

Python μ Frameworks

Bottle and Flask

The good, the bad and the ugly
(and the code)





- Author: Marcel Hellkamp
- Framework-in-a-file, bottle.py
- 3500 loc - YES, all in one file!



The Good

- One file means easy deploy
 - Include it in your repository



The Good

- One file means easy deploy
 - Include it in your repository
- Nice routing with function-call mapping
 - Bonus for regex style routes



The Good

- One file means easy deploy
 - Include it in your repository
- Nice routing with function-call mapping
 - Bonus for regex style routes
- Easy access to full request object including form data, file uploads, cookies, headers...



The Good

- One file means easy deploy
 - Include it in your repository
- Nice routing with function-call mapping
 - Bonus for regex style routes
- Easy access to full request object including form data, file uploads, cookies, headers...
- Python 2.5+ and 3.x compliant



The Good

- One file means easy deploy
 - Include it in your repository
- Nice routing with function-call mapping
 - Bonus for regex style routes
- Easy access to full request object including form data, file uploads, cookies, headers...
- Python 2.5+ and 3.x compliant
- Built-in development server



The Bad

- Built in template language is a bit sub-par
- Does include support for Mako, Jinja2 and Cheetah templates, but you handle that yourself.



The Bad

- Built in template language is a bit sub-par
- Does include support for Mako, Jinja2 and Cheetah templates, but you handle that yourself.
- Documentation



The Bad

- Built in template language is a bit sub-par
 - Does include support for Mako, Jinja2 and Cheetah templates, but you handle that yourself.
- Documentation
- Separation of views is “difficult”



The Ugly

- One file!
- WSGI/HTTP are incredibly complex. In fact, too complex to fit into a single Python file



The Ugly

- One file!
- WSGI/HTTP are incredibly complex. In fact, too complex to fit into a single Python file
- Documentation



The Ugly

- One file!
- WSGI/HTTP are incredibly complex. In fact, too complex to fit into a single Python file
- Documentation
- global request and response magic



- Author: Armin Ronacher
- Framework built around other tech
- 4940 loc - multiple files
 - 13.5K loc for werkzeug
 - 9K loc for jinja2



The Great

- Fully testable with Werkzeug test Client



The Good

- Documentation
 - Some of the best I have seen (for any project in any language)



The Good

- Documentation
 - Some of the best I have seen (for any project in any language)
- Built on top of existing tech
 - Werkzeug (<http://wsgi.org>)
 - Jinja2 (templating)



The Good

- Documentation
 - Some of the best I have seen (for any project in any language)
- Built on top of existing tech
 - Werkzeug (<http://wsgi.org>)
 - Jinja2 (templating)
- Built-in development server
 - Comes with built in debugger - courtesy of werkzeug



The Good

- Documentation
 - Some of the best I have seen (for any project in any language)
- Built on top of existing tech
 - Werkzeug (<http://werkzeug.pocoo.org/>)
 - Jinja2 (<http://jinja.pocoo.org/>)
- Built-in development server
 - Comes with built in debugger - courtesy of werkzeug
- Blueprints!



The Good

- Documentation
 - Some of the best I have seen (for any project in any language)
- Built on top of existing tech
 - Werkzeug (<http://werkzeug.pocoo.org/>)
 - Jinja2 (templating)
- Built-in development server
 - Comes with built in debugger - courtesy of werkzeug
- Blueprints!
- Extensions, extensions, extensions



The Bad

- Makes some decisions for you



The Bad

- Makes some decisions for you
- No regex routing built-in
 - Though easy to implement



The Bad

- Makes some decisions for you
- No regex routing built-in
 - Though easy to implement
- Pro-configured for jinja2
 - What if you love mako, genshi, cheetah...?



The Bad

- Makes some decisions for you
- No regex routing built-in
 - Though easy to implement
- Pro-configured for jinja2
 - What if you love mako, genshi, cheetah...?
- Doing tricky things can be... tricky
 - Have to use werkzeug directly



The Ugly



The Ugly





Which should I use?



Which should I use?

- Choose Bottle if...



Which should I use?

- Choose Bottle if...
 - Having the entire framework in a file will ease your deployment.



Which should I use?

