mimerender Documentation

Release 0.4

Martin Blech

July 02, 2012

CONTENTS

mimerender is a Python module for RESTful resource variant selection using the HTTP Accept header

It acts as a decorator that wraps a HTTP request handler to select the correct render function for a given HTTP Accept header. It uses mimeparse to parse the accept string and select the best available representation.

Support for webapp2 (Google App Engine), web.py, Flask and Bottle is available out of the box and it's easy to add support for your favourite framework, just extend the `MimeRenderBase` class.

CHAPTER

ONE

CONTENTS

1.1 Usage Examples

1.1.1 Content-Type selection

A few examples on how to use `mimerender` with the different supported frameworks. Any of these will behave this way:

```
$ curl -iH "Accept: application/html" localhost:8080/x
...
Content-Type: text/html
...
<html><body>Hello, x!</body></html>

$ curl -iH "Accept: application/xml" localhost:8080/x
...
Content-Type: application/xml
...
<message>Hello, x!</message>

$ curl -iH "Accept: application/json" localhost:8080/x
...
Content-Type: application/json
...
{"message": "Hello, x!"}

$ curl -iH "Accept: text/plain" localhost:8080/x
...
Content-Type: text/plain
...
Hello, x!
```

Bottle

```
from bottle import Bottle, run
try:
    import simplejson as json
except ImportError:
    import json
import mimerender
```

```
mimerender = mimerender.BottleMimeRender()

render_xml = lambda message: '<message>%s</message>'%message
render_json = lambda **args: json.dumps(args)
render_html = lambda message: '<html><body>%s</body></html>'%message
render_txt = lambda message: message

app = Bottle()

@app.route('/')
@app.route('/<name>')
@mimerender(
    default = 'html',
    html = render_html,
    xml  = render_xml,
    json = render_json,
    txt  = render_txt
)
def greet(name='world'):
    return {'message': 'Hello, ' + name + '!'}

if __name__ == "__main__":
    run(app, host='localhost', port=8080)
```

Flask

```
from flask import Flask
try:
    import simplejson as json
except ImportError:
    import json
import mimerender

mimerender = mimerender.FlaskMimeRender()

render_xml = lambda message: '<message>%s</message>'%message
render_json = lambda **args: json.dumps(args)
render_html = lambda message: '<html><body>%s</body></html>'%message
render_txt = lambda message: message

app = Flask(__name__)

@app.route('/')
@app.route('/<name>')
@mimerender(
    default = 'html',
    html = render_html,
    xml  = render_xml,
    json = render_json,
    txt  = render_txt
)
def greet(name='world'):
    return {'message': 'Hello, ' + name + '!'}

if __name__ == "__main__":
    app.run(port=8080)
```

Webapp2

```
import webapp2
try:
    import simplejson as json
except ImportError:
    import json
import mimerender

mimerender = mimerender.Webapp2MimeRender()

render_xml = lambda message: '<message>%s</message>'%message
render_json = lambda **args: json.dumps(args)
render_html = lambda message: '<html><body>%s</body></html>'%message
render_txt = lambda message: message

class Greet(webapp2.RequestHandler):
    @mimerender(
        default = 'html',
        html = render_html,
        xml  = render_xml,
        json = render_json,
        txt  = render_txt
    )
    def get(self, name):
        if not name:
            name = 'world'
        return {'message': 'Hello, ' + name + '!'}

app = webapp2.WSGIApplication([
    ('/(.*)', Greet),
], debug=True)

def main():
    from paste import httpserver
    httpserver.serve(app, host='127.0.0.1', port='8080')

if __name__ == '__main__':
    main()
```

web.py

```
import web
try:
    import simplejson as json
except ImportError:
    import json
import mimerender

mimerender = mimerender.WebPyMimeRender()

render_xml = lambda message: '<message>%s</message>'%message
render_json = lambda **args: json.dumps(args)
render_html = lambda message: '<html><body>%s</body></html>'%message
render_txt = lambda message: message

urls = (
```

```
    '/(.*)', 'greet'
)
app = web.application(urls, globals())

class greet:
    @mimerender(
        default = 'html',
        html = render_html,
        xml  = render_xml,
        json = render_json,
        txt  = render_txt
    )
    def GET(self, name):
        if not name:
            name = 'world'
        return {'message': 'Hello, ' + name + '!'}

if __name__ == "__main__":
    app.run()
```

