Developing Applications With The
Web Server Gateway Interface

James Gardner

EuroPython 3rd July 2006

www.3aims.com

Aims

- Show you how to write WSGI applications
  - Quick recap of HTTP, then into the nitty gritty
- Show you the benefits of WSGI
  - Applications can run on multiple servers
  - Middleware can be reused easily so you don't have to write functionality yourself
- Leave you inspired by the concept to go away to write adaptors, middleware or applications or contribute to existing projects.

Python Enhancement Proposal

- http://www.python.org/dev/peps/pep-0333/
- Drawn up two and half years old, written back in 2003, last modified April this year
- Submitted by P J Eby, after discussion with others on the Web-SIG mailing list like Ian Bicking

The Problem

- Lots of web frameworks Zope, Quixote, Webware, SkunkWeb and Twisted Web etc
- Applications written for one framework often weren't compatible with the server components of the others
- Made the choosing a Python web framework hard as there were so many different and incompatible options
- The PEP compares the situation to Java which had its Servelet API

The Solution

The abstract of PEP 333 states:

"This document specifies a proposed standard interface between web servers and Python web applications or frameworks, to promote web application portability across a variety of web servers."

getting started

Get the right tools

- Mozilla Firefox
  http://www.mozilla.com/firefox/
- LiveHTTPHeaders
  http://livehttpheaders.mozdev.org/
  View->Sidebar->LiveHTTPHeaders
- wsgiref
  http://peak.telecommunity.com/wsgiref_docs/

HTTP Basics

- When you request a page the browser sends an HTTP request
- When the server receives that request it will perform some action, (typically running an application) and return an HTTP response
- There are different request methods such as GET and POST.

An HTTP Request

```
GET /screenshots.html HTTP/1.1
Host: livehttpheaders.mozdev.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-
   US; rv:1.8.0.4) Gecko/20060508 Firefox/1.5.0.4
Accept:
   text/xml,application/xml,application/xhtml+xml,text/html;q=
   0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en,en-us;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

HTTP Response Headers

```
HTTP/1.x 200 OK
Date: Fri, 30 Jun 2006 12:09:34 GMT
Server: Apache/1.3.33 (Unix) mod_gzip/1.3.26.1a PHP/4.3.11
Vary: Host
X-Powered-By: PHP/4.3.11
Connection: close
Content-Type: text/html
Content-Encoding: gzip
Content-Length: 2752
```

A Simple Complete Response

```
HTTP/1.x 200 OK
Server: SimpleHTTP/0.6 Python/2.4.1
Content-Type: text/html

<html><body>Hello World!</body></html>
```

```
print "Content-Type: text/html\n\n"
print "<html><body>Hello World!</body></html>"
```

here it comes…

```
print "Content-Type: text/html\n\n"
print "<html><body>Hello World!</body></html>"
```

```
def application(environ, start_response):
    start_response('200 OK',[('Content-type','text/html')])
    return ['<html><body>Hello World!</body></html>']
```

(Also exc_info=None, write(body_data) callable)

Recap: What makes this a WSGI application

- It is a callable (in this case a simple function) taking environ and start_response as positional parameters
- It calls start_response() with a status code and a list of tuple pairs of headers before it returns a value. It should only be called once.
- The response it returns is an iterable (in this case a list with just one string).

The environ Dictionary

- A dictionary of strings
  - CGI strings
  - WSGI strings: wsgi.version, wsgi.url_scheme, wsgi.input, wsgi.errors, wsgi.multithread, wsgi.multiprocess, wsgi.run_once
  - Server extension strings, which we'll talk about later
- From the information in environ you can build any web application

that's it

so lets test it

Testing Our Application

Since we took so much time understanding and writing our application we should really test it.

wsgiref

```
> easy_install wsgiref

from wsgiref import simple_server
from hello_wsgi import application
httpd = simple_server.WSGIServer(
    ('',8000),
    simple_server.WSGIRequestHandler,
)
httpd.set_app(application)
httpd.serve_forever()
```

Actual HTTP Output

```
HTTP/1.x 200 OK
Date: Fri, 30 Jun 2006 12:59:51 GMT
Server: WSGIServer/0.1 Python/2.4.2
Content-Type: text/html
Content-Length: 38
```

You can now run this application on lots of WSGI compliant servers

Also as a server developer you know that just implementing WSGI is enough to make your server compatible with most applications and frameworks

flup

(also supports ajp or scgi)

```
> easy_install flup
```

```
from flup.server.fcgi import WSGIServer
from hello_wsgi import application
WSGIServer(application).run()
```

WSGIUtils

```
> easy_install WSGIUtils

from wsgiutils import wsgiServer
from hello_wsgi import application
server = wsgiServer.WSGIServer (
    ('localhost', 1088),
    {'/': application},
)
server.serve_forever()
```

CGI

You can even run your WSGI application as a CGI script.