- Choose Bottle if...
 - Having the entire framework in a file will ease your deployment.
 - You have little to no need for templates



Which should I use?

- Choose Bottle if...
 - Having the entire framework in a file will ease your deployment.
 - You have little to no need for templates
 - Flask seems too heavy weight... (?)



Which should I use?

- Choose Bottle if...
 - Having the entire framework in a file will ease your deployment.
 - You have little to no need for templates
 - Flask seems too heavy weight... (?)
- Choose Flask if...



Which should I use?

- Choose Bottle if...
 - Having the entire framework in a file will ease your deployment.
 - You have little to no need for templates
 - Flask seems too heavy weight... (?)
- Choose Flask if...
 - You don't have the above requirements



To the code!



Hello World

```
1 import bottle
2
3 @bottle.route("/")
4 def hello():
5     return "Hello World!"
6
7 if __name__ == "__main__":
8     bottle.run()
```



Hello Anyone

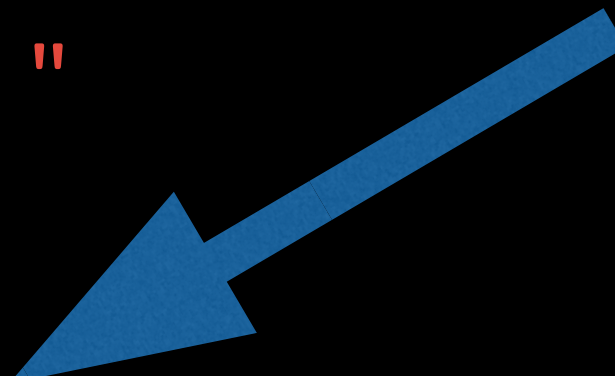
```
1 import bottle
2
3 @bottle.route("/")
4 def hello():
5     return "Hello World!"
6
7 @bottle.route("/hello")
8 @bottle.route("/hello/")
9 @bottle.route("/hello/<name>")
10 def hello_name(name="World"):
11     return "Hello {}".format(name)
12
13 if __name__ == "__main__":
14     bottle.run()
```



Hello Anyone

```
1 import bottle
2
3 @bottle.route("/")
4 def hello():
5     return "Hello World!"
6
7 @bottle.route("/hello")
8 @bottle.route("/hello/")
9 @bottle.route("/hello/<name>")
10 def hello_name(name="World"):
11     return "Hello {}".format(name)
12
13 if __name__ == "__main__":
14     bottle.run()
```

These are two
different things

A blue arrow pointing from the text "These are two different things" to the route decorators in the code, specifically highlighting the difference between the root route and the named routes.



Hello Middleware

```
1 import bottle
2
3 class StripPathMiddleware(object):
4     def __init__(self, app):
5         self.app = app
6     def __call__(self, e, h):
7         e['PATH_INFO'] = e['PATH_INFO'].rstrip('/')
8         return self.app(e, h)
9
10 @bottle.route("/")
11 def hello():
12     return "Hello World!"
13
14 @bottle.route("/hello")
15 @bottle.route("/hello/<name>")
16 def hello_name(name="World"):
17     return "Hello {}".format(name)
18
19 if __name__ == "__main__":
20     app = bottle.app()
21     modified_app = StripPathMiddleware(app)
22     bottle.run(app=modified_app)
```



Hello Middleware

```
1 import bottle
2
3 class StripPathMiddleware(object):
4     def __init__(self, app):
5         self.app = app
6     def __call__(self, e, h):
7         e['PATH_INFO'] = e['PATH_INFO'].rstrip('/')
8         return self.app(e, h)
9
10 @bottle.route("/")
11 def hello():
12     return "Hello World!"
13
14 @bottle.route("/hello")
15 @bottle.route("/hello/<name>")
16 def hello_name(name="World"):
17     return "Hello {}".format(name)
18
19 if __name__ == "__main__":
20     app = bottle.app()
21     modified_app = StripPathMiddleware(app)
22     bottle.run(app=modified_app)
```

This is middleware.

Two blue arrows originate from the text "This is middleware." and point towards the code. One arrow points to line 8 of the code, which is the return statement of the __call__ method in the StripPathMiddleware class. The other arrow points to line 21, which is the line where the StripPathMiddleware object is instantiated and assigned to modified_app.



