

Uliweb Cheat Sheet 0.2

作者: limodou@gmail.com

(发布日期 2011/4/21)

整体介绍

说明:Uliweb 是一个 Python Web 开发框架

开发模式:MVT(Model, View, Template)

项目结构:

```
project/
  apps/
    settings.ini
    app1/
      templates/
        xxx.html
      settings.ini
      views.py
      models.py
      ...
  app.yaml
  gae_handler.py
  fcgi_handler.py
  wsgi_handler.py
```

项目主页:

<http://code.google.com/p/uliweb>

安装

svn co <http://uliweb.googlecode.com/svn/trunk/> uliweb

python setup.py develop

Hello,Uliweb - 第一个 Uliweb 程序

uliweb makeproject Project1

cd Project1

uliweb makeapp Hello

uliweb runserver

在浏览器访问: <http://localhost:8000>

开发流程

功能开发: 创建 App->编辑 views->设置 url|处理 Model|处理 Form

配置开发: 修改 app 的 settings|使用 dispatch

View 基本内容

文件名:views.py, views_xx.py, views_yy.py

```
def __begin__():
    #进入 view 函数之前的处理

def __end__():
    #执行 view 函数之后的处理
```

```
@expose('/')
def index():
    return 'Hello,world'
```

```
@expose('/user/<name>') #expose 中可以定义参数
def user_view(name): #在 view 函数中要定义同名参数
    from werkzeug import Response
    return Response("Hello, world")#返回一个 Response
```

Class View

在 views.py 中也可以使用类来定义 view 函数。

```
@expose('/')
class TestView(object):
    def __begin__(self):
        print 'test begin'

    def hello(self):
        return 'hello %s %s' % (url_for('print'),
url_for(TestView.printf))

    @expose('/tprint', name='print')
    def printf(self):
        return 'print'
```

说明:

1. 类不需要从特殊父类继承
2. 使用实例方法(带 self)或类方法(使用 classmethod 修

饰)来定义。不要使用静态方法。

3. 如果方法名开头为 '_' 则不会自动输出对应的 URL。
4. 如果方法如示例中的 hello 函数, 未带 @expose, 则自动输出对应的 URL。即类的 URL+方法名, 如: /hello。如果还有参数, 则自动将参数添加到后面。
5. 如果方法也使用了 @expose, 如 printf 函数, 则方法的 URL 会与类的 URL 进行合并。**注意: 支持相对路径的定义方式。**
6. 在类方法中, 也可以定义类似 views.py 中的 __begin__ 和 __end__ 方法。
7. 模板自动套用名, 缺省为: 类名/方法名。在 settings.ini 的缺省定义为:
[GLOBAL]
TEMPLATE_TEMPLATE = ('%(view_class)s/%(function)s', '%(function)s')
其中 view_class 为类名, function 为函数名。

Expose 格式

```
@expose('/user/<name>')
@expose('/users', default={'pageno':0})#可以传入缺省值
@expose('/users/<int:pageno>')#匹配整数
@expose('/files/<path:file>') #匹配/files/之后的内容(含/)
@GET(URL) #只允许 GET 方式
@POST(URL) #只允许 POST 方式
命名 url
@expose('/user/<name>', name='user_view')
#反向获取 url_for('user_view', name='test')
```

View 的返回值

```
Response() #from werkzeug import Response
'Hello'
{} #自动套用同名的模板
json(data)
redirect(new_url)
error(error_message)
others #自动使用 str(others)
```

可以在 view 中和 template 中直接使用的变量

```
request      #线程安全
response     #线程安全
application  #当前应用对象
settings     #配置对象
json        #json 数据
url_for     #反向 url 获取
error       #出错
redirect    #重定向
```

☉也可以通过 `from uliweb import xx` 来导入上面内容

反向 url 获取(url_for)

```
url_for('appname.views.view_func', **kwargs)
url_for('named_url', **kwargs)
url_for(view_func, **kwargs)
安装 uliweb.contrib.staticfiles 之后, 可以使用:
url_for_static(filename) #获得静态文件 url
```