```
from wsgiref.handlers import CGIHandler
from hello_wsgi import application
CGIHandler().run(application)
```

middleware

Middleware

- Component that acts like an application from the server's point of view
  - It is a callable that accepts environ and start_response
  - Calls start_repsonse once with status and headers etc
  - Returns an iterable
- Looks like a server to another piece of middleware or an application
  - Provides start_response and environ dictionaries
  - Expects a response

Changing the status or headers

```
class MyStatusMiddleware:
    def __init__(self, app):
        self.app=app
    def __call__(self, environ, start_response):
        def fake_start_response(status, headers, exc_info=None):
            if status[:3] == '200':
                status = '200 But I could have made it anything'
            return start_response(status, headers, exc_info)
        return self.app(environ, fake_start_response)

application = MyStatusMiddleware(application)
```

What can you do with it?

- Fundamentally you can do the following
  - Provide more functionality by adding a key to the environ dictionary
  - Change the status
  - Intercepting an error
  - Adding/removing/changing headers
  - Change a response

- These in turn allow you to all sorts of other clever things:
  - Provide error documents
  - Email error reports
  - Interactive debugging
  - Request forwarding
  - Etc etc.
  - Testing a component

An Example

```
from beaker.session import SessionMiddleware

def application(environ, start_response):
    session = environ['beaker.session']
    if not session.has_key('value'):
        session['value'] = 0
    session['value'] += 1
    session.save()
    start_response('200 OK', [('Content-type', 'text/plain')])
    return ['The current value is: %d' % session['value']]

application = SessionMiddleware(
    application,
    key='mysession',
    secret='randomsecret',
)
```

Middleware Chains

```
application = MyApplication(['Example','Chaining'])
application = MyStatusMiddleware(application)
application = MyEnvironMiddleware(application, 'Hi!')
```

or

```
MyEnvironMiddleware(
    MyStatusMiddleware(
        MyApplication(['Example','Chaining'])
    ),
   'Hi!',
)
```

This is Really Powerful!

- Suddenly you can add a single component to your application and get loads of functionality
- Hard for a user to configure -> Paste deploy
  - Lets you specify config files
  - Turns the settings in the config files into WSGI apps:

```
from paste.deploy import loadapp
wsgi_app = loadapp('config:/path/to/config.ini')
```

Error Handling

```
...
from paste.cgitb_catcher import CgitbMiddleware
app = CgitbMiddleware(app,{'debug':False})
#app = CgitbMiddleware(app,{'debug':True})
...
```

or

```
...
from paste.evalexception import EvalException
app = EvalException(app)
...
```

demo

Future..

- More projects adopting WSGI
- More middleware components springing up
- An active community at wsgi.org
- Web frameworks with full WSGI stacks so that you can pick and choose the best components for your particular needs. (See Pylons).
- A real alternative to Rails for rapid web development?

If middleware can be both simple and robust, and WSGI is widely available in servers and frameworks, it allows for the possibility of an entirely new kind of Python web application framework: one consisting of loosely-coupled WSGI middleware components. Indeed, existing framework authors may even choose to refactor their frameworks' existing services to be provided in this way, becoming more like libraries used with WSGI, and less like monolithic frameworks. This would then allow application developers to choose "best-of-breed" components for specific functionality, rather than having to commit to all the pros and cons of a single framework.

Summary

- WSGI isn't too complicated
- If your app is WSGI compliant you can instantly deploy it on a number of servers
- There are lots of powerful tools and middleware already in existence and you can easily re-use them -> see wsgi.org
- I'll be talking about Pylons later today which is one of the first projects to use WSGI throughout its stack.

Resources

- PEP 333
  - http://www.python.org/dev/peps/pep-0333/
- Paste Website
  - www.pythonpaste.org
- WSGI Website
  - www.wsgi.org
- Web-SIG Mailing List
  - http://www.python.org/community/sigs/current/web-sig/

questions?

james@pythonweb.org

thank you