

Flask Series: Templating

Friday, May 8, 2015 08:49 · 464 words · 3 minutes read

[PYTHON FLASK FLASK SERIES](#)

Flask Series

1. [Prepare the Environment](#)
2. [Structure the Application](#)
3. [Application Configuration](#)
4. [Templating](#)
5. [Model](#)
6. [Testing](#)
7. [Views and Web Forms](#)
8. [Error Management](#)
9. [Security](#)
10. [Optimizations](#)
11. [Healthcheck and Monitoring](#)
12. [Internationalization](#)
13. [Deployment](#)

Templating allows data representation in different way, it combines template and data. The template is a document with placeholders, where the actual data will be used, when the template is processed by the template engine. The template can contain control structures like for loops, if statements, etc.

By default Flask uses Jinja2 as its template engine. Jinja2 allows developers to produce different output result file based on simple template text file. In this blog post I will cover how to use:

- rendering templates
- variables
- comments
- control structures: for loops and if statements

- filters
- template inheritance and blocks

Rendering Templates

You should use the `render_template` method to achieve this goal, where you have to specify the template name and you could pass some data as keyword arguments:

```
1  from flask import Blueprint, render_template
2
3  @main.route('books/')
4  def display_books():
5      books = {
6          "Learn Python The Hard Way": {
7              "author": "Shaw, Zed",
8              "rating": "3.92",
9              "image": "ef0ceaab-32a8-47fb-ba13-c0b362d970da.jpg"
10         }
11     }
12
13     # passing data to the template
14     return render_template("books.htm", books=books)
```

main_controllers.py hosted with ❤ by GitHub

[view raw](#)

Variables

Variables are passed to the template via the context dictionary. In Jinja2 the double curly braces are used as a print statement. So if you want to print the value of the title variable, you should use the following code snippet:

```
1  {{title}}
```

variables.htm hosted with ❤ by GitHub

[view raw](#)

Should you want to access variable's attributes you have two approaches to achieve that goal:

```
1  {{book.title}}
```

variables.htm hosted with ❤ by GitHub

[view raw](#)

or

```
1  {{book['title']}}
```

variables.htm hosted with ❤️ by GitHub

[view raw](#)

In case of non-existent variable or attribute, an undefined value will be returned.

Comments

To be able to provide a comment in your template file or just to comment out a code block you should surround it with

```
1  {# your comment or code goes here #}  
2  
3  {# Checks if book is marked as hidden #}
```

comments.htm hosted with ❤️ by GitHub

[view raw](#)

or

```
1  {#  
2    {% for title, data in books.iteritems() %}  
3      {{title}}  
4    {% endfor %}  
5  #}
```

comments.htm hosted with ❤️ by GitHub

[view raw](#)

Control Structures: for loop

for control structure allows loop over a sequence or dictionary of items.

Sequence:

```
1  {% for data in books %}  
2    <tr>  
3      <td>{{data.author}}</td>  
4      <td>{{data.rating}}</td>  
5    </tr>  
6  {% endfor %}
```

foor_loop.htm hosted with ❤️ by GitHub

[view raw](#)

Control Structures: if

if statement is used for branching a logic internally in the template:

```
1 {% for title, data in books.iteritems() %}
2     {% Checks if book is marked as hidden %}
3     {%- if data.hidden %}{% continue %}{% endif %}
4     <tr>
5         <td>
6             
8         </td>
9         <td>{{title | upper}}</td>
10        <td>{{data.author}}</td>
11        <td>{{data.rating}}</td>
12    </tr>
13 {% endfor %}
```

if_structure.htm hosted with ❤ by GitHub

[view raw](#)

Multiple branches can be used in the following way:

```
1 {% if condition1 %}
2     Condition1 is true
3 {% elif condition2 %}
4     Condition2 is true
5 {% else %}
6     Default logic goes here
7 {% endif %}
```

if_structure.htm hosted with ❤ by GitHub

[view raw](#)

Filters

Filters allows developers to modify the value of the variable, they are separated by the variable with the pipe symbol '|'.

```
1 {{title|upper}}
```

filters.htm hosted with ❤ by GitHub

[view raw](#)

Filters can be chained:

```
1 {{title|trim|upper|}}
```

filters.htm hosted with ❤ by GitHub

[view raw](#)

