False, error_messages={'required': 'Missing data for required field.', 'null': 'Field may not be null.', 'validator_failed': 'Invalid value.', 'invalid': 'Not a valid delimited list.'})>, 'page': <fields.Integer(dump_default=<marshmallow.missing>, attribute=None, validate=<Range(min=1, max=2147483648, min_inclusive=True, max_inclusive=True, error=None)>, required=False, load_only=False, dump_only=False, load_default=1, allow_none=False, error_messages={'required': 'Missing data for required field.', 'null': 'Field may not be null.', 'validator_failed': 'Invalid value.', 'invalid': 'Not a valid integer.', 'too_large': 'Number too large.'})>, 'page_size': <fields.Integer(dump_default=<marshmallow.missing>, attribute=None, validate=<Range(min=5, max=100, min_inclusive=True, max_inclusive=True, error=None)>, required=False, load_only=False, dump_only=False, load_default=10, allow_none=False, error_messages={'required': 'Missing data for required field.', 'null': 'Field may not be null.', 'validator_failed': 'Invalid value.', 'invalid': 'Not a valid integer.', 'too_large': 'Number too large.'})>}**

Base params for list view func which contains `page`、`page_size`、`order_by` params.

Example:

```
@use_kwargs(PageParams)
def list_users(page, page_size, order_by):
    pass
```

class hobbit_core.pagination.**PaginationType**(**args*, ***kwargs*)

page: int

page_size: int

total: int

hobbit_core.pagination.**pagination**(*obj: DefaultMeta*, *page: int*, *page_size: int*, *order_by: Optional[Union[str, List[str]]] = 'id'*, *query_exp=None*) → *PaginationType*

A pagination for sqlalchemy query.

参数

- **obj** (*db.Model*) -- Model class like User.
- **page** (*int*) -- Page index.
- **page_size** (*int*) -- Row's count per page.
- **order_by** (*str, list, None*) -- Example: 'id'、['-id', 'column_name'].
- **query_exp** (*flask_sqlalchemy.BaseQuery*) -- Query like `User.query.filter_by(id=1)`.

返回

Dict contains `items`、`page`、`page_size` and `total` fileds.

返回类型
dict

schemas

class hobbit_core.schemas.**ORMSchema**(**args*, ***kwargs*)

Base schema for ModelSchema. See webargs/issues/126.

Example:

```
from hobbit_core.schemas import ORMSchema


class UserSchema(ORMSchema):

    class Meta:
        model = User
        load_only = ('password')
```

`@use_kwargs(UserSchema())` use in combination with `load_only`:

```
@bp.route('/users/', methods=['POST'])
@use_kwargs(UserSchema())
def create_user(username, password):
    pass
```

opts = <flask_marshmallow.sqla.SQLAlchemyAutoSchemaOpts object>

class hobbit_core.schemas.**SchemaMixin**

Add `id`, `created_at`, `updated_at` fields to schema, default `dump_only=True`.

Example:

```
from marshmallow import Schema

from hobbit_core.schemas import SchemaMixin

class UserSchema(Schema, SchemaMixin):
    pass
```

class hobbit_core.schemas.**PagedSchema**(**, only: types.StrSequenceOrSet | None = None, exclude: types.StrSequenceOrSet = (), many: bool = False, context: dict | None = None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: bool | types.StrSequenceOrSet = False, unknown: str | None = None*)

Base schema for list api pagination.

Example:

```
from marshmallow import fields

from hobbit_core.schemas import PagedSchema

from . import models
from .exts import ma
```

(续下页)

(接上页)

```
class UserSchema(ma.ModelSchema):

    class Meta:
        model = models.User


class PagedUserSchema(PagedSchema):
    items = fields.Nested('UserSchema', many=True)


paged_user_schemas = PagedUserSchema()
```

class Meta

class hobbit_core.schemas.**ModelSchema**(**args*, ***kwargs*)

Base ModelSchema for `class Model(db.SurrogatePK)`.

- Auto generate load and dump func for EnumField.
- Auto dump_only for `id`, `created_at`, `updated_at` fields.
- Auto set dateformat to `'%Y-%m-%d %H:%M:%S'`.
- Auto use verbose for dump EnumField. See `db.EnumExt`. You can define verbose in `Meta`.