WSGI: Brief introduction to Middleware

Middleware is simply a wrapper for an application

- It follows a spec laid out in PEPs 333, 3333.
- Basically uses `__call__` method (taking environment and output handler function), does some magic, and returns an invocation of the original app.



WSGI: Brief introduction to Middleware

Middleware can perform various tasks such as...

PEP 333
PEP 3333

- Routing a request to different application objects based on the target URL, after rewriting the environ accordingly.
- Allowing multiple applications or frameworks to run side-by-side in the same process
- Load balancing and remote processing, by forwarding requests and responses over a network
- Perform content postprocessing, such as applying XSL stylesheets



Back to the code!



Debugging Middleware

```
1 import bottle
2 from werkzeug.debug import DebuggedApplication
3 from middleware import StripPathMiddleware
4
5
6 @bottle.route("/fail")
7 def fail():
8     user = bottle.request.params.get("name")
9     some_data = {"sean": "admin", "joe": "user"}
10    return "{} is of role {}".format(user, some_data[user])
11
12 if __name__ == "__main__":
13     app = bottle.app()
14     app.catchall = False
15     debugger_app = DebuggedApplication(app=app, evalex=True)
16     modified_app = StripPathMiddleware(debugger_app)
17     bottle.run(app=modified_app, debug=True)
```



Debugging Middleware

```
1 import bottle
2 from werkzeug.debug import DebuggedApplication
3 from middleware import StripPathMiddleware
4
5
6 @bottle.route("/fail")
7 def fail():
8     user = bottle.request.params.get("name")
9     some_data = {"sean": "admin", "joe": "user"}
10    return "{} is of role {}".format(user, some_data[user])
11
12 if __name__ == "__main__":
13     app = bottle.app()
14     app.catchall = False
15     debugger_app = DebuggedApplication(app=app, evalex=True)
16     modified_app = StripPathMiddleware(debugger_app)
17     bottle.run(app=modified_app, debug=True)
```



Hey Now!
This is from Flask

Middleware for everything

- Gzip content based on mime-type

Middleware for everything

- Gzip content based on mime-type
- Append slash to routes

Middleware for everything

- Gzip content based on mime-type
- Append slash to routes
- Dispatch (different routes to different apps)

Middleware for everything

- Gzip content based on mime-type
- Append slash to routes
- Dispatch (different routes to different apps)
- Shared Data

Middleware for everything

- Gzip content based on mime-type
- Append slash to routes
- Dispatch (different routes to different apps)
- Shared Data
- Debugger

Middleware for everything

- Gzip content based on mime-type
- Append slash to routes
- Dispatch (different routes to different apps)
- Shared Data
- Debugger
- Authentication / Authorization
- etc, etc, etc...



Back to the code!

... again



Streaming Data

Yield from handler to stream data

```
1 import bottle
2 import time
3
4 @bottle.route("/beer")
5 def hello():
6     for i in range(100, 0, -1):
7         if i < 100:
8             yield "Take one down, pass it around, " \
9                 "{} bottles of beer on the wall\n".format(i)
10            time.sleep(.2)
11            yield "{} bottles of beer on the wall, " \
12                "{} bottles of beer\n".format(i, i)
13
14
15 if __name__ == "__main__":
16     bottle.run()
```



The Codez...

Too much for a slide... let's get crazy in the terminal!



The Other Awesomez...

- Over 50 Extensions
- My top favorites:
 - Flask-Cache
 - Flask-Classy
 - Flask-DebugToolbar
 - Flask-Gravatar
 - Flask-Login
 - Flask-Oauth / Flask-OpenID
 - Flask-Principal
 - Flask-Script
 - Flask-SQLAlchemy
 - Flask-WTF



The Other Awesomez...

- Flask-API
 - Written by author of Django-Rest-Framework
 - But written atop Flask
 - Still very early (pre 1.0)
 - But has an aggressive roadmap
 - <http://www.flaskapi.org/>

`sys.exit(0)`

- Questions?



WE'RE HIRING

- Great pay, benefits, stock options, and bonus plan
- Top notch developers
- Challenging problems
- Large scale distributed cloud architecture
- We use Python
- “Hilarious quotes” wiki page from IRC logs

crowdstrike.com/about-us/careers/