dragonfly

Release 1.0

Mar 04, 2020

Contents:

1 Installation **1**
- 1.1 Installing Python . . . 1
- 1.2 Installing Dragonfly . . . 1
- 1.3 Installing requirements . . . 1
- 1.4 Setting up a MySQL database . . . 2
- 1.5 Running dragonfly . . . 2
- 1.6 Usage . . . 2

2 Quick Start **3**
- 2.1 Design Philosophy . . . 3
- 2.2 Routing . . . 3
- 2.3 Controllers . . . 5
- 2.4 Middleware . . . 5
- 2.5 Database . . . 7
- 2.6 Models (ORM) . . . 7
- 2.7 Templates . . . 8
- 2.8 Builder.py (CLI) . . . 10
- 2.9 Config . . . 10
- 2.10 Demo App . . . 10

3 API Reference **11**
- 3.1 Database . . . 11
- 3.2 Middleware . . . 22
- 3.3 Routes . . . 23
- 3.4 Templates . . . 25
- 3.5 Web . . . 25
- 3.6 Other . . . 28

Python Module Index **29**

Index **31**

CHAPTER 1

Installation

1.1 Installing Python

For dragonfly to work you must have Python 3.6 or above installed. For further details on how to do this please see here.

1.2 Installing Dragonfly

First, download the Dragonfly repository

- Via git

  ```
  git clone git@github.com:MattJMLewis/dragonfly-app.git
  ```

- Via GitHub

  Simply download the repository and unzip it.

1.3 Installing requirements

Next, enter the dragonfly directory and run the following command.

```
pip install -r requirements.txt
```

On Windows you may encounter an error installing `mysqlclient`. If this happens you can download the latest `.whl` here. There is also an unoffical repo of `.whl` files here (recommended source). To install simply run `pip install name-of-the-whl-file.whl`.

After this, run the following command to create the necessary directories for `dragonfly` to work.

```
python builder.py setup
```

1.4 Setting up a MySQL database

Dragonfly currently only supports `MySQL` databases. See here for more details on how to set up a server.

1.5 Running dragonfly

Simply run the `main.py` file. Before doing this you should modify the `config.py` file to match your setup. As `dragonfly` implements the Python WSGI interface any WSGI server should work with the application.

1.6 Usage

Please see *here* for the quick start guide on using dragonfly.

CHAPTER 2

Quick Start

This guide assumes you already have Dragonfly installed. Please see the *installation* section if you do not.

2.1 Design Philosophy

Dragonfly is based on the model-view-controller architectural pattern. This means that the model structure, application logic and user interface are divided into separate components. As this is a brief quick start guide some features won't be shown. To get a full list of features please see the *API reference*.

This quick guide details some of the code in the demo project that is displayed on this site.

2.2 Routing

Routing allows Dragonfly to know the location to send a HTTP request. In the example below the GET request to `/users/<id:int>` is routed to the `show` function on `UserController`. This is all registered in the `routes.py` file.

```
from dragonfly.routes import routes

routes.get('users/<id:int>', 'UserController@show')
```

2.2.1 HTTP Methods

Dragonfly supports the following HTTP methods:

HTTP Method	Router method
GET	`.get()`
POST	`.post()`
PUT	`.put()`
PATCH	`.patch()`
OPTIONS	`.options()`
DELETE	`.delete()`
All of the above	`.any()`

2.2.2 Resource Routing

The `Router` class also has a special method called `resource`. Its second argument is an entire controller. Here is an example:

```
routes.resource('articles', 'ArticleController') # Notice there is no @{method_name}
```

Calling `.resource('ArticleController')` will register the following routes:

Route	HTTP Method	Controller function	Description
`/articles`	GET	`ArticleController@index`	Return all articles in the database.
`/articles/<id:int>`	GET	`ArticleController@show`	Return the article with the given id or fail.
`/articles/create`	GET	`ArticleController@create`	Show the form to create a new article.
`/articles`	POST	`ArticleController@store`	Store (and validate) the given values in the POST request.
`/articles/<id:int>/edit`	GET	`ArticleController@edit`	Show the form to edit a article.
`/articles/<id:int>`	PUT	`ArticleController@update`	Update the given article using the given values.
`/articles/<id:int>`	DELETE	`ArticleController@delete`	Delete the given article.

2.2.3 Route Parameters

Route parameters are a way of passing variables to the controller to help find a certain piece of data. In the example above if the URL `/articles/1` was called, the integer `1` would be passed to the controller `show` function through a variable named `id`. This allows for the look up and return of the given article from the database. A route parameter should follow the pattern below:

```
<name_of_variable:expected_type>
```

It is possible to have multiple parameters on a route. For example:

```
routes.get('/articles/<id:int>/<comment_id:int>')
```

Dragonfly supports the following types by default:

Type	Regex
`int`	`([0-9]+)`
`str`	`(.+)`

Custom types

It is very easy to define your own custom types. Simply add a new key (name of the type), value (regex to match) pair in the `PYTHON_TO_REGEX`dictionary in `config.py`. For example:

```
PYTHON_TO_REGEX = {"int": "([0-9]+)", "str": "(.+)",
                   "str_capitalised": "(\b[A-Z].*?\b)"}
```

2.3 Controllers

A controller should contain all of your application logic to do with that resource. The following command will create a file in the `controllers` directory called `article_controller`:

```
python builder.py generate --type=controller article_controller
```

Each time a request is routed a new instance of the registered controller will be instantiated and the registered function run.

The following is the basic structure of a controller:

```
from dragonfly import Auth, view
from models.article import Article

class ArticleController:

    def show(self, id):
        return View('articles.show', article=Article().find(id), user=Auth.user())
```

The following route would match to this controller method:

```
routes.get('/articles/<id:int>', 'ArticleController@show')
```

A controller method should always return a `Response` class of some sort.

2.4 Middleware

Middleware provides a way to stop or modify a request cycle. This can occur before the request is routed, after a response is returned from the controller or both. Dragonfly comes with a few premade middleware such as CSRF protection and a user middleware. You can also create your own middleware using the following command:

```
python builder.py generate --type=middleware article_middleware
```

The following is an example of middleware that will run on any route that resolves the `show` method in the `ArticleController`. It is possible to assign a middleware to multiple actions by appending to the `actions` list. The `before` method here uses the singleton of the `DeferredResponse` class to set the header for the response before it has been generated (NOTE: This does **not** set the headers for any `Response` other than the one returned by the controller).