1.1.2 Content-Type selection plus Exception Mapping

`mimerender` provides a helper decorator for mapping exceptions to HTTP Status Codes.

```
import webapp2
try:
    import simplejson as json
except ImportError:
    import json
import mimerender

mimerender = mimerender.Webapp2MimeRender()

render_xml = lambda message: '<message>%s</message>'%message
render_json = lambda **kwargs: json.dumps(kwargs)

render_xml_exception = lambda exception: '<exception>%s</exception>'%exception
render_json_exception = lambda exception: json.dumps(exception.args)

class NotFound(Exception): pass

class Greet(webapp2.RequestHandler):
    @mimerender.map_exceptions(
        mapping=(
            (ValueError, '500 Internal Server Error'),
            (NotFound, '404 Not Found')
        ),
        xml  = render_xml_exception,
        json = render_json_exception,
    )
    @mimerender(
        xml  = render_xml,
        json = render_json,
    )
    def get(self, id_):
        int_id = int(id_)
        if int_id != 1:
```

```
            raise NotFound('could not find item with id = %d' % int_id)
        return dict(message='found it!')

app = webapp2.WSGIApplication([
    ('/(.*)', Greet),
], debug=True)

def main():
    from paste import httpserver
    httpserver.serve(app, host='127.0.0.1', port='8080')

if __name__ == '__main__':
    main()
```

`mimerender` will take care of mapping `ValueError` and `NotFound` to the specified HTTP status codes, and it will serialize the exception with an acceptable Content-Type:

```
$ curl -iH "Accept: application/xml" localhost:8080/1
HTTP/1.0 200 OK
...
Content-Type: application/xml
...
<message>found it!</message>

$ curl -iH "Accept: application/xml" localhost:8080/2
HTTP/1.0 404 Not Found
...
Content-Type: application/xml
...
<exception>could not find item with id = 2</exception>

$ curl -iH "Accept: application/xml" localhost:8080/abc
HTTP/1.0 500 Internal Server Error
...
Content-Type: application/xml
...
<exception>invalid literal for int() with base 10: 'abc'</exception>

$ curl -iH "Accept: application/json" localhost:8080/1
HTTP/1.0 200 OK
...
Content-Type: application/json
...
{"message": "found it!"}

$ curl -iH "Accept: application/json" localhost:8080/2
HTTP/1.0 404 Not Found
...
Content-Type: application/json
...
["could not find item with id = 2"]

$ curl -iH "Accept: application/json" localhost:8080/abc
HTTP/1.0 500 Internal Server Error
...
Content-Type: application/json
...
["invalid literal for int() with base 10: 'abc'"]
```

1.2 API

class mimerender.**MimeRenderBase**(*global_default=None*, *global_override_arg_idx=None*, *global_override_input_key=None*, *global_charset=None*, *global_not_acceptable_callback=None*)

__call__(*default=None*, *override_arg_idx=None*, *override_input_key=None*, *charset=None*, *not_acceptable_callback=None*, ***renderers*)

Main mimerender decorator. Usage:

```
@mimerender(default='xml', override_arg_idx=-1, override_input_key='format', , <renderers>)
GET(self, ...) (or POST, etc.)
```

The decorated function must return a dict with the objects necessary to render the final result to the user. The selected renderer will be called with the dict contents as keyword arguments. If override_arg_idx isn't None, the wrapped function's positional argument at that index will be used instead of the Accept header. override_input_key works the same way, but with web.input().

Example:

```
@mimerender(
    default = 'xml',
    override_arg_idx = -1,
    override_input_key = 'format',
    xhtml   = xhtml_templates.greet,
    html    = xhtml_templates.greet,
    xml     = xml_templates.greet,
    json    = json_render,
    yaml    = json_render,
    txt     = json_render,
)
def greet(self, param):
    message = 'Hello, %s!'%param
    return {'message':message}
```

map_exceptions(*mapping*, **args*, ***kwargs*)

Exception mapping helper decorator. Takes the same arguments as the main decorator, plus *mapping*, which is a list of *(exception_class, status_line)* pairs.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*