Flask-Holster Documentation

Release 0.1.1

Corbin Simpson

Sep 27, 2017

Contents

CHAPTER 1

Introduction

Flask-Holster, or Holster for short, is a simple Flask extension for performing automated content negotiation with rigid MVC on ordinary request handlers.

CHAPTER 2

Holster in 3 Lines

In a nutshell:

```
@app.holster("/route/<arg>")
def route(arg):
    return {"argument": arg}
```

CHAPTER 3

What just happened?

This snippet creates a route, `/route/<arg>` in the same way as a normal Flask route, and it also creates an extra route, `/route/<arg>.<ext>`. This extra route can capture a file extension.

When a request comes in for this handler, Holster determines which type of data it will return, and then runs the handler normally. The handler returns a **mapping** of some sort, like a `dict`, and then Holster renders that mapping into the desired type and returns it with the appropriate headers.

In MVC terms, Holster separates the model and view in a rigid, completely enforced manner. The data returned from the request handler is independent of, and compatible with, all of the view renderers which can render it.

CHAPTER 4

Why is this a good thing?

The original motivation here is creating RESTful APIs. For many modern programmers working on the Web, "REST API" is a buzzphrase that deciphers roughly to "stateless JSON-based low-level API over HTTP *which is not the same as my actual website*." This is a very unfortunate reading because HTTP and REST are vastly more powerful and flexible than this. When the Web was formalized, a few forward-thinking framers wrote in the ability for the user agent (a client) and the Web application (a server) to **negotiate content metadata**, including language, encoding, format, and so forth.

Holster can return both JSON and HTML formats from a given set of data. This means that programmers can write a single site once, and instantly gain a JSON-based interface for free.

Of course, there's no such thing as a free lunch. The site still probably needs to be structured in a way that fits how you want the data to be formatted. Even so, Holster makes this easy.

CHAPTER 5

Where should I get started?

Holster's enhancements are per-handler. If you want to dive in and try out Holster on your site, it's only three simple steps.

Step 1: Download Holster

Holster is on PyPI; run `pip install Flask-Holster`.

Step 2: Holsterize your App

When you create your Flask application object, initialize Holster:

```
from flask import Flask
from flask.ext.holster.main import init_holster

app = Flask(__name__)
init_holster(app)
```

Step 3: Holsterize your Handlers

Simply go from this:

```
@app.route("/<foo>/<bar>/baz")
def baz(foo, bar):
    return render_template("baz.html", foo=foo, bar=bar)
```

To this:

```
from flask.ext.holster.simple import html

@app.holster("/<foo>/<bar>/baz")
@html("baz.html")
def baz(foo, bar):
    return {"foo": foo, "bar": bar}
```

Writing Custom Renderers

Holster's flexibility and power is in the hands of developers with custom renderers. Rather than force users to convolute their data, Holster permits every handler to register a custom renderer for any format.

Renderers are pretty simple. Extremely simple, actually. Any object which has a callable attribute named `format()` is a renderer. `format()` will be called with a single argument, which is the mapping of data to render.

As an example, here is the default JSON renderer, which simply dumps the entire mapping using the `json` module:

```
from flask import json

class JSONRenderer(object):
    def format(self, d):
        return json.dumps(d)

json_renderer = JSONRenderer()
```

And that's it! This renderer is implemented as a class in case it needs to be customized in the future, but otherwise it should be completely obvious how it works. Aside from some simplifications, this is actually the code that renders JSON inside Holster. It's that easy!

Escaping Holster

Holster is rigid. This means that there will be times when one wishes to do things, but cannot do them because Holster is in the way. Fortunately, it is possible to escape Holster's control.

Step 0: Do You Need Holster?

First, ask yourself: "Do I *need* Holster for this view?" Holster only operates on a per-view basis, and each view must explicitly ask for holstering via `holster()` (or other mechanisms, detailed below.) If you don't need Holster, don't use it. It's okay; Holster doesn't have feelings and won't feel neglected if you don't use it on a view.

Step 1: Altering the Response

So, you need to adjust the response slightly. Maybe there's a header that you need to set, or you want to do some brief accounting. Whatever the reason is, you want to do it on one single specific view, and so you don't want to abuse Flask's global response hooks.

The solution is easy. Break Holster's decorator into two steps, and insert your function between them.

Starting with something like:

```
@app.holster("/escape")
def escape():
    return {}
```

This is equivalent:

```
@app.bare_holster("/escape")
@app.holsterize
def escape():
    return {}
```

And now, using `lift()` from `flask.ext.holster.helpers`, let's disable caching, as an example:

```
def no_cache(response):
    response.cache_control.no_cache = True
    return response

@app.bare_holster("/escape")
@lift(no_cache)
@app.holsterize
def escape():
    return {}
```

Tips & Tricks

Defining a New Format

Adding new renderers for a format is done in an ad-hoc manner. Users can also override renderers for any format or MIME type they would like, with `with_template`:

```
from my_sweet_website import PNGHeaderMaker
from flask.ext.holster.main import with_template

@app.holster("/customized")
@with_template("image/png", PNGHeaderMaker)
def custom():
    return {"header": "Welcome to my site!"}
```

There isn't currently a way to register renderers which cover an entire application. That should really be fixed at some point...

Forcing a Renderer

Sometimes one wants `url_for()` to pick a particular renderer for a target endpoint, instead of letting Holster and the user agent negotiate a particular format. This is relatively straightforward; just adjust the endpoint slightly and explain which extension you want to use:

```
from flask import url_for

@app.holster("/force")
def force():
    return {
        "url": url_for("sabers", style="cutlass"),
        "html_url": url_for("sabers-ext", ext="html", style="cutlass"),
    }
```

In this example, the `"sabers-ext"` endpoint is just like `"sabers"`, but requires an additional argument, `ext`, which contains the extension to use. This allows generation of URLs for endpoints which have fine-grained control over the preferred output formats.

API

Settings

- **HOLSTER_COMPRESS** *default: False*

Whether to attempt compression of response data if the browser indicates support for it. This feature is disabled by default due to the potential for high CPU load. It can be useful for sites that have high bandwidth costs and very compressible data.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search