Programowanie i projektowanie obiektowe

CherryPy, Genshi

Paweł Daniluk

Wydział Fizyki

Jesień 2016

Aplikacje webowe

Podejście standardowe

- Serwer (np. Apache)
- Pliki HTML, CSS i grafiki w odpowiednich katalogach
- Skrypty PHP

Pewna niedogodność

Trudno jest zrealizować przyjazne dla użytkownika adresy URL, np:
`http://mojastrona.pl/blog/2013/05/11`
Zamiast tego może być:
`http://mojastrona.pl/blog.php?year=2013&month=05&day=11`
Można stosować `mod_rewrite`.

Aplikacje webowe

Podejście standardowe

- Serwer (np. Apache)
- Pliki HTML, CSS i grafiki w odpowiednich katalogach
- Skrypty PHP

Pewna niedogodność

Trudno jest zrealizować przyjazne dla użytkownika adresy URL, np:
`http://mojastrona.pl/blog/2013/05/11`
Zamiast tego może być:
`http://mojastrona.pl/blog.php?year=2013&month=05&day=11`
Można stosować `mod_rewrite`.

Struktura plików/skryptów zazwyczaj nie odpowiada logicznej strukturze serwisu.

CherryPy

- Katalogi w strukturze plików serwisu odpowiadają klasom
- Elementy katalogów (pliki i podkatalogi) odpowiadają atrybutom
- Aplikacja jest autonomicznym serwerem webowym

CherryPy

- Katalogi w strukturze plików serwisu odpowiadają klasom
- Elementy katalogów (pliki i podkatalogi) odpowiadają atrybutom
- Aplikacja jest autonomicznym serwerem webowym

- Treść nie jest przechowywana w plikach, ale generowana na bieżąco.
- Zazwyczaj konieczne jest połączenie z bazą danych (np. SQLAlchemy).
- HTML wygodnie jest tworzyć przy pomocy tzw. *template processor* (np. Genshi)

CherryPy

- Katalogi w strukturze plików serwisu odpowiadają klasom
- Elementy katalogów (pliki i podkatalogi) odpowiadają atrybutom
- Aplikacja jest autonomicznym serwerem webowym

- Treść nie jest przechowywana w plikach, ale generowana na bieżąco.
- Zazwyczaj konieczne jest połączenie z bazą danych (np. SQLAlchemy).
- HTML wygodnie jest tworzyć przy pomocy tzw. *template processor* (np. Genshi)

- Możliwe jest serwowanie plików statycznych
- Współpraca z serwerem Apache (na kilka sposobów)
- SSL

Hello World

hello.py

```
import cherrypy

class HelloWorld(object):
    def index(self):
        return "Hello World!"

    index.exposed = True

cherrypy.quickstart(HelloWorld())
```

Dostęp do żądania i odpowiedzi HTTP

Żądanie (ang. *request*) HTTP składa się z opisu żądania (GET lub POST, ścieżka), nagłówków i treści.

Odpowiedź (ang. *response*) HTTP składa się z kodu statusu, nagłówków i treści.

```
import cherrypy

class HelloWorld(object):
    def index(self):
        res=""
        for k,v in cherrypy.request.headers.iteritems():
            res+="%s: %s\n" % (k,v)
        cherrypy.response.headers['Content-Type'] = 'text/plain'
        return res

    index.exposed = True

cherrypy.quickstart(HelloWorld())
```

Eksponowanie

Jedynie metody z atrybutem exposed są widoczne.

Przez atrybut

```
class Root(object):
    def index(self):
        """Handle the / URI"""
    index.exposed = True
```

Przez dekorator

```
class Root(object):
    @cherrypy.expose
    def index(self):
        """Handle the / URI"""
```

Atrybut klasy, jeżeli jest metoda __call__

```
class Node(object):
    exposed = True
    def __call__(self):
        """ """
```

Struktura serwisu

```
"""This example can handle the URIs:
/       -> Root.index
/page   -> Root.page
/node   -> Node.__call__
"""
import cherrypy


class Node(object):
    exposed = True
    def __call__(self):
        return "The node content"


class Root(object):
    def __init__(self):
        self.node = Node()

    @cherrypy.expose
    def index(self):
        return "The index of the root object"

    def page(self):
        return 'This is the "page" content'
    page.exposed = True


if __name__ == '__main__':
    cherrypy.quickstart(Root())
```