In the `before` and `after` method if any `Response` class or child of the `Response` class is returned the processing of the request will stop and the response returned.

```
from dragonfly import request
from dragonfly import ErrorResponse, deferred_response

class ArticleMiddleware:

    actions = ['ArticleController@show']

    def before(self):
        if visited in request.cookies:
            return ErrorResponse(404, "You have already visited the page.")

        deferred_response.header('Set-Cookie', 'visited=True')


    def after(self):
        pass
```

Below is an example of the CSRF protection middleware class that comes pre-packaged with Dragonfly.

Warning: Please note that in any `Middleware` class object any packages imported from the framework must be imported by their full import path. This is as the actual middleware file is executed in the package middleware directory.

```
from dragonfly.constants import DATA_METHODS
from dragonfly.request import request
from dragonfly.response import ErrorResponse
from dragonfly.auth import Auth
from config import NO_AUTH

import os


class CsrfMiddleware:
    actions = '*'

    def before(self):
        # Determine if csrf_token for form request is valid
        if request.method in DATA_METHODS and request.path not in NO_AUTH:

            try:
                token = request.get_data()['csrf_token']
            except KeyError:
                return ErrorResponse('No CSRF token', 500)

            if token != Auth.get('csrf_token'):
                return ErrorResponse('CSRF invalid', 500)

        # Set a csrf_token for the form request
        elif request.path not in NO_AUTH:
            Auth.set('csrf_token', os.urandom(25).hex())

    def after(self):
        pass
```

2.5 Database

The database module provides any easy way to interact with the configured `MySQL` database. It simply provides Python functions that are equivalent to most commonly used SQL commands.

The code below demonstrates some of its usage (note this code is not present in the actual demo application)

```
res = DB('articles').select('name').where('name', '=', 'Testing').first()
```

This will generate and execute the following SQL code:

```
SELECT 'name' FROM `articles` WHERE 'name' = 'testing';
```

2.6 Models (ORM)

Models provide an easy way to read, write and update a table in the database through a Python class. To start using the ORM you first need to define the attributes of a model. This is all done through a model class file. This can be generated using the CLI:

```
python builder.py generate --type=model article
```

A new file will be created in the `models` directory. Below is an example of an articles model and the SQL it generates.

```
from dragonfly import models

class Article(models.Model):

    name = models.VarCharField(length=255)
    text = models.TextField()
    user_id = models.IntField(unsigned=True)

    class Meta:
        article_user_fk = models.ForeignKey('user_id').references('id').on('users')

    def url(self):
    return f"{URL}/articles/{self.id}"
```

There are many field types and options for each field type. For an exhaustive list of these please see the *API reference*. It is also important to note that you can add any function you would like to the model class. For example a way to generate the slug for an article:

```
def url(self):
    return f"/articles/{self.id}"
```

This is also where relationships are defined. The following code would be used to define a one-to-many relationship with the `Comments` class and a many-to-one relationship with the `User` class.

```
def comments(self):
    return self.add_relationship(models.HasMany(target='comment'))

def user(self):
    return self.add_relationship(models.BelongsTo(target='user'))
```

Once you have defined the model you need to generate and execute the SQL to create the needed tables. To do this simply run the following command.

```
python builder.py migrate
```

Once complete you should be able to manipulate the newly created `articles` table through the `Article` model. Below is an example of retrieving all articles in the database:

```
from models.article import Article

articles = Article().get()
```

The ORM has a large number of methods that are all listed in the *API reference*.

To interact with the relationships defined in the class simply call the defined functions.

```
# Returns a list of ``Comment`` objects that belong to the given ``Article`` class
comments = Article().first().comments()

# Returns the ``User`` object that this ``Article`` belongs to.
user = Article().first().user()
```

2.7 Templates

Dragonfly provides an easier way to join Python and HTML by using a templating system. A template is stored in the `templates` directory and should be a `html`file. The templates can also be in subdirectories of the `templates` directory.

Below is an example of a html file saved in `templates/articles/index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Articles</title>
    <link rel="stylesheet" href="https://bootswatch.com/4/materia/bootstrap.min.css">
</head>
<body>
<div class="navbar navbar-expand-lg fixed-top navbar-dark bg-primary">
    <div class="container">
        <a href="{{ Utils.url('') }}" class="navbar-brand">Dragonfly</a>
        <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item">
                    <a class="nav-link" href="{{ Utils.url('articles') }}">Articles</
↪a>
                </li>
            </ul>
            <form class="form-inline my-2 my-lg-0" method="POST" action="{{ Utils.url(
↪'logout') }}">
                <input type="hidden" name="csrf_token" value="{{ Auth.get('csrf_token
↪') )}}">
                <button class="btn btn-secondary my-2 my-sm-0" type="submit">Log out</
↪button>
            </form>
        </div>
    </div>
</div>
```

(continues on next page)

(continued from previous page)

```
<div class="container mt-5 pt-5">
    <div class="row">
        <div class="col-lg-12">
            <div class="card border-secondary mb-3">
                <div class="card-header">Articles</div>
                <div class="card-body">
                    <div class="list-group list-group-flush">
                        @if(articles is None)
                            No articles
                        @else
                            @for(article in articles)
                            <a href="{{ $article.url()$ }}"
                               class="list-group-item list-group-item-action flex-
↪column align-items-start">
                                <div class="d-flex w-100 justify-content-between">
                                    <h5 class="mb-1">{{ $article.name$ }}</h5>
                                </div>
                                <p class="mb-1">{{ $article.text$ }}</p>
                            </a>
                            @endfor
                        @endif
                    </div>
                </div>
            </div>
            <a href="{{ Utils.url('articles/create') }}"><button type="button" class=
↪"btn btn-primary btn-lg btn-block">Create an Article</button></a>