Example:

```
class UserSchema(ModelSchema):
    role = EnumField(RoleEnum)

    class Meta:
        model = User

data = UserSchema().dump(user).data
assert data['role'] == {'key': 1, 'label': 'admin', 'value': '管理员'}
```

opts = <flask_marshmallow.sqla.SQLAlchemyAutoSchemaOpts object>

utils

class hobbit_core.utils.**ParamsDict**

Just available update func.

Example:

```
@use_kwargs(PageParams.update({...}))
def list_users(page, page_size, order_by):
    pass
```

update(*other=None*)

Update self by other Mapping and return self.

class hobbit_core.utils.**dict2object**

Dict to fake object that can use getattr.

Examples:

```
In [2]: obj = dict2object({'a': 2, 'c': 3})

In [3]: obj.a
Out[3]: 2

In [4]: obj.c
Out[4]: 3
```

hobbit_core.utils.**secure_filename**(*filename: str*) → str

Borrowed from werkzeug.utils.secure_filename.

Pass it a filename and it will return a secure version of it. This filename can then safely be stored on a regular file system and passed to `os.path.join()`.

On windows systems the function also makes sure that the file is not named after one of the special device files.

```
>>> secure_filename(u'哈哈.zip')
'哈哈.zip'
>>> secure_filename('My cool movie.mov')
'My_cool_movie.mov'
>>> secure_filename('../../../etc/passwd')
'etc_passwd'
>>> secure_filename(u'i contain cool ümläuts.txt')
'i_contain_cool_umlauts.txt'
```

hobbit_core.utils.**use_kwargs**(*argmap*, *schema_kwargs: Optional[Dict] = None*, ***kwargs: Any*)

For fix `Schema(partial=True)` not work when used with `@webargs.flaskparser.use_kwargs`. More details `see webargs.core`.

参数

- **argmap** (*marshmallow.Schema, dict, callable*) -- Either a *marshmallow.Schema*, *dict* of argname -> *marshmallow.fields.Field* pairs, or a callable that returns a *marshmallow.Schema* instance.
- **schema_kwargs** (*dict*) -- kwargs for argmap.

返回

A dictionary of parsed arguments.

返回类型

dict

hobbit_core.utils.**import_subs**(*locals_*, *modules_only: bool = False*) → List[str]

Auto import submodules, used in __init__.py.

参数

- **locals** -- *locals()*.
- **modules_only** -- Only collect modules to __all__.

Examples:

```
# app/models/__init__.py
from hobbit_core.utils import import_subs

__all__ = import_subs(locals())
```

Auto collect Model's subclass, Schema's subclass and instance. Others objects must defined in submodule.__all__.

hobbit_core.utils.**bulk_create_or_update_on_duplicate**(*db*, *model_cls*, *items*, *updated_at='updated_at'*, *batch_size=500*)

Support MySQL and postgreSQL. https://dev.mysql.com/doc/refman/8.0/en/insert-on-duplicate.html

参数

- **db** -- Instance of *SQLAlchemy*.
- **model_cls** -- Model object.
- **items** -- List of data,[example: *[{key: value}, {key: value}, ...]*.
- **updated_at** -- Field which recording row update time.
- **batch_size** -- Batch size is max rows per execute.

返回

A dictionary contains rowcount and items_count.

返回类型

dict

response

class hobbit_core.response.**RespType**(**args*, ***kwargs*)

code: str

message: str

detail: Any

hobbit_core.response.**gen_response**(*code: int*, *message: Optional[str] = None*, *detail: Optional[str] = None*, *data=None*) → *RespType*

Func for generate response body.

参数

- **code** (*string, int*) -- Extension to interact with web pages. Default is http response `status_code` like 200、404.
- **message** (*string*) -- For popup windows.
- **data** (*object*) -- Real response payload.
- **detail** (*object*) -- For debug, detail server error msg.

返回

A dict contains all args.

返回类型

dict

2021-07-08 Updated:

Default type of *code* in response is force conversion to *str*, now support set *HOBBIT_USE_CODE_ORIGIN_TYPE = True* to return origin type.

