

Scripting & Computer Environments

Web2py: The Views

IIT-H

Nov 13, 2013

...Previously & Today...

Previously: The Models

- The Database Abstraction Layer (DAL) → The `db.py` file
- Menus → The `menu.py` file

Today: The Views

❶ Layouts

❷ HELPERS

❸ Form Processing

- View files (a.k.a. templates) control *data presentation*.
- Presentation \equiv layout, style, look and feel, aesthetics
- Render the output of a controller to the browser.
- Embed Python code into HTML inside $\{\{...\}\}$.
- Are multilingual:
HTML, XML, RSS, ATOM, AJAX, JSON, RTF, CSV, WIKI, XML-RPC,
REST, Flash, etc.

- Code indentation is according to HTML, not Python.
- Code blocks start with ‘:’ and end with `pass` (unless obvious).
- The `response.write()` method
- Error in a view shows the generated view code, not the developer's.

Examples

- `{{=object}}`

Examples

- `{{=object}}`
- Looping constructs: `for...in` / `while`

Examples

- `{{=object}}`
- Looping constructs: `for...in / while`
- Conditionals: `if...elif...else`

Examples

- `{{=object}}`
- Looping constructs: `for...in / while`
- Conditionals: `if...elif...else`
- Functions: `def...return`

- Views can extend another view / be embeded in another view.
- Via the **extend** and **include** directives

```
{{extend 'thisfile.html'}}           # 'layout.html'
```

```
{{include 'thatfile.html'}}
```

The Layout File

- The default layout file is `views/layout.html` (in HTML5)

- Includes CSS, JS and other view files.

e.g. `web2py.css`, `web2py_ajax.html`

- Contains blocks:

`status bar`, `left_sidebar`, `center`, `right_sidebar`, `header` and `footer`.

- Is highly customizable and replaceable ¹.

¹Check this out: layout styles available [here](#)

- Are used to build HTML programmatically.
- Can be used in Models, Views and Controllers.
- The corresponding helpers exist for most HTML tags.
- Arguments that start with an underscore are interpreted as HTML tag attributes (few exceptions).
- Written in upper-case

e.g. A, B, BEAUTIFY, DIV, FORM, HTML, INPUT, MENU, P ...

- Positional arguments are interpreted as objects contained between `<` and `>`.
- A helper can take a list/tuple and dictionary arguments as its set of components using the `*` and `**` notations.
- Strings in the components of a helper are escaped.
- The `XML()` helper turns off the escaping.

```
>>>print H1('<hi>', XML('<there>'))
```

Some HELPERS

1 The **B** and **I** Helpers (bold/italic)

2 The **BODY** Helper

```
>>>print BODY('hello world', _someattribute = 'somevalue')
```

N.B. Web2py does not check for valid attributes.

3. The **H1-H6** Helpers

4. The **A** Helper: (Creates links)

```
{{=A('Search', _href='http://www.google.com')}}}
```

```
>>>print A('Search', _href='http://www.google.com')
```

```
{{=A('click me', callback = URL('myfunction'))}}}
```

Some HELPERS

❶ The **B** and **I** Helpers (bold/italic)

❷ The **BODY** Helper

```
>>>print BODY('hello world', _someattribute = 'somevalue')
```

N.B. Web2py does not check for valid attributes.

3. The **H1-H6** Helpers

4. The **A** Helper: (Creates links)

```
{%=A('Search', _href='http://www.google.com')%}
```

```
>>>print A('Search', _href='http://www.google.com')
```

```
{%=A('click me', callback = URL('myfunction'))%}
```

Some HELPERS

1 The **B** and **I** Helpers (bold/italic)

2 The **BODY** Helper

```
>>>print BODY('hello world', _someattribute = 'somevalue')
```

N.B. Web2py does not check for valid attributes.