            @if(pagination is not None)
                @if(pagination['last_page'] != 1)
                <div class="btn-toolbar justify-content-center" role="toolbar">
                    <div class="btn-group mr-2" role="group" aria-label="First group">
                        @for(i in range(0, pagination['last_page']))
                        <a href="{{ Utils.url('articles?page=' + str(i + 1)) }}">
                            <button type="button" class="btn btn-secondary">{{ $(i +␣
↪1)$ }}</button>
                        </a>
                        @endfor
                    </div>
                </div>
                @endif
            @endif
        </div>
    </div>
</div>
</body>
</html>
```

To call and render this view you would write the following in your controller:

```
articles = Article().paginate(size=5)
return view('articles.index', articles=articles[0], pagination=articles[1])
```

There are a few important things to note about the syntax of the templating system:

- To write variables simply wrap `{{ }}` around the variable name.
- Due to the way the template compiler works if the variable is one 'generated' by a for loop, like the `article` variable in the example above, it must also be wrapped by `$ $`.

2.8 Builder.py (CLI)

The builder (CLI) can be called by executing the following:

```
python builder.py [command name] --[option]=[value] [argument]
```

Below is an exhaustive list of all commands currently implemented:

Command	Option	Accepted values	Argument	Accepted values	Role
setup	None	None	None	None	Creates the required directories for the application to work (controllers, models, storage, middleware and templates)
generate	type	model, middleware or controller	name	str	according to the PEP 8 naming scheme (snake case). Please note that the file names are converted to camel case for the class names.
migrate	None	None	None	None	Generates (and executes) the SQL to create the tables for all developer created models.
drop	None	None	None	None	Drops all tables that correspond to defined model classes.
auth	None	None	None	None	Generates the authentication scaffold for the given project.

2.9 Config

Detailed below is an exhaustive list of configuaration options that can be put in 'config.py':

Variable name	Type	Function
ROOT_DIR	Str	This is the root directory of the project. This value should not be changed.
MIDDLEWARE	List	A list of middleware to be registered. To register a middleware file dot notation should be used. For example to register a middleware file called 'user_middleware' that is in the 'middleware' directory the following should be added to the list (as a string): 'middleware.user_middleeware'
NO_AUTH	List	A list of routes that do not require the user to be authenticated.
PYTHON_DOT_REGEX	Dict	A dictionary that contains mappings from route parameter definitions to regular expression. Any new mapping added can be used in the router.
URL	Str	The root URL of the application.
DATABASE	Dict	A dictionary containing the configuration settings for the database.

2.10 Demo App

Please see here for further code from the example project.

CHAPTER 3

API Reference

3.1 Database

3.1.1 DB

class dragonfly.db.database.**DB**(*database_settings=<sphinx.ext.autodoc.importer._MockObject object>*)

Bases: object

An easy way to interact with the configured database.

chunk(*chunk_loc*, *chunk_size*)

This will run the given query and return the given number of results at the given location.

Parameters

- **chunk_loc** (*int*) – The location to chunk e.g the first chunk or second chunk.
- **chunk_size** (*int*) – The number of rows each chunk should contain.

comparison_operators = ['=', '<=>', '<>', '!=', '>', '>=', '<', '<=', 'IN()', 'NOT',

custom_sql(*sql*, *n_rows=None*)

Execute the custom SQL passed to the function.

Parameters

- **sql** (*str*) – The SQL code to execute
- **n_rows** (*int*) – The number of rows to retrieve. If set to *None* returns all rows

Returns The result of the SQL executed

Return type dict

delete()

Deletes the given row/rows baed on the developer defined query.

For this method to run the *where* method must have been called first.

first()
This will execute the developer defined query and return only the first result (uses `LIMIT 1`)

get()
This will execute the developer defined query and return all results.

insert(*insert_dict*)
Inserts the given values into the database.

Parameters **insert_dict** (*dict*) – The dictionary containing the column and the value to insert into the specified table

multiple_where(*where_dict*)
Allows for multiple where clauses through one command. Note this only supports the = operator.

Parameters **where_dict** (*dict*) – The values to match

select(**args*)
Equivalent to the `SELECT` command in MySQL.

Pass all the columns you want to select to the function as normal arguments.

Example `DB().select('title', 'text')`

Note: If you would like to select all (`*`) columns then simply do not use the select argument when building your query.

table(*table_name*)
The table that the query should be run on. This method must be run for any query to be executed.

update(*update_dict*)
Updates the given row/rows based on the dictionary.

For this method to run the *where* method must have been called before this one.

Parameters **update_dict** (*dict*) – The dictionary containing the column to update and the value to update it with.

where(*condition_1*, *comparison_operator*, *condition_2*)
Equivalent to the `WHERE` command in SQL.

Parameters

- **condition_1** – The value to the left of the operator.
- **comparison_operator** (*str*) – The comparison operator, e.g =
- **condition_2** – The value to the right of the operator.

3.1.2 Fields

Note:

- Please note that the majority of MySQL field types are available for usage. There name will just be the camel case of the MySQL type with `Field` appended.
- All fields accept the following parameters: `null`, `default`, `unique`, `primary_key`. These values are, by default, `False`. Some fields will have extra parameters which can be seen below.

class dragonfly.db.models.fields.**BigIntField**(***kwargs*)
Bases: *dragonfly.db.models.fields.IntField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**BinaryField**(***kwargs*)
Bases: *dragonfly.db.models.fields.StringField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**BitField**(*length=None*, ***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**BoolField**(***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**CharField**(***kwargs*)