Template 语法

```
{{=var}}          #escape html chars
{{=func()}}       #escape html chars
{{python code}}
{{<< htmlcode}}  #unescape html chars
{{if user=='admin':}}
    <p>Welcome</p>
{{else:}}
    <a href="/login">Login</a>
{{pass}}
{{block content}}<div>default content</div>{{end}}
{{embed var}}
{{include "template.html"}} or {{include var}}
{{extend "template.html"}} or {{extend var}}
安装 uliweb.contrib.template 之后, 可以使用:
{{use "template_plugin_name"}}
{{link "js|css"}}
```

settings.ini 语法

```
#coding=UTF-8
```

```
[GLOBAL]
DEBUG = True
INSTALLED_APPS = [
    'uliweb.contrib.staticfiles',
]
MIDDLEWARE_CLASSES = [
    'uliweb.orm.middle_transaction.TransactionMiddle'
]
TIME_ZONE = 'UTC'
```

uliweb 命令行

uliweb 显示全部可用的命令列表(根据有效的 app 生成)
使用 `uliweb help <command>` 可以查看某个命令的详细信息
命令格式:

```
Usage: uliweb [global_options] [subcommand [options]
[args]]
```

Global Options:

```
--help          show this help message and exit.
-v, --verbose   Output the result in verbose mode.
-s SETTINGS, --settings=SETTINGS
                Settings file name. Default is "settings.ini".
--project=PROJECT Project "apps" directory.
--pythonpath=PYTHONPATH
                A directory to add to the Python path, e.g.
                "/home/myproject".
--version       show program's version number and exit
```

Type 'uliweb help <subcommand>' for help on a specific subcommand.

常用命令:

```
uliweb runserver 启动开发服务器
uliweb develop 启动开发服务器, 同时装入 develop app
uliweb makeproject 创建一个项目
```

```
uliweb makeapp 创建一个 app
uliweb makepkg 创建一个 python 的包目录
uliweb shell 进入 shell 环境
uliweb i18n 执行 i18n 的处理
uliweb extracturls 将所有 url 取出放入 urls.py 取消
uliweb exportstatic 将所有静态文件导出到指定目录
uliweb call 调用 app 下的命令
```

安装 uliweb.contrib.orm 后, 可以使用:

```
uliweb dbinit 执行 app 下的 dbinit.py 程序, 进行初始化
uliweb dump dump 出数据库中的数
uliweb load 将 dump 的数据重新装回数据库
uliweb syncdb 自动创建表
uliweb sql 查看 create table 语句
uliweb reset 重新建表
uliweb sqldot 生成 Model 关系的 dot 文件, 可以使用
graphviz 进行图形化处理
同时还支持表级的处理命令:
uliweb loadtable
uliweb dumptable
uliweb resettable
安装 uliweb.contrib.auth 后可以执行:
uliweb createsuperuser 创建超级用户
```

ORM—settings.ini 设置

```
[ORM]
DEBUG_LOG = False
AUTO_CREATE = True
CONNECTION = 'sqlite:///'
CONNECTION 格式:
#sqlite
'sqlite:///absolute/path/to/database.txt'
'sqlite:///d:/absolute/path/to/database.txt'
'sqlite:///relative/path/to/database.txt'
'sqlite://' # in-memory database
'sqlite://:memory:' # the same
```

```
# postgresql
'postgres://scott:tiger@localhost/mydatabase'
```

```
# mysql
'mysql://scott:tiger@localhost/mydatabase'
```

```
# oracle
'oracle://scott:tiger@127.0.0.1:1521/sidname'
```

```
# oracle via TNS name
'oracle://scott:tiger@tnsname'
```

```
# mssql using ODBC datasource names. PyODBC is the
default driver.
'mssql://mydsn'
'mssql://scott:tiger@mydsn'
```

```
# firebird
'firebird://scott:tiger@localhost/sometest.gdm'
```

ORM—Model 定义

```
from uliweb.orm import *
class Requirement(Model):
    req_id = Field(CHAR, max_length=12,)
    year = Field(int, required=True)
```