3. The **H1-H6** Helpers

4. The **A** Helper: (Creates links)

```
{%=A('Search', _href='http://www.google.com')%}
```

```
>>>print A('Search', _href='http://www.google.com')
```

```
{%=A('click me', callback = URL('myfunction'))%}
```

Some HELPERS

❶ The **B** and **I** Helpers (bold/italic)

❷ The **BODY** Helper

```
>>>print BODY('hello world', _someattribute = 'somevalue')
```

N.B. Web2py does not check for valid attributes.

❸ The **H1-H6** Helpers

❹ The **A** Helper: (Creates links)

```
{{=A('Search', _href='http://www.google.com')}}}
```

```
>>>print A('Search', _href='http://www.google.com')
```

```
{{=A('click me', callback = URL('myfunction'))}}}
```


5. The `UL/OL/LI` Helpers

6. The `BEAUTIFY` Helper

- Builds HTML representations of compound objects: lists, tuples, dictionaries.

```
{{=BEAUTIFY({1: ['hello'], 2: ('hi', 'bye')}) }}
```

7. The `IMG` Helper

```
IMG(_src='http://example.com/image.png',_alt='test')
```

8. The `TABLE/TR/TD` Helpers

```
TABLE(TR(TD('cell1'), TD('cell2')), TR(TD('cell3'), TD('cell4')))
```

```
TABLE(TR('cell1', 'cell2'), TR('cell3', 'cell4'))
```

9. The `MENU` Helper

5. The `UL/OL/LI` Helpers

6. The `BEAUTIFY` Helper

- Builds HTML representations of compound objects: lists, tuples, dictionaries.

```
{{=BEAUTIFY({1: ['hello'], 2: ('hi', 'bye')}) }}
```

7. The `IMG` Helper

```
IMG(_src='http://example.com/image.png',_alt='test')
```

8. The `TABLE/TR/TD` Helpers

```
TABLE(TR(TD('cell1'), TD('cell2')), TR(TD('cell3'), TD('cell4')))
```

```
TABLE(TR('cell1', 'cell2'), TR('cell3', 'cell4'))
```

9. The `MENU` Helper

5. The `UL/OL/LI` Helpers

6. The `BEAUTIFY` Helper

- Builds HTML representations of compound objects: lists, tuples, dictionaries.

```
{{=BEAUTIFY({1: ['hello'], 2: ('hi', 'bye')}) }}
```

7. The `IMG` Helper

```
IMG(_src='http://example.com/image.png',_alt='test')
```

8. The `TABLE/TR/TD` Helpers

```
TABLE(TR(TD('cell1'), TD('cell2')), TR(TD('cell3'), TD('cell4')))
```

```
TABLE(TR('cell1', 'cell2'), TR('cell3', 'cell4'))
```

9. The `MENU` Helper

5. The `UL/OL/LI` Helpers

6. The `BEAUTIFY` Helper

- Builds HTML representations of compound objects: lists, tuples, dictionaries.

```
{{=BEAUTIFY({1: ['hello'], 2: ('hi', 'bye')}) }}
```

7. The `IMG` Helper

```
IMG(_src='http://example.com/image.png',_alt='test')
```

8. The `TABLE/TR/TD` Helpers

```
TABLE(TR(TD('cell1'), TD('cell2')), TR(TD('cell3'), TD('cell4')))
```

```
TABLE(TR('cell1', 'cell2'), TR('cell3', 'cell4'))
```

9. The `MENU` Helper

5. The `UL/OL/LI` Helpers

6. The `BEAUTIFY` Helper

- Builds HTML representations of compound objects: lists, tuples, dictionaries.

```
{{=BEAUTIFY({1: ['hello'], 2: ('hi', 'bye')}) }}
```

7. The `IMG` Helper

```
IMG(_src='http://example.com/image.png',_alt='test')
```

8. The `TABLE/TR/TD` Helpers

```
TABLE(TR(TD('cell1'), TD('cell2')), TR(TD('cell3'), TD('cell4')))
```

```
TABLE(TR('cell1', 'cell2'), TR('cell3', 'cell4'))
```