Bases: *dragonfly.db.models.fields.StringField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

class dragonfly.db.models.fields.**DateField**(***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**DateTimeField**(*fsp=None*, ***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**DecimalField**(*digits=None*, *decimal_places=None*, *unsigned=False*, *zerofill=False*, ***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**DoubleField**(*digits=None*, *decimal_places=None*, *unsigned=False*, *zerofill=False*, ***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**Enum**(**args*, ***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**Field**(*name=None*, *null=False*, *blank=False*, *default=None*, *unique=False*, *primary_key=False*)
Bases: abc.ABC

An abstract class that defines the interface each Field class should have.

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**FloatField**(*digits=None*, *unsigned=False*, *zerofill=False*, ***kwargs*)

Bases: *dragonfly.db.models.fields.Field*

to_database_type()

This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)

This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**ForeignKey**(**args*)

Bases: object

Provides a way to define foreign key relationships in a model.

Should be called in the following order: ForeignKey('local_key').refrences('foreign_key').on('table')

on(*table*)

The table the foreign keys are on.

Parameters **table** – The table that the foreign keys are located on

Returns This ForeignKey object.

Return type *ForeignKey*

references(**args*)

Defines the foreign keys that the local keys reference.

Parameters **args** – A list of foreign keys that the defined local keys reference.

Returns This ForeignKey object.

Return type *ForeignKey*

class dragonfly.db.models.fields.**IntField**(*length=None*, *unsigned=False*, *auto_increment=False*, *zerofill=False*, ***kwargs*)

Bases: *dragonfly.db.models.fields.Field*

to_database_type()

This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)

This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**LongBlob**(***kwargs*)

Bases: *dragonfly.db.models.fields.Field*

to_database_type()

This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)

This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**MediumBlob**(***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**MediumIntField**(***kwargs*)
Bases: *dragonfly.db.models.fields.IntField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**MediumText**(***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**PrimaryKey**(**args*)
Bases: object

Provides a way to make the given field(s) a primary key

class dragonfly.db.models.fields.**Set**(**args*, ***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**SmallIntField**(***kwargs*)
Bases: *dragonfly.db.models.fields.IntField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

class dragonfly.db.models.fields.**StringField**(*length=None*, ***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**TextField**(**kwargs*)
Bases: *dragonfly.db.models.fields.StringField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

class dragonfly.db.models.fields.**TimeField**(*fsp=None*, **kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**TimestampField**(*fsp=None*, *on=None*, **kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**TinyBlobField**(**kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**TinyIntField**(**kwargs*)
Bases: *dragonfly.db.models.fields.IntField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

class dragonfly.db.models.fields.**TinyTextField**(**kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

class dragonfly.db.models.fields.**Unique**(**args*)
Bases: object

Provides a way to make a field a model unique

class dragonfly.db.models.fields.**VarCharField**(***kwargs*)
Bases: *dragonfly.db.models.fields.StringField*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

class dragonfly.db.models.fields.**YearField**(***kwargs*)
Bases: *dragonfly.db.models.fields.Field*

to_database_type()
This instructs the database migrator on how to generate the SQL for the field.

to_python_type(*value*)
This is how the value from the database should be converted to python. Note that at the moment this is not currently in use as the MySQLdb package does this automatically.

Parameters **value** – The value to convert

3.1.3 Model

class dragonfly.db.models.model.**Model**(*data=None*)
Bases: object

A way to easily interact with rows in a table.

add_relationship(*relationship_class*, *update=False*)
Adds a relationship to another model to this model instance using the relationship classes. Note that the method will cache the values of a relationship unless update is set to true.

Parameters

- **relationship_class** (*Relationship*) – An instantiated relationship class.
- **update** (*bool*) – If the method should retrieve fresh data from the database.

Returns The retrieved, related, model(s).

Return type *Relationship*

all()
Get all rows in the table .

Returns A list object models

Return type list

create(*create_dict*)
Creates a new row in the table and returns a model representation of this row.

Parameters **create_dict** (*dict*) – The values to create the new row with

Returns A list of object models

Return type list

delete()
Delete this row from the database .

find(*primary_key*)
Find a row by passing in the value of the row's primary key.

Note that if you would like to find a model with more than one key pass through a dictionary containing the column and value.

Example `Article().find({'id': 1, 'author': 1})`

Parameters **primary_key** (*int*) – The value of the primary key to find.

Returns An object model that has the given value as its primary key

Return type *Model*

first()
Get the first row in the table.

Returns An object model

Return type *Model*

get()
Get all rows in the table.

Returns An object model

Return type list

multiple_where(*where_dict*)
Select a row from the table using multiple values/columns

Parameters **where_dict** (*dict*) – A dictionary where the key is the column in the table and the value is the value in the table.

:return This model object :rtype: *Model*

paginate(*size*, *to_json=False*)
Paginates the data in the table by the given size.

Note that if `to_json` is `True` a *Response* will be returned containing the appropriate JSON. Otherwise a list of rows that correspond to the page requested will be returned (the page number is known from the request object).

Parameters

