CherryPy

Seth House <seth@eseth.com>

Utah Python

2013-05-09

Outline

1. A minimalist Python web framework
2. CherryPy basics
3. Building an app
4. Conclusion

History

- Remi Delon

History

- Remi Delon
- 2002: CherryPy
  - CherryPy class processed to be self-contained module (app & server)

History

- Remi Delon
- 2002: CherryPy
  - CherryPy class processed to be self-contained module (app & server)
- 2004: CherryPy 2
  - Object publishing
  - Filters

History

- Remi Delon
- 2002: CherryPy
  - CherryPy class processed to be self-contained module (app & server)
- 2004: CherryPy 2
  - Object publishing
  - Filters
- 2005: CherryPy 2.1
  - Shipped with Turbogears
  - Scrutiny; performance; WSGI support

History

- Remi Delon
- 2002: CherryPy
  - CherryPy class processed to be self-contained module (app & server)
- 2004: CherryPy 2
  - Object publishing
  - Filters
- 2005: CherryPy 2.1
  - Shipped with Turbogears
  - Scrutiny; performance; WSGI support
- 2006: CherryPy 3
  - Book
  - Turbogears 2.x chose Pylons

History

- Remi Delon
- 2002: CherryPy
  - CherryPy class processed to be self-contained module (app & server)
- 2004: CherryPy 2
  - Object publishing
  - Filters
- 2005: CherryPy 2.1
  - Shipped with Turbogears
  - Scrutiny; performance; WSGI support
- 2006: CherryPy 3
  - Book
  - Turbogears 2.x chose Pylons
- 2011: CherryPy 3.2

Why CherryPy

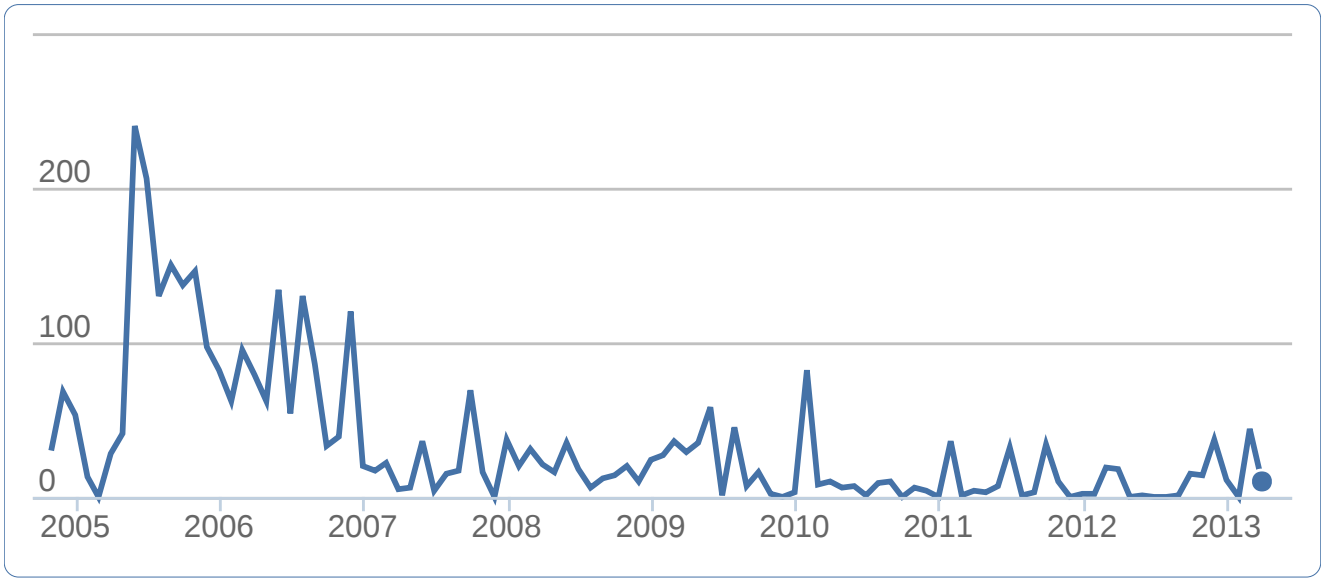

Figure: `https://www.ohloh.net/p/cherrypy`

Goals

- Simplicity
- No deps
- Lightly opinionated
- Community driven

Features

- Small (~600k)
- Featureful
- Plain functions or objects
- Extremely extendible

Outline

1 A minimalist Python web framework

2 CherryPy basics

3 Building an app

4 Conclusion

Basics

- Application framework
- Webserver

Framework

- Configuration via dictionaries
- Seven hook functions
  - Called during request/response cycle
- Caching
- Encoding
- Sessions & cookies
- Authorization
- Uploads
- Static content