9. The `MENU` Helper

Can be generated in various ways:

- 1 The FORM Helper
- 2 SQLFORM form from DB table
- 3 SQLFORM.factory form from description of a table
- 4 The CRUD API

- Can be used with other helpers: INPUT, SELECT, TEXTAREA ...

```
def index():  
    form = FORM('Your name:', INPUT(_name='name'), INPUT(_type='submit'))  
    # do some form processing  
    # on success, do this thing  
    # on failure, do that thing  
  
    return dict(form=form)
```

- The form fields can be validated using *validators*.
- Useful variables:

- `request.vars` submitted values
- `form.vars` accepted values
- `form.errors` erroneous values

- The `form.accepts()` method
 - Filters `request.vars` based on the specified validators.
 - Returns `True` if the form is accepted and `False` otherwise.
 - Stores the values into `form.vars` and `form.errors`.
 - Takes the `request` and `session` arguments.
- Its shortcut is `form.process().accepted`
 - No need to specify `request` and `session`.
 - `Process()` may take other arguments:
`onsuccess, onfailure, next, message_onsuccess,`
`message_onfailure ...`



```
form = SQLFORM(db.tablename, record=None)
```

```
db.define_table('registration',  
    Field('Name', requires=IS_NOT_EMPTY()),  
    Field('Age', requires=IS_NOT_EMPTY()),  
    Field('Gender', requires=IS_IN_SET(['Male', 'Female'])),  
    Field('username', requires=IS_NOT_EMPTY()),  
    Field('password', 'password'),  
    Field('about_you', 'text'),  
    Field('image', 'upload') )
```

- Display selected fields?

- Mimics SQLFORM as if there was a DB table.
- The values can be stored inside a `session` object ² (which is an instance of the `Storage` class).

```
def index():  
    form = SQLFORM.factory( Field("name", requires=IS_NOT_EMPTY() )  
                           Field("email", requires=IS_EMAIL() )  
                           )  
  
    if form.process().accepted:  
        response.flash= 'Welcome %s' % form.vars.name  
        session.your_name = form.vars.name  
        session.your_email = form.vars.email
```

²Its behavior is affected by a call to the `session.forget()` method 

The CRUD API

- Stands for Create/Read/Update/Delete.
- Can be called as “SQLFORM simplified”.
- Must be imported and linked to a specific DB (by instantiation).

```
from gluon.tools import Crud  
  
crud = Crud(db)
```

The CRUD Methods

- `crud.tables()` # list of tables
- `form = crud.create(db.tablename)` # an insert form
- `form = crud.read(db.tablename, record_id)` # a read-only form
- `form = crud.update(db.tablename, record_id)` # an update form
- `crud.delete(db.tablename, record_id)` # delete the record
- `crud.select(db.tablename, query)` # query the table
- `crud.search(db.tablename)` # a search form

```
db.define_table("student", Field("name", requires=IS_NOT_EMPTY()),  
                    Field("mailID", requires=IS_EMAIL()),  
                    Field("birthdate", "date")  
                )
```

```
def index():  
    return dict(form = crud.create(db.student))
```

```
def index():  
    form = crud.read(db.student, request.args(0))  
    return dict(form = form)
```

Now, try to pass an argument, say 2, to the controller as:

127.0.0.0:8000/[application]/[controller]/[function]/args

e.g.

http://127.0.0.1:8000/MyApp/default/index/2

request.args(0)=2

- Returns an update form with a delete checkbox.
- Submission updates the record.

```
def func():  
    form = crud.update(db.student, request.args(0))  
    return dict(form = form)
```

`http://127.0.0.1:8000/SomeApp/default/func/1`

- `crud.delete()` does NOT generate forms.
- Deletion + redirection

```
def func1():  
    form = crud.delete(db.student, request.args(0), \  
                        next = URL("func2") )  
    return dict(form = form)
```

```
def func2():  
    # do some thing here  
    return dict()
```

`http://127.0.0.1:8000/MyApp/default/func1/3`