- **size** (*int*) – The number of rows on each page
- **to_json** (*bool*) – If the function should return a JSON response, default is False

Returns A tuple containing a dictionary if results and a dictionary contaning meta information

Return type tuple

save()
Permeate the changes made to the Python model to the database.

select(**args*)
Same as the *DB* class *select* method. Note that a dictionary will be returned as a model cannot be represented with incomplete data.

Parameters **args** – A list of columns to select

Returns This model object

Return type *Model*

to_dict()
Return a dictionary equivalent of this row.

Returns A dictionary equivalent of this row

Return type dict

update(*update_dict*)
Update this model with the given values.

Parameters **update_dict** (*dict*) – Update the given columns (key) with the given values

Returns This model object

Return type *Model*

where(*column*, *comparator*, *value*)
Same as the *DB* class *where* method.

Parameters

- **column** (*str*) – The column in the database to check the value for
- **comparator** (*str*) – The comparison operator e.g. =
- **value** – The value to test for

Returns This model object

Return type *Model*

dragonfly.db.models.model.**default**(*o*)
Function from StackOverflow - https://stackoverflow.com/questions/12122007/python-json-encoder-to-support-datetime

3.1.4 Relationships

class dragonfly.db.models.relationships.**BelongsTo**(***kwargs*)
Bases: *dragonfly.db.models.relationships.Relationship*

Retrieves the class that 'owns' the model this relationship is defined in.

delayed_init(*values*, *meta*)
This function is executed when data needs to be retrieved.

Parameters

- **values** (*dict*) – The database values of the given model
- **meta** (*dict*) – The meta values of the model

Returns The relationship class

Return type *Relationship*

class dragonfly.db.models.relationships.**HasMany**(***kwargs*)
Bases: *dragonfly.db.models.relationships.Relationship*

Retrieves all rows that have a have-many relationship with the model this class is initialised in.

delayed_init(*values*, *meta*)
This function is executed when data needs to be retrieved.

Parameters

- **values** (*dict*) – The database values of the given model

- **meta** (*dict*) – The meta values of the model

Returns The relationship class

Return type *Relationship*

class dragonfly.db.models.relationships.**ManyToMany**(*table=None*, ***kwargs*)
Bases: *dragonfly.db.models.relationships.Relationship*

delayed_init(*values*, *meta*)
This function is executed when data needs to be retrieved.

Parameters

- **values** (*dict*) – The database values of the given model
- **meta** (*dict*) – The meta values of the model

Returns The relationship class

Return type *Relationship*

class dragonfly.db.models.relationships.**Relationship**(*target*, *this_key=None*, *target_key=None*)
Bases: abc.ABC

An abstract class that defines the interface each *Relationship* class should have

delayed_init(*values*, *meta*)
This function is executed when data needs to be retrieved.

Parameters

- **values** (*dict*) – The database values of the given model
- **meta** (*dict*) – The meta values of the model

Returns The relationship class

Return type *Relationship*

Warning: The following classes should not be called directly.

3.1.5 DatabaseMigrator

class dragonfly.db.database_migrator.**DatabaseMigrator**(*path='models'*)
Bases: object

Generates the SQL to create a table that corresponds to the defined model(s)

3.1.6 Table

class dragonfly.db.table.**Table**
Bases: object

Returns the MySQL code to create a column in a table with the given type.

static bigint(**args*, ***kwargs*)

static binary(**args*, ***kwargs*)

static bit(**args*, ***kwargs*)

static blob(**args*, ***kwargs*)

static boolean(**args*, ***kwargs*)

static char(**args*, ***kwargs*)

static date(**args*, ***kwargs*)

static datetime(**args*, ***kwargs*)

static decimal(**args*, ***kwargs*)

static double(**args*, ***kwargs*)

static enum(**args*, ***kwargs*)

static float(**args*, ***kwargs*)

static foreign_key(*constraint_name*, *table*, *local_keys*, *foreign_keys*)

static integer(**args*, ***kwargs*)

static longblob(**args*, ***kwargs*)

static longtext(**args*, ***kwargs*)

static mediumblob(**args*, ***kwargs*)

static mediumint(**args*, ***kwargs*)

static mediumtext(**args*, ***kwargs*)

static primary_key(**args*)

static set(**args*, ***kwargs*)

static smallint(**args*, ***kwargs*)

static text(**args*, ***kwargs*)

static time(**args*, ***kwargs*)

static timestamp(**args*, ***kwargs*)

static tinyblob(**args*, ***kwargs*)

static tinyint(**args*, ***kwargs*)

static tinytext(**args*, ***kwargs*)

static unique(**args*, *constraint_name=None*)

static varbinary(**args*, ***kwargs*)

static varchar(**args*, ***kwargs*)

static year(**args*, ***kwargs*)

dragonfly.db.table.**handle_options**(*func*)

Handles any extra options for the methods on the *Table* class

3.2 Middleware

Warning: The following classes should not be called directly.

3.2.1 MiddlewareController

class dragonfly.middleware.middleware_controller.**MiddlewareController**
Bases: object

Loads all registered middleware and controls their execution.

run_after(*action*, *response*)
Run all the after methods on the middleware that are assigned to the given action.

Parameters

- **action** (*str*) – The action currently being executed.
- **response** (*Response*) – The Response that the router has generated. If the 'after' function accepts the Response class it is passed on (to allow for its modification).

Returns If a Response is generated return this

Return type *Response*

run_before(*action*)
Run all the before methods on middleware that are assigned to the given action.

Parameters **action** (*str*) – The action currently being executed.

Returns If a Response is generated return this

Return type *Response*

3.3 Routes

3.3.1 Router

class dragonfly.routes.router.**Router**
Bases: object

Routes the given route to the defined ''Controller''and returns its generated *Response*.

add_route(*uri*, *action*, *method*)
Adds a route to the *RouteCollection* object.

Parameters

- **uri** (*str*) – The URI of the route
- **action** (*str*) – The action of the route e.g 'HomeController@home'
- **method** (*str*) – The HTTP method verb e.g 'GET'

any(*uri*, *action*)

delete(*uri*, *action*)

dispatch_route()
Dispatches the appropriate route based on the request method and path.

get(*uri*, *action*)

options(*uri*, *action*)

patch(*uri*, *action*)

