
Toolkit Documentation

Release 0.0.0

Yufei Li

October 12, 2015

1	Feature	3
2	Install	5
3	Tools Document	7
3.1	Unicode Tools	7
3.2	Jinja2 Tools	8
4	API Reference	11
4.1	API	11
5	Indices and tables	15
	Python Module Index	17

Toolkit is a library has some utils and helper function. It's also have a command line tool for managing scripts writed with python & bash.

- *Feature*
- *Install*
- *Tools Document*
- *API Reference*

Feature

- Easy to refactor, every tools as a module.
- Pure python implemented.
- Continual developing && updating.

Install

pip

```
pip install toolkit
```

source

```
git clone https://github.com/yufeiminds/toolkit.git  
cd toolkit  
python setup.py install
```

Tools Document

3.1 Unicode Tools

ulib is a library for processing the unicode character or string more pythonic.

Warning: This module not finished, don't use it on product environment.

3.1.1 Common

```
from toolkit import ulib

ulib.is_digital(u'9')
>>> True

ulib.is_digital(u'q')
>>> False
```

3.1.2 Chinese

if the param is not unicode, is_cn() will raise a NotUnicodeException.

```
from toolkit import ulib

ulib.is_cn(u'äÿ')
>>> True

ulib.is_cn(u'c')
>>> False

try:
    ulib.is_cn('äÿ')
except NotUnicodeException as e:
    pass
```

the has_cn() test if there is chinese unicode character in the unicode string.

```
ulib.has_cn(u'Here is a äÿ character.')
>>> True

ulib.has_cn(u'Here is not any chinese character.')
>>> False
```

cnlen() also can caculate unicode string that contained chinese character length simply,

```
ulib.cnlen('Here is äÿæŮĜ')
>>> 12
```

Note: `cnlen()` caculate one full-width character as two halt-width characters.

3.2 Jinja2 Tools

The `tpl` module has some helper function implements base on `jinja2` for human.

3.2.1 Environment

Register is a fast function for injecting function to `jinja2` environment that create by `tpl`,

Note: `register` must be call before the `jinja2` environment has been created.

Environment can be created by:

- `create_env_by_folder`

Tip: the `create_env_by_folder()` based on `jinja2.environment` with `jinja2.FileSystemLoader()`

Generally speaking, there are three ways to call a register

- immutable function
- decorator
- decorator with name

Toolkit implemented 2 register for `jinja2` environment:

register_filter

```
register_filter('lower', str.lower)

@register_filter
def lower(s):
    return s.lower()

@register_filter('lower')
def xxx_lower():
    return s.lower()
```

and you can use it on `jinja2` template:

```
{{HERE is a Demo | lower }}

>>> here is a demo
```

register_test

the `register_test()` can register the test function, use it on template:

```
register_test('digital', lambda v: type(v) in (int, float))

@register_test
def digital(v):
    return type(v) in (int, float)
```

```
@register_test('digital')
def test_if_it_is_digital():
    return type(v) in (int, float)
```

```
{{ 9 is digital }}
```

3.2.2 Template

See the template as an object, `get_template()` get template object from absolute path.

For example, the template has:

```
{% macro add(a, b) -%}
    {{a + b}}
{%- endmacro %}
```

And call the macro as template object method.

```
tpl = get_template('template.tpl')
print(tpl.add(1, 2))

>>> 3
```

We also can render jinja2 template file with absolute path.

A simple template as follow:

```
This is a test template

{{ title }}
```

Then write render code by single line.

```
render('template.tpl', name='toolkit')
```

API Reference

4.1 API

4.1.1 Ulib

ulib is a library for processing the unicode character or string more pythonic.

Warning: This module not finished, don't use it on product environment.

Support:

- Chinese

`toolkit.ulib.cnlen(us)`

Calculate the length of unicode string. length of chinese character is 2.

Parameters **us** – Unicode string.

Returns Bool Value

`toolkit.ulib.f2h(u)`

Parameters **us** – Unicode character or string.

Returns

`toolkit.ulib.has_cn(us)`

Test if the unicode string contain an unicode chinese character.

Parameters **us** – Unicode string.

Returns Bool Value

`toolkit.ulib.is_cn(u)`

Test if the unicode character is a chinese character.

Parameters **u** – Unicode character.

Returns Bool Value

`toolkit.ulib.is_digital(u)`

Test if the unicode character is a digital.

Parameters **u** – Unicode character.

Returns Bool Value

4.1.2 Tpl

The `tpl` module has some helper function implements base on `jinja2` for human.

`toolkit.tpl.create_env_by_folder(folder)`
Create `jinja2` environment with `jinja2.FileSystemLoader()`

Parameters `folder` – folder path.

Returns `jinja2` environment object.

`toolkit.tpl.get_template(template_path)`
Get template object from absolute path.

For example, the template has:

```
{% macro add(a, b) -%}
    {{a + b}}
{%- endmacro %}
```

And call the macro as template object method.

```
tpl = get_template('template.tpl')
print(tpl.add(1, 2))

>>> 3
```

Parameters `template_path` – template absolute path.

Returns template object

`toolkit.tpl.proxy_register(register_funtion)`
Proxy a function to a register function.

Parameters `register_funtion` – a function need to proxy.

Returns a proxy wrapper

`toolkit.tpl.register_filter(excepted, filter_function=None)`
Add default filter function to template rendered environment. Register provide 3-way to add a function to environment and use on template.

```
register_filter('lower', str.lower)

@register_filter
def lower(s):
    return s.lower()

@register_filter('lower')
def xxx_lower():
    return s.lower()
```

`toolkit.tpl.register_test(excepted, filter_function=None)`
Add default test function to template rendered environment. Register provide 3-way to add a function to environment and use on template.

```
register_test('digital', lambda v: type(v) in (int, float))

@register_test
def digital(v):
    return type(v) in (int, float)

@register_test('digital')
def test_if_it_is_digital():
    return type(v) in (int, float)
```

```
toolkit.tpl.render(template_path, output_file=None, **kwargs)
```

Render jinja2 template file use absolute path.

Usage

A simple template as follow:

```
This is a test template
```

```
{{ title }}
```

Then write render code by single line.

```
render('template.tpl', name='toolkit')
```

Parameters

- **template_path** – absolute file path of template file.
- **output_file** – the path of output file.
- **kwargs** – keyword arguments for template.

Indices and tables

- `genindex`
- `modindex`
- `search`

t

`toolkit.tpl`, [12](#)

`toolkit.ulib`, [11](#)

C

`cnlen()` (in module `toolkit.ulib`), [11](#)
`create_env_by_folder()` (in module `toolkit.tpl`), [12](#)

F

`f2h()` (in module `toolkit.ulib`), [11](#)

G

`get_template()` (in module `toolkit.tpl`), [12](#)

H

`has_cn()` (in module `toolkit.ulib`), [11](#)

I

`is_cn()` (in module `toolkit.ulib`), [11](#)
`is_digital()` (in module `toolkit.ulib`), [11](#)

P

`proxy_register()` (in module `toolkit.tpl`), [12](#)

R

`register_filter()` (in module `toolkit.tpl`), [12](#)
`register_test()` (in module `toolkit.tpl`), [12](#)
`render()` (in module `toolkit.tpl`), [12](#)

T

`toolkit.tpl` (module), [12](#)
`toolkit.ulib` (module), [11](#)