Custom filters can be implemented and used in the templates:

```
@main.template_filter('trim_upper')
```

```

1
2 def string_trim_upper(value):
3     return value.strip().upper()

```

custom_filters.py hosted with ❤ by GitHub

[view raw](#)

And now you could use it in the following way:

```

1 {{title|trim_upper}}

```

filters.htm hosted with ❤ by GitHub

[view raw](#)

Template Inheritance and Blocks

Template inheritance allows developers to prepare a common layout for the application and to define blocks that could be overridden in the child templates. In the Flask Bookshelf application I have implemented a layout.htm, that contains the skeleton of the web application:

```

1 <!DOCTYPE html>
2 <html lang="en">
3     <head>
4         <title>Flask Bookshelf</title>
5         <meta charset="utf-8">
6         <link href="{{ url_for('static', filename='css/bootstrap.min.css') }}" rel="stylesheet">
7     </head>
8     <body>
9         <nav class="navbar navbar-inverse">
10             <div class="container-fluid">
11                 <div class="navbar-header">
12                     <a class="navbar-brand" href="#">Flask Bookshelf</a>
13                 </div>
14                 <div>
15                     <ul class="nav navbar-nav">
16                         <li><a class="navbar-brand" href="{{ url_for('main.index') }}">Home</a>
17                         <li><a class="navbar-brand" href="{{ url_for('main.display_books') }}">Books</a>
18                     </ul>
19                 </div>
20             </div>
21         </nav>
22         <div class="container">
23             {% block container %}{% endblock %}
24         </div>
25         <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js">

```

```

25
26     <script src="{{ url_for('static', filename='js/bootstrap.min.js') }}"></script>
27 </body>
28 </html>

```

layout.htm hosted with ❤ by GitHub

[view raw](#)

The child templates inherit the base one (layout.htm) and provide their specifics using the defined blocks:

```

1 {% extends 'layout.htm' %}
2 {% block container %}
3     <div class="top-pad">
4         Flask Bookshelf!
5     </div>
6 {% endblock %}

```

index.htm hosted with ❤ by GitHub

[view raw](#)

Below I will describe how to structure your Flask application to enable the Jinja2 template engine and use it internally.

```

1 ...
2 # specifies the main template folder for the application
3 app = Flask(__name__,
4             instance_path=get_instance_folder_path(),
5             instance_relative_config=True,
6             template_folder='templates')
7
8 # enable jinja2 extensions - i.e. continue in for loops
9 app.jinja_env.add_extension('jinja2.ext.loopcontrols')
10 ...

```

__init__.py hosted with ❤ by GitHub

[view raw](#)

Added a templates folder under bookshelf/main

```

1 ...
2 # specifies the template folder for the main blueprint
3 main = Blueprint('main', __name__, template_folder='templates')
4 ...

```

main_controllers.py hosted with ❤ by GitHub

[view raw](#)

Added a templates folder under bookshelf/admin

```
1  ...  
2  # specifies the template folder for the admin blueprint  
3  admin = Blueprint('admin', __name__, template_folder='templates')  
4  ...
```

admin_controllers.py hosted with ❤️ by GitHub

[view raw](#)

In the next blog post I will describe how to model your data in your Flask project and use database systems.

The complete demo application, described in this blog post, can be found [here](#).

 [Tweet](#)

 [Share](#)

0 Comments **Damyan's Blog****1 Login** ▾

Recommend Tweet Share

Sort by Best ▾

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

Be the first to comment.

ALSO ON DAMYAN'S BLOG

How to integrate OpenAccess ORM and NAnt · Technical blog of ...

1 comment • 2 years ago

**OpenAccess ORM** — Great article, thank you very much!**How to fix an AssertionError when trying to list Python packages with ...**

1 comment • 2 years ago

**Selva Prakash** — Thanks much. It was very helpful :)**Flask Series: Error Management**

1 comment • 4 years ago

**buryboi** — Thanks for the tutorial, it was very useful**Flask Series: Application Configuration**

9 comments • 4 years ago

**damyanbogoev** — Thank you for noticing it. It is a typo in both the blog post and the code. I will fix it.

Subscribe Add Disqus to your siteAdd DisqusAdd

Powered by [Hugo](#) © Copyright 2019 Damyan Bogoev