post(*uri*, *action*)

put (*uri*, *action*)

resource (*uri*, *controller*)

dragonfly.routes.router.**to_snake** (*name*)
From StackOverflow https://stackoverflow.com/questions/1175208/elegant-python-function-to-convert-camelcase-to-snake-case

Warning: The following classes should not be called directly.

3.3.2 RouteCollection

class dragonfly.routes.route_collection.**RouteCollection**
Bases: object

A way to store registered routes.

add (*uri*, *action*, *method*)
Add a new route to either the static or dynamic routes dictionary.

Parameters

- **uri** (*str*) – The route uri
- **action** (*str*) – The route action
- **method** (*str*) – The route HTTP method

match_route (*uri*, *method*)
Match the given route using its URI and method. First we check if it is a static route before checking all dynamic routes.

Parameters

- **uri** (*str*) – The URI to match
- **method** (*str*) – The HTTP method

Returns Any matching routes

Type dict

3.3.3 RouteRule

class dragonfly.routes.route_rule.**RouteRule** (*uri*)
Bases: object

Data structure to store dynamic routes. Allows for an easy check of whether a given route matches a dynamic route.

match (*uri*)
Matches the given route to an action and extracts any router parameters.

Parameters **uri** (*str*) – The URI to match.

Returns A dictionary containing the the action and any route parameters.

Return type dict

3.4 Templates

3.4.1 View

class dragonfly.template.template.**View**(*template*, ***kwargs*)
Bases: object

Returns a HTML version (view) of the requested template. The class first attempts to locate the deisred view. If a pre-compiled python version of the template does not exist or is out of date, the class will generate one. Otherwise it imports the compiled python file and runs the `get_html` method, passing in any variables that the user.py has given to the constructor (via `**kwargs`). It then returns a *Response* with this HTML. :param template: The view to return :type template: str

make()
Returns a response with the generated HTML.

Returns The `Response`

Return type *Response*

Warning: The following classes should not be called directly.

3.4.2 Line

class dragonfly.template.template.**Line**(*line*, *indent*)
Bases: object

Object to represent a single line in the template file

reduce_indent()
Reduce the indent of the `Line` by 1

to_python()
Converts the `Line` to its Python equivalent

3.4.3 Converter

class dragonfly.template.template.**Converter**(*file*)
Bases: object

convert()
Convert the given file to Python.

Returns The Python code.

Return type str

3.5 Web

3.5.1 Request

class dragonfly.request.**Request**(*environ*)
Bases: object

The request object is a class representation of the WSGI environ dictionary

get_data()
Gets any from data/query strings from the given request

Returns A dictionary containing the given data

Return type dict

update_environ(*new_environ*)
Updates the request object (singleton) with new data

Parameters **new_environ** (*dict*) – The new environ dictionary

3.5.2 Response

class dragonfly.response.**Response**(*content=''*, *content_type='text/html'*, *status_code=200*, *reason_phrase=None*)
Bases: object

The base *Response* class that is readable by the WSGI server.

Parameters

- **content** (*str*) – The content that will be delivered to the user. This defaults to empty.
- **content_type** (*str*) – The MIME type. This defaults to 'text/html'.
- **status_code** (*int*) – The HTTP status code. This defaults to success (200).
- **reason_phrase** (*int*) – A written meaning of the HTTP status code. If left as *None* the reason phrase will be chosen from a pre-determined list.

header(*field_name*, *field_value*)
Updates an existing header or creates a new one.

Parameters

- **field_name** (*str*) – The header field name.
- **field_value** (*str*) – The header field value.

set_content()
Converts the given content to bytes.

set_status(*status_code*, *reason_phrase*)
Sets the status of the *Response* object. If the reason_phrase is None then a reason phrase that corresponds to the status code will be retrieved from a constants file.

Parameters

- **status_code** (*int*) – The status code of the response.
- **reason_phrase** (*str*) – The reason phrase to accompany the status code. This can be *None*.

translate_deferred(*deferred*)
Merges the given *DeferredResponse* object to this *Response* instance.

Parameters **deferred** (*DeferredResponse*) – The *DeferredResponse* object.

3.5.3 ErrorResponse

class dragonfly.response.**ErrorResponse**(*error_message*, *status_code*)
Bases: *dragonfly.response.Response*

A *Response* object that returns an error page.

Parameters

- **error_message** (*str*) – The error message.
- **status_code** (*int*) – The status code.

3.5.4 DeferredResponse

class dragonfly.response.**DeferredResponse**
Bases: object

Allows headers for a future response to be set before it exists.

This singleton enables attributes of any *Response* object returned in the normal fashion, i.e through the *dispatch_route* method, to be set before it exists. The primary use of this class would be in the *before* method of a middleware.

header(*field_name*, *field_value*)
Define the headers to be set on the real *Response* object.

3.5.5 Auth

class dragonfly.auth.**Auth**
Bases: object

Provides a way to interact with the currently authenticated user.

static get(*key*, *model=False*)
Retrieve any keys stored in the Sessions table associated with the current session_id

Parameters

- **key** – The key of the value to retrieve
- **model** – If the entire model should be returned or the individual key value pair

Type str

Type bool

Returns The model or value

static set(*key*, *value*, *force=False*)
Set a key value pair in the Sessions table

Parameters

- **key** – The key
- **value** – The value
- **force** – If the sessions table should be forced to update the value

Type str

Type bool

static user()
Get the currently logged in user using the `session_id` stored in the cookies. It should be very unlikely that two sessions with the same ID exist

Returns The currently logged in user.

Type `User`

3.6 Other

3.6.1 Exceptions

exception dragonfly.exceptions.**ChunkOutOfRange**
Bases: `Exception`

exception dragonfly.exceptions.**InvalidControllerMethod**
Bases: `Exception`

exception dragonfly.exceptions.**InvalidOperator**
Bases: `Exception`

exception dragonfly.exceptions.**MethodDoesNotExist**
Bases: `Exception`

exception dragonfly.exceptions.**MissingClause**
Bases: `Exception`

