Web Development with CherryPy

Jeffrey Clement

July 8, 2003

1 Overview

- What is CherryPy?
- Design Philosophy
- Installation
- Classes, Masks, Views, Functions and URLs
- ”Hello World!”
- Compiling
- The HTTP Server
- ”Hello World! v2”
- Standard Cherry Classes
- Example: Guestbook
- Why CherryPy?
- References

2 What is CherryPy?

- A Python based web application development toolkit: libraries, and compiler
- Allows us to build web applications in Python in much the same way we would build an other Python program
- CherryPy applications are compiled into a standalone Python app complete with all your program logic and a web server (really, really easy deployment)
- My favorite web development tool ever! (not that I’m biased :)

3 Design Philosophy

- Speed - compiled into one app, runs from memory
- Scalability - simply start more processes on one or more machines and add a load balancer
- Web app is just a normal Python app - cross platform deployment
- Developers can use OOP
- Templating Language (putting HTML in code is annoying) - allows web developers to maintain look and feel without seeing (much) code.

4 Noteworthy Features

- Good documentation and examples
- Standard library provides some very helpful things like authentication, database connectivity, etc
- Supports SSL using PyOpenSSL
- Supports XML-RPC calls to server (really easy way to setup an XMLRPC server!)
- CherryPy applications are a simple standalone Python script that runs anywhere Python does
- Simple. Learning and using CherryPy is very quick and intuitive.

5 Installation

Debian (I love Debian):

- (root) apt-get install cherrypy (it's in testing)
- Then to invoke the compiler just call cherrypy. It's in the path.

Other Unix / Win:

- (user) tar -xzvf cherrypy-XXX.tgz
- Just put your project under the cherrypy directory and invoke the compiler with ../cherrypy.py.

6 Classes, Views, Masks and Functions

- CherryPy adds a few new constructs to the Python language:
  - CherryClasses: A Class construct for web apps. Requests are mapped to method calls on CherryClasses

  - Views: Pure python code that can be called by browser. This is just a normal Python method that returns a string of HTML
  - Masks: Method which consists of HTML marked up with templating code. Easier than embedding HTML code in Python code such as we would do in a View. Again can called by browser.
  - Functions: Pure python code that can not be called by browser but can be used by views and masks.

7 URL Mapping

- CherryPy takes requests from the browser and translates them to method calls on instantiated CherryClasses
- If the browser sends a request that looks like ”/a/b” then CherryPy executes a.b()
- If the browser sends a request that looks like ”/a” then CherryPy executes the first of root.a() or a.index()
- If the browsers sends a request like ”/” then CherryPy executes root.index()
- So it’s really easy to make method calls to CherryClasses
- Even more exciting is passing data to the methods. Form data is automatically passed as named arguments to the method. ie) ”/a/b?p1=abc&p2=def invokes a.b(p1=’abc’,p2=’def’).
- This means we layout our application into logical sections (CherryClasses) and add methods to that class that behave just like normal python methods. No complicated request.getField() syntax or anything like that.

8 ”Hello World!”

- The default CherryClass should be called ”Root”. Within a CherryClass the default method is called ”index”.
- The Root/index method is called it the user doesn’t specify a class/method to invoke.

`Hello.cpy`

```
CherryClass Root:
mask:
    def index(self):
        <h1>Hello World</h1>
```

9 ”Compiling and Testing”

- To try out our first application we run it through the CherryPy compiler which produces the standalone Python script:
  `$ cherrypy Hello.cpy`

- This creates a python script called `HelloServer.py` which contains the webserver and all your content. Simply ship this script to your server machine and run it.
- By default CherryPy apps bind to port 8000. You can override this by creating a config file called ”HelloServer.cfg”
- After we startup the server we just point our web browser at ”http://127.0.0.1:8000/” and voila!
- We can even take our script and run it through something like py2exe to convert it to a standalone executable for Windows users.

10 The HTTP Server

CherryPy ”executables” include a fairly complete HTTP server that can be run in several ways:

- single threaded - Server consists of only one process which handles all requests
  - Very fast. Minimal overhead per request.
  - Not appropriate for use on it’s own since only one request is processed at a time. A slow web client holds up everyone.
  - Usually run behind Apache using mod_proxy or mod_rewrite (easy to do)
  - Can easily run on multiple machines and load balance between them to scale up
- forking - starts a fixed number of forked servers
  - Useful for multiprocessor servers.
  - Should still be run behind Apache. Since we just need one slow request per forked copy still holds up the whole system
- multithreaded - each request is processed in separate thread
  - A bit more overhead per request runs well standalone.
  - One/Many slow client(s) will not hold up others

11 ”Hello World!” v2

- Python maps URLs to method calls: ie)
  http://server:8000/ invokes Root.index
  http://server:8000/hello invokes Root.hello or ...
  http://server:8000/root/hello invokes Root.hello

- Query parameters are automatically parsed and pased as named parameters to the method call. ie)
  http://server:8000/hello?name=Jeff invokes Root.hello(name=’jeff’)

- This example collects the users name and then uses the HTML templating to insert that into a webpage.