可用字段 (简便方式):

```
StringProperty Field(str) #vchar
CharProperty Field(CHAR) #char
UnicodeProperty Field(Unicode)#vchar
TextProperty Field(TEXT) #Text
BlobProperty Field(BLOB) #Blob
FileProperty Field(FILE) #vchar,保存文件名
IntegerProperty Field(int) #int
FloatProperty Field(float) #float
BooleanProperty Field(bool) #boolean
```

```
DateTimeProperty Field(datetime)#datetime
DateProperty Field(date) #date
TimeProperty Field(time) #time
DecimalProperty Field(DECIMAL)#numeric
```

关系定义:

```
Reference
SelfReference
OneToOne
ManyToMany
```

自定义表属性:

```
class Note(Model):
    __table__ = 't_note'#表名
    __table_args__ = dict(mysql_charset='utf8')
```

```
@classmethod
def OnInit(cls):
    Index('my_idx', cls.c.title, cls.c.owner,
unique=True)
```

Model 属性:

```
.table #sqlalchemy 中的 Table 对象
.metadata #sqlalchemy 中的 metadata 对象
.c #等同于 model.table.c
.properties #属性列表
._manytomany #manytomany 属性
```

ORM—实例级操作

```
class User(Model):
    username = Field(CHAR, max_length=20)
    year = Field(int)
```

创建:

```
user = User(username='user', year=20)
user.save()
```

获取:

```
user = User.get(1)
user = User.get(User.c.id == 1)
```

删除:

```
user.delete()
修改:
user.username = 'user1'
user.save()
or
user.update(**data)
其它方法:
user.to_dict(*fields)
```

ORM—表级操作

```
User.all()
User.filter(User.c.year > 18)
User.remove(condition)
User.count(condition)
```

ORM—结果集操作

直接返回单表结果集:

```
User.all()
User.filter()
```

通过关系返回结果集:

```
class User(Model):
    username = Field(CHAR, max_length=20)
    year = Field(int)
class Group(Model):
    name = Field(str, max_length=20)
    manager = Reference(User,
collection_name='m_groups')
    users = ManyToMany(User, collection_name='groups')
```

以下方法返回结果集:

```
user.m_groups #多表结果集
group.users #多表结果集
user.groups #单表结果集
```

单表结果集方法:

```
all() #全部, 返回结果集
filter() #过滤, 返回结果集
order_by() #排序, 返回结果集
```

limit() #限制, 返回结果集
offset() #偏移, 返回结果集
distinct() #不重复, 返回结果集
返回结果:
clear() #清除
count() #计数
one() #单条记录
values() #返回字段列表的[]结果
values_one() #返回字段列表的单条结果

多表结果集方法:

all() #全部, 返回结果集
filter() #过滤, 返回结果集
order_by() #排序, 返回结果集
limit() #限制, 返回结果集
offset() #偏移, 返回结果集
distinct() #不重复, 返回结果集
返回结果:
clear() #清除
count() #计数
one() #单条记录
values() #返回字段列表的[]结果
values_one() #返回字段列表的单条结果
has() #存在

Model—Settings 配置

在 settings.ini 中, 如下:

```
[MODELS]
```

```
assignment = 'assignments.models.Assignment'
```

以后可以通过以下方式获得 Model:

```
from uliweb.orm import get_model
```

```
Assignment = get_model('assignment')
```

同时可以在定义关系时, 使用字符串来表示一个表。

Dispatch

```
dispatch.call(application, 'startup_installed')
```

```
dispatch.call(application, 'startup')
```

```
dispatch.call(application, 'prepare_view_env', env)
```

安装完 uliweb.contrib.orm 后增加:

```
dispatch.call(model, 'pre_save', instance, created, data,  
old_data)
```

```
dispatch.call(model, 'post_save', instance, created, data,  
old_data)
```

```
dispatch.call(model, 'pre_delete', instance)
```

```
dispatch.call(model, 'post_delete', instance)
```