exception dragonfly.exceptions.**MissingTable**
Bases: `Exception`

Python Module Index

d

dragonfly.db.database, 11
dragonfly.db.database_migrator, 21
dragonfly.db.models.fields, 12
dragonfly.db.models.model, 18
dragonfly.db.models.relationships, 20
dragonfly.db.table, 21
dragonfly.exceptions, 28
dragonfly.middleware.middleware_controller, 23
dragonfly.request, 25
dragonfly.routes.route_collection, 24
dragonfly.routes.route_rule, 24
dragonfly.routes.router, 23

Index

A

add() (*dragonfly.routes.route_collection.RouteCollection method*), 24
add_relationship() (*dragonfly.db.models.model.Model method*), 18
add_route() (*dragonfly.routes.router.Router method*), 23
all() (*dragonfly.db.models.model.Model method*), 18
any() (*dragonfly.routes.router.Router method*), 23
Auth (*class in dragonfly.auth*), 27

B

BelongsTo (*class in dragonfly.db.models.relationships*), 20
bigint() (*dragonfly.db.table.Table static method*), 21
BigIntField (*class in dragonfly.db.models.fields*), 12
binary() (*dragonfly.db.table.Table static method*), 21
BinaryField (*class in dragonfly.db.models.fields*), 13
bit() (*dragonfly.db.table.Table static method*), 21
BitField (*class in dragonfly.db.models.fields*), 13
blob() (*dragonfly.db.table.Table static method*), 21
boolean() (*dragonfly.db.table.Table static method*), 22
BoolField (*class in dragonfly.db.models.fields*), 13

C

char() (*dragonfly.db.table.Table static method*), 22
CharField (*class in dragonfly.db.models.fields*), 13
chunk() (*dragonfly.db.database.DB method*), 11
ChunkOutOfRange, 28
comparison_operators (*dragonfly.db.database.DB attribute*), 11
convert() (*dragonfly.template.template.Converter method*), 25
Converter (*class in dragonfly.template.template*), 25
create() (*dragonfly.db.models.model.Model method*), 18
custom_sql() (*dragonfly.db.database.DB method*), 11

D

DatabaseMigrator (*class in dragonfly.db.database_migrator*), 21
date() (*dragonfly.db.table.Table static method*), 22
DateField (*class in dragonfly.db.models.fields*), 13
datetime() (*dragonfly.db.table.Table static method*), 22
DateTimeField (*class in dragonfly.db.models.fields*), 14
DB (*class in dragonfly.db.database*), 11
decimal() (*dragonfly.db.table.Table static method*), 22
DecimalField (*class in dragonfly.db.models.fields*), 14
default() (*in module dragonfly.db.models.model*), 20
DeferredResponse (*class in dragonfly.response*), 27
delayed_init() (*dragonfly.db.models.relationships.BelongsTo method*), 20
delayed_init() (*dragonfly.db.models.relationships.HasMany method*), 20
delayed_init() (*dragonfly.db.models.relationships.ManyToMany method*), 21
delayed_init() (*dragonfly.db.models.relationships.Relationship method*), 21
delete() (*dragonfly.db.database.DB method*), 11
delete() (*dragonfly.db.models.model.Model method*), 18
delete() (*dragonfly.routes.router.Router method*), 23
dispatch_route() (*dragonfly.routes.router.Router method*), 23
double() (*dragonfly.db.table.Table static method*), 22
DoubleField (*class in dragonfly.db.models.fields*), 14
dragonfly.db.database (*module*), 11
dragonfly.db.database_migrator (*module*), 21
dragonfly.db.models.fields (*module*), 12

dragonfly.db.models.model (*module*), 18
dragonfly.db.models.relationships (*module*), 20
dragonfly.db.table (*module*), 21
dragonfly.exceptions (*module*), 28
dragonfly.middleware.middleware_controller (*module*), 23
dragonfly.request (*module*), 25
dragonfly.routes.route_collection (*module*), 24
dragonfly.routes.route_rule (*module*), 24
dragonfly.routes.router (*module*), 23

E

Enum (*class in dragonfly.db.models.fields*), 14
enum() (*dragonfly.db.table.Table static method*), 22
ErrorResponse (*class in dragonfly.response*), 27

F

Field (*class in dragonfly.db.models.fields*), 14
find() (*dragonfly.db.models.model.Model method*), 18
first() (*dragonfly.db.database.DB method*), 11
first() (*dragonfly.db.models.model.Model method*), 19
float() (*dragonfly.db.table.Table static method*), 22
FloatField (*class in dragonfly.db.models.fields*), 15
foreign_key() (*dragonfly.db.table.Table static method*), 22
ForeignKey (*class in dragonfly.db.models.fields*), 15

G

get() (*dragonfly.auth.Auth static method*), 27
get() (*dragonfly.db.database.DB method*), 12
get() (*dragonfly.db.models.model.Model method*), 19
get() (*dragonfly.routes.router.Router method*), 23
get_data() (*dragonfly.request.Request method*), 26

H

handle_options() (*in module dragonfly.db.table*), 22
HasMany (*class in dragonfly.db.models.relationships*), 20
header() (*dragonfly.response.DeferredResponse method*), 27
header() (*dragonfly.response.Response method*), 26

I

insert() (*dragonfly.db.database.DB method*), 12
integer() (*dragonfly.db.table.Table static method*), 22
IntField (*class in dragonfly.db.models.fields*), 15
InvalidControllerMethod, 28
InvalidOperator, 28

L

Line (*class in dragonfly.template.template*), 25
LongBlob (*class in dragonfly.db.models.fields*), 15
longblob() (*dragonfly.db.table.Table static method*), 22
longtext() (*dragonfly.db.table.Table static method*), 22

M

make() (*dragonfly.template.template.View method*), 25
ManyToMany (*class in dragonfly.db.models.relationships*), 21
match() (*dragonfly.routes.route_rule.RouteRule method*), 24
match_route() (*dragonfly.routes.route_collection.RouteCollection method*), 24
MediumBlob (*class in dragonfly.db.models.fields*), 15
mediumblob() (*dragonfly.db.table.Table static method*), 22
mediumint() (*dragonfly.db.table.Table static method*), 22
MediumIntField (*class in dragonfly.db.models.fields*), 16
MediumText (*class in dragonfly.db.models.fields*), 16
mediumtext() (*dragonfly.db.table.Table static method*), 22
MethodDoesNotExist, 28
MiddlewareController (*class in dragonfly.middleware.middleware_controller*), 23
MissingClause, 28
MissingTable, 28
Model (*class in dragonfly.db.models.model*), 18
multiple_where() (*dragonfly.db.database.DB method*), 12
multiple_where() (*dragonfly.db.models.model.Model method*), 19

O

on() (*dragonfly.db.models.fields.ForeignKey method*), 15
options() (*dragonfly.routes.router.Router method*), 23

P

paginate() (*dragonfly.db.models.model.Model method*), 19
patch() (*dragonfly.routes.router.Router method*), 23
post() (*dragonfly.routes.router.Router method*), 23
primary_key() (*dragonfly.db.table.Table static method*), 22
PrimaryKey (*class in dragonfly.db.models.fields*), 16
put() (*dragonfly.routes.router.Router method*), 23

R

reduce_indent() (*dragonfly.template.template.Line method*), 25
references() (*dragonfly.db.models.fields.ForeignKey method*), 15