Hello2.cpy

```
CherryClass Root:
mask:
    def index(self):
        <h1>Hello World</h1>
        <form action="/hello">
          Name: <input type="text" name="name">
        </form>
    def hello(self, name):
        <h1>Hello <py-eval="name"></h1>
```

12 Example: Guest Book

- We are going to make three CherryClasses:
  1. MyPage: Base class for all our pages. Includes "header" and "footer" methods to add common look and feel for us.
  2. Root: Has our main homepage which links to our guest book
  3. Guestbook: Contains all the methods related to the guestbook including "view" and "add"
- We have a prebuilt normal Python class for handling the guest book (Our business object) – To keep the code simple the guest book is stored in memory only.
- So we'll have four files: guestbook.py, Root.cpy, Guestbook.cpy and MyPage.cpy
- Oh... And maybe RootServer.cfg if we want to enable SSL or change the server port

13 Example: "guestbook.py"

guestbook.py

```
import time
class Guestbook:
    def __init__(self):
        self._list = []
    def add(self, name, comment):
        """ Add given name and comment to the guestbook """
        newRecord = {} # new guestbook entry - it's a dictionary
        newRecord['name'] = name
        newRecord['comment'] = comment
        newRecord['date'] = time.ctime()
        # add new record onto tail of guestbook
        self._list.append(newRecord)
    def list(self):
        """ Returns array containing all guest book entries """
        return self._list
```

- This is just a normal python class I'm going to use to build my web app

14 Example: "MyPage.cpy"

MyPage.cpy

```
CherryClass MyPage abstract:  # abstract class isn't instantiated
function:
    def redirect(self, url):
        # handly little function to send a redirect
        response.headerMap['status']=302
        response.headerMap['location']=url
        return 'Moved <a href="%s">here</a>' % url
mask:
    def header(self, title):
        <html>
        <head><title><py-eval="title"></title></head>
        <body bgcolor="lightblue">
        <h2><py-eval="title"></h2>
        <hr>
    def footer(self):
        <hr>
        &copy; 2003 Someone or Something.
        </body>
        </html>
```

15 Example: "Root.cpy"

Root.cpy

```
use MyPage

CherryClass Root(MyPage):
mask:
    def index(self):
        <py-eval="self.header('Homepage')">
        <p>Welcome to my homepage
        <p>Click <a href="/guestbook/index"> here to view
            my guestbook
        <py-eval="self.footer()">
```

- Remember: the Root class is the default class called if the browser doesn't specificially name one.

16 Example: ”Guestbook.cpy (1/2)”

Guestbook.cpy

```
use MyPage

import guestbook
import cgi

CherryClass Guestbook(MyPage):
variable:
    data = guestbook.Guestbook()   # new instance of guestbook
function:
    def viewGuestbook(self):
        out = ’’
        for record in self.data.list():
            out += "<B>%s - %s</b><br>%s<br><br>" % (
                cgi.escape(record[’name’]),     # escape any HTML
                cgi.escape(record[’date’]),
                cgi.escape(record[’comment’]))
        return out
```

17 Example: ”Guestbook.cpy (2/2)”

Guestbook.cpy

```
view:
    def addFormSubmit(self, name, comment):
        self.data.add(name, comment)
        return self.redirect(’index’)
mask:
    def index(self):
        <py-eval="self.header(’View Guestbook’)">
        <py-eval="self.viewGuestbook()">
        <center><a href="addForm">Add New Record</a></center>
        <py-eval="self.footer()">
    def addForm(self):
        <py-eval="self.header(’Add to Guestbook’)">
        <form action="addFormSubmit" method="post">
        Name: <input type="text" name="name"><br>
        Comment:
         <textarea rows=5 cols=40 name="comment"></textarea><br>
        <input type="submit">
        </form>
        <py-eval="self.footer()">
```

18 Example: Compiling and Running

```
jsc@mico:~/sample$ cherrypy Root.cpy Guestbook.cpy
jsc@mico:~/sample$ python RootServer.py
Reading parameters from RootServer.cfg ...
Server parameters:
  logToScreen: 1
  logFile:
  socketPort: 0
  socketFile:
  fixedNumberOfProcesses: 1
  threading: 0
  forking: 0
  sslKeyFile:
  sslCertificateFile:
  xmlRpc: 0
  flushCacheDelay: 0 min
  staticContent: []
Calling initServer() ...
Serving HTTP on socket port: 8000
```

- Now point your web browser to ”http://127.0.0.1:8000” and voila!

19 Why CherryPy? (1/2)

- It's Open Source (GPL'd) – The GPL is only applied to the Cherry compiler. The output of the compiler, your website, is your property.
- Python based – Python is a very good tool for rapidly developing software and it shows here. We have access to all the standard python libraries.
- Again... Fantastic documentation.
- Minimal Dependencies / Simple Installation – Server only requires Python to run. Plugins into Apache using standard modules.
- Cross platform – Runs on any platform that supports Python (Win, Lin, Max, Jython (JAVA)).
- Applications can run behind other webservers or standalone.
- Simple – Very logical and pythonic. Easy to get started.

20 Why CherryPy? (2/2)

- Gives an easy way for hobby users to setup a website/app without the effort of installing Apache, PHP, ...

- Great way to build web apps for a third party. Installation is always easy and very few dependencies.
- Easy debugging: Errors just throw standard Python exceptions with traceback.
- Very fast, stable, and scales well. `http://www.waypath.com/` runs CP with about 10k hits/day with no problems.
- The underlying Python HTTPd is quite stable and has a good security track record.

21 References

- `http://cherrpy.org`
  CherryPy homepage
- `http://tinyurl.com/fc6w`
  Introduction to CherryPy
- `http://www.freecherrypy.org/`
  Free CherryPy application hosting for personal use
- `http://www.python-hosting.com/`
  Commercial hosting
- `http://jclement.ca/clug/cherrypy-clug`
  This presentation in PDF/TEX formats as well as sample code
- `http://python.org`
  Homepage of the Python programming language
- `http://jclement.ca/software/jcard2/`
  JCard2 is a little web contact manager I wrote in CherryPy. Might be handy for a bigger example.