Framework

- Configuration via dictionaries
- Seven hook functions
  - Called during request/response cycle
- Caching
- Encoding
- Sessions & cookies
- Authorization
- Uploads
- Static content
- No ORM / no templating / no forms

Server

- Pure Python (Python 2.3+)
- HTTP/1.1 compliant
- Thread pooled
- Fast (1-2 ms per request)
- SSL (!)
- Cheroot: Stand-alone version

Hello world

Application and server:

hello.py

```
import cherrypy

class HelloWorld:
    def index(self):
        return "Hello world!"
    index.exposed = True

cherrypy.quickstart(HelloWorld())
```

import cherrypy

```
help(cherrypy)
```

Outline

1 A minimalist Python web framework

2 CherryPy basics

3 Building an app

4 Conclusion

Dispatchers

- Determine handler (via app config and URI)
- Set `cherrypy.request.handler`
- Wraps handler
- Collect config in `cherrypy.request.config`
- Extremely open-ended
- Use a custom by setting `request.dispatch`
- `http://docs.cherrypy.org/stable/refman/_cpdispatch.html`

Default dispatcher

For example:

- `http://localhost/` -> `root.index`
- `http://localhost/onepage` -> `root.onepage`
- `http://localhost/some/page` -> `root.some.page`

Or possibly:

- `http://localhost/blog/2005/01/17` -> `root.blog(self, year, month, day)`

Handlers

- Any callable
- Current handler is `cherrypy.request.handler`
- Replace on the fly
- Must have `exposed=True` attribute

Handlers

- Any callable
- Current handler is `cherrypy.request.handler`
- Replace on the fly
- Must have `exposed=True` attribute
- `.index` attribute will take precedence
- `default` callable as fallback

Handlers

- Any callable
- Current handler is `cherrypy.request.handler`
- Replace on the fly
- Must have `exposed=True` attribute
- `.index` attribute will take precedence
- `default` callable as fallback
- `POST` data available as kwargs:

```
class MyHandler(object):
    def search(self, q, lang, page):
        # do something with ''q''
```

Config

- Dictionaries (!)
  - Python code (run-time)
  - ConfigParser ini files
  - Python code (execution-time)

Config

- Dictionaries (!)
  - Python code (run-time)
  - ConfigParser ini files
  - Python code (execution-time)
- Configure
  - Dispatcher (per URL)
  - `request` / `response` object attributes
  - Hooks
  - Tools
  - Logging
  - Server options

Config

- Dictionaries (!)
  - Python code (run-time)
  - ConfigParser ini files
  - Python code (execution-time)
- Configure
  - Dispatcher (per URL)
  - `request` / `response` object attributes
  - Hooks
  - Tools
  - Logging
  - Server options
- Global config
- Application config
- Handler config:

```
class MyHandler(object):
    _cp_config = {}
```

Tools

- Behavior outside handlers
- Many builtin
- Register / enable in the config:

```
[/images]
tools.staticdir.on: True
```

Tools

- Behavior outside handlers
- Many builtin
- Register / enable in the config:

```
[/images]
tools.staticdir.on: True
```

- Some usable as handler
- Directly callable

Tools

- Behavior outside handlers
- Many builtin
- Register / enable in the config:

```
[/images]
tools.staticdir.on: True
```

- Some usable as handler
- Directly callable
- Usable as decorators:

```
@tools.staticdir(dir='static')
def images():
    ...
```

Writing tools

```
def mytool():
    # something!

cherrypy.tools.mytool = Tool('on_some_hook', mytool)
```

Hooks

- `on_start_resource`
- `before_request_body`
- `before_handler`
- `before_finalize`
- `on_end_resource`
- `before_error_response`
- `after_error_response`
- `on_end_request`

Outline

1 A minimalist Python web framework

2 CherryPy basics

3 Building an app

4 Conclusion

Good

- Crazy flexible
- Lots of documentation
- Small
- Fast
- No deps

Bad

- Documentation is scattered / some holes
- CherryPy 2 vs. 3 vs. 3.2
- Small community
- Flexibility style can be tough to organize

Walkthrough of a real app

- Demo of REST interface to Salt
  - Request
  - Response
  - Reading headers
  - Writing headers
  - Redirects
  - Exceptions
- WSGI Server
- Writing tests