Relationship (*class in dragonfly.db.models.relationships*), 21
Request (*class in dragonfly.request*), 25
resource() (*dragonfly.routes.router.Router method*), 24
Response (*class in dragonfly.response*), 26
RouteCollection (*class in dragonfly.routes.route_collection*), 24
Router (*class in dragonfly.routes.router*), 23
RouteRule (*class in dragonfly.routes.route_rule*), 24
run_after() (*dragonfly.middleware.middleware_controller.MiddlewareController method*), 23
run_before() (*dragonfly.middleware.middleware_controller.MiddlewareController method*), 23

S

save() (*dragonfly.db.models.model.Model method*), 19
select() (*dragonfly.db.database.DB method*), 12
select() (*dragonfly.db.models.model.Model method*), 19
Set (*class in dragonfly.db.models.fields*), 16
set() (*dragonfly.auth.Auth static method*), 27
set() (*dragonfly.db.table.Table static method*), 22
set_content() (*dragonfly.response.Response method*), 26
set_status() (*dragonfly.response.Response method*), 26
smallint() (*dragonfly.db.table.Table static method*), 22
SmallIntField (*class in dragonfly.db.models.fields*), 16
StringField (*class in dragonfly.db.models.fields*), 16

T

Table (*class in dragonfly.db.table*), 21
table() (*dragonfly.db.database.DB method*), 12
text() (*dragonfly.db.table.Table static method*), 22
TextField (*class in dragonfly.db.models.fields*), 17
time() (*dragonfly.db.table.Table static method*), 22
TimeField (*class in dragonfly.db.models.fields*), 17
timestamp() (*dragonfly.db.table.Table static method*), 22
TimestampField (*class in dragonfly.db.models.fields*), 17
tinyblob() (*dragonfly.db.table.Table static method*), 22
TinyBlobField (*class in dragonfly.db.models.fields*), 17
tinyint() (*dragonfly.db.table.Table static method*), 22
TinyIntField (*class in dragonfly.db.models.fields*), 17
tinytext() (*dragonfly.db.table.Table static method*), 22
TinyTextField (*class in dragonfly.db.models.fields*), 17
to_database_type() (*dragonfly.db.models.fields.BigIntField method*), 13
to_database_type() (*dragonfly.db.models.fields.BinaryField method*), 13
to_database_type() (*dragonfly.db.models.fields.BitField method*), 13
to_database_type() (*dragonfly.db.models.fields.BoolField method*), 13
to_database_type() (*dragonfly.db.models.fields.CharField method*), 13
to_database_type() (*dragonfly.db.models.fields.DateField method*), 13
to_database_type() (*dragonfly.db.models.fields.DateTimeField method*), 14
to_database_type() (*dragonfly.db.models.fields.DecimalField method*), 14
to_database_type() (*dragonfly.db.models.fields.DoubleField method*), 14
to_database_type() (*dragonfly.db.models.fields.Enum method*), 14
to_database_type() (*dragonfly.db.models.fields.Field method*), 14
to_database_type() (*dragonfly.db.models.fields.FloatField method*), 15
to_database_type() (*dragonfly.db.models.fields.IntField method*), 15
to_database_type() (*dragonfly.db.models.fields.LongBlob method*), 15
to_database_type() (*dragonfly.db.models.fields.MediumBlob method*), 16
to_database_type() (*dragonfly.db.models.fields.MediumIntField method*), 16
to_database_type() (*dragonfly.db.models.fields.MediumText method*), 16
to_database_type() (*dragonfly.db.models.fields.Set method*), 16
to_database_type() (*dragon-*

fly.db.models.fields.SmallIntField method), 16
to_database_type() (*dragonfly.db.models.fields.StringField method*), 16
to_database_type() (*dragonfly.db.models.fields.TextField method*), 17
to_database_type() (*dragonfly.db.models.fields.TimeField method*), 17
to_database_type() (*dragonfly.db.models.fields.TimestampField method*), 17
to_database_type() (*dragonfly.db.models.fields.TinyBlobField method*), 17
to_database_type() (*dragonfly.db.models.fields.TinyIntField method*), 17
to_database_type() (*dragonfly.db.models.fields.TinyTextField method*), 17
to_database_type() (*dragonfly.db.models.fields.VarCharField method*), 18
to_database_type() (*dragonfly.db.models.fields.YearField method*), 18
to_dict() (*dragonfly.db.models.model.Model method*), 19
to_python() (*dragonfly.template.template.Line method*), 25
to_python_type() (*dragonfly.db.models.fields.BigIntField method*), 13
to_python_type() (*dragonfly.db.models.fields.BinaryField method*), 13
to_python_type() (*dragonfly.db.models.fields.BitField method*), 13
to_python_type() (*dragonfly.db.models.fields.BoolField method*), 13
to_python_type() (*dragonfly.db.models.fields.DateField method*), 13
to_python_type() (*dragonfly.db.models.fields.DateTimeField method*), 14
to_python_type() (*dragonfly.db.models.fields.DecimalField method*), 14
to_python_type() (*dragonfly.db.models.fields.DoubleField method*), 14
to_python_type() (*dragonfly.db.models.fields.Enum method*), 14
to_python_type() (*dragonfly.db.models.fields.Field method*), 14
to_python_type() (*dragonfly.db.models.fields.FloatField method*), 15
to_python_type() (*dragonfly.db.models.fields.IntField method*), 15
to_python_type() (*dragonfly.db.models.fields.LongBlob method*), 15
to_python_type() (*dragonfly.db.models.fields.MediumBlob method*), 16
to_python_type() (*dragonfly.db.models.fields.MediumIntField method*), 16
to_python_type() (*dragonfly.db.models.fields.MediumText method*), 16
to_python_type() (*dragonfly.db.models.fields.Set method*), 16
to_python_type() (*dragonfly.db.models.fields.StringField method*), 16
to_python_type() (*dragonfly.db.models.fields.TimeField method*), 17
to_python_type() (*dragonfly.db.models.fields.TimestampField method*), 17
to_python_type() (*dragonfly.db.models.fields.TinyBlobField method*), 17
to_python_type() (*dragonfly.db.models.fields.TinyTextField method*), 17
to_python_type() (*dragonfly.db.models.fields.YearField method*), 18
to_snake() (*in module dragonfly.routes.router*), 24
translate_deferred() (*dragonfly.response.Response method*), 26

U

Unique (*class in dragonfly.db.models.fields*), 18
unique() (*dragonfly.db.table.Table static method*), 22
update() (*dragonfly.db.database.DB method*), 12
update() (*dragonfly.db.models.model.Model method*), 20
update_environ() (*dragonfly.request.Request method*), 26
user() (*dragonfly.auth.Auth static method*), 27

V

varbinary() (*dragonfly.db.table.Table static method*), 22
varchar() (*dragonfly.db.table.Table static method*), 22
VarCharField (*class in dragonfly.db.models.fields*), 18

View (*class in dragonfly.template.template*), 25

W

where() (*dragonfly.db.database.DB method*), 12
where() (*dragonfly.db.models.model.Model method*), 20

Y

year() (*dragonfly.db.table.Table static method*), 22
YearField (*class in dragonfly.db.models.fields*), 18