2021-07-13 Updated:

Support set *HOBBIT_RESPONSE_MESSAGE_MAPS* to use self-defined response message. *HOBBIT_RESPONSE_MESSAGE_MAPS* must be dict.

class hobbit_core.response.**Result**(*response=None*, *status=None*, *headers=None*, *mimetype='application/json'*, *content_type=None*, *direct_passthrough=False*)

Base json response.

response: Union[Iterable[str], Iterable[bytes]]

The response body to send as the WSGI iterable. A list of strings or bytes represents a fixed-length response, any other iterable is a streaming response. Strings are encoded to bytes as UTF-8.

Do not set to a plain string or bytes, that will cause sending the response to be very inefficient as it will iterate one byte at a time.

class hobbit_core.response.**SuccessResult**(*message: str = ''*, *code: Optional[int] = None*, *detail: Optional[Any] = None*, *status: Optional[int] = None*, *data=None*)

Success response. Default status is 200, you can cover it by status arg.

response: Union[Iterable[str], Iterable[bytes]]

The response body to send as the WSGI iterable. A list of strings or bytes represents a fixed-length response, any other iterable is a streaming response. Strings are encoded to bytes as UTF-8.

Do not set to a plain string or bytes, that will cause sending the response to be very inefficient as it will iterate one byte at a time.

headers: Headers

class hobbit_core.response.**FailedResult**(*message: str = ''*, *code: Optional[int] = None*, *detail: Optional[Any] = None*)

Failed response. status always 400.

response: Union[Iterable[str], Iterable[bytes]]

The response body to send as the WSGI iterable. A list of strings or bytes represents a fixed-length response, any other iterable is a streaming response. Strings are encoded to bytes as UTF-8.

Do not set to a plain string or bytes, that will cause sending the response to be very inefficient as it will iterate one byte at a time.

headers: Headers

class hobbit_core.response.**UnauthorizedResult**(*message: str = ''*, *code: Optional[int] = None*, *detail: Optional[Any] = None*)

response: Union[Iterable[str], Iterable[bytes]]

The response body to send as the WSGI iterable. A list of strings or bytes represents a fixed-length response, any other iterable is a streaming response. Strings are encoded to bytes as UTF-8.

Do not set to a plain string or bytes, that will cause sending the response to be very inefficient as it will iterate one byte at a time.

headers: Headers

class hobbit_core.response.**ForbiddenResult**(*message: str = ''*, *code: Optional[int] = None*, *detail: Optional[Any] = None*)

response: Union[Iterable[str], Iterable[bytes]]

The response body to send as the WSGI iterable. A list of strings or bytes represents a fixed-length response, any other iterable is a streaming response. Strings are encoded to bytes as UTF-8.

Do not set to a plain string or bytes, that will cause sending the response to be very inefficient as it will iterate one byte at a time.

headers: Headers

class hobbit_core.response.**ValidationErrorResult**(*message: str = ''*, *code: Optional[int] = None*, *detail: Optional[Any] = None*)

response: Union[Iterable[str], Iterable[bytes]]

The response body to send as the WSGI iterable. A list of strings or bytes represents a fixed-length response, any other iterable is a streaming response. Strings are encoded to bytes as UTF-8.

Do not set to a plain string or bytes, that will cause sending the response to be very inefficient as it will iterate one byte at a time.

headers: Headers

class hobbit_core.response.**ServerErrorResult**(*message: str = ''*, *code: Optional[int] = None*, *detail: Optional[Any] = None*)

response: Union[Iterable[str], Iterable[bytes]]

The response body to send as the WSGI iterable. A list of strings or bytes represents a fixed-length response, any other iterable is a streaming response. Strings are encoded to bytes as UTF-8.

Do not set to a plain string or bytes, that will cause sending the response to be very inefficient as it will iterate one byte at a time.

headers: Headers

err_handler

exception hobbit_core.err_handler.**HobbitException**

Base class for all hobbitcore-related errors.

class hobbit_core.err_handler.**ErrHandler**

Base error handler that catch all exceptions. Be sure response is:

```
{
    "code": "404",  # error code, default is http status code, you can change it
    "message": "Not found",  # for alert in web page
    "detail": "id number field length must be 18",  # for debug
}
```

Examples:

```
app.register_error_handler(Exception, ErrHandler.handler)
```

Python 模块索引

h

索引

L

M

N

O

P

Q

R

S

T

U

V

模块