Argumenty eksponowanych metod

Nazwane (nie ma rozróżnienia między GET a POST)

```
class Root(object):
    def doLogin(self, username=None, password=None):
        """Check the username & password"""
    doLogin.exposed = True
```

```
http://localhost/doLogin?username=user&password=pass
```

Pozycyjne

```
class Root(object):
    def blog(self, year, month, day):
        """Deliver the blog post. According to *year* *month* *d
    blog.exposed = True
```

```
http://localhost/blog/2005/01/17
```

Argumenty eksponowanych metod

Argumenty pozycyjne mogą być dopasowane do różnych metod. CherryPy znajduje "najlepiej pasującą".

Przykład

`/branch/leaf/4`:

- `app.root.branch.leaf('4')`
- `app.root.branch('leaf', '4')`
- `app.root.index('branch','leaf', '4')`

Mnóstwo innych funkcji

- obsługa sesji
- wtyczki i narzędzia
- SSL
- logi
- cache

Łatwe generowanie HTML

Podejście tradycyjne

- Skrypt PHP
- generowanie wyjścia wymieszane z obliczeniami
- program trudny w utrzymaniu

Genshi

- HTML generowany na podstawie przekazanych obiektów Pythonowych
- rozszerzenia składni XHTML o dodatkowe atrybuty
- możliwość osadzania kodu Pythonowego

Przykład

```
<?python
   title = "A␣Genshi␣Template"
   fruits = ["apple", "orange", "kiwi"]
?>
<html xmlns:py="http://genshi.edgewall.org/">
  <head>
    <title py:content="title">This is replaced.</title>
  </head>
  <body>
    <p>These are some of my favorite fruits:</p>
    <ul>
      <li py:for="fruit␣in␣fruits">
        I like ${fruit}s
      </li>
    </ul>
  </body>
</html>
```

Przykład

Wynik

```
<html>
  <head>
    <title>A Genshi Template</title>
  </head>
  <body>
    <p>These are some of my favorite fruits:</p>
    <ul>
      <li>I like apples</li>
      <li>I like oranges</li>
      <li>I like kiwis</li>
    </ul>
  </body>
</html>
```

Z poziomu Pythona

```
>>> from genshi.template import MarkupTemplate
>>> tmpl = MarkupTemplate('<h1>Hello, $name!</h1>')
>>> stream = tmpl.generate(name='world')
>>> print(stream.render('xhtml'))
<h1>Hello, world!</h1>
```

TemplateLoader

```
from genshi.template import TemplateLoader
loader = TemplateLoader([templates_dir1, templates_dir2])
tmpl = loader.load('test.html')
stream = tmpl.generate(title='Hello, world!')
print(stream.render())
```

Dostęp do argumentów

```
>>> from genshi.template import MarkupTemplate
>>> tmpl = MarkupTemplate('<em>$items[0].capitalize() item</em>')
>>> print(tmpl.generate(items=['first', 'second']))
<em>First item</em>
```

```
>>> from genshi.template import MarkupTemplate
>>> tmpl = MarkupTemplate('<em>$dict.foo</em>')
>>> print(tmpl.generate(dict='foo': 'bar'))
<em>bar</em>
```

Dostęp do elementów słownika i atrybutów obiektów jest możliwy w dwóch notacjach: z kropką (`obj.attr`) i nawiasami kwadratowymi (`obj[attr]`).

Dyrektywy

if

```
<div py:if="foo">
  <p>Bar</p>
</div>

<py:if test="foo">
  <div>
    <p>Bar</p>
  </div>
</py:if>
```

Dyrektywy c.d.

choose

```
<div py:choose="">
  <span py:when="0 == 1">0</span>
  <span py:when="1 == 1">1</span>
  <span py:otherwise="">2</span>
</div>

<div py:choose="1">
  <span py:when="0">0</span>
  <span py:when="1">1</span>
  <span py:otherwise="">2</span>
</div>

<py:choose test="1">
  <py:when test="0">0</py:when>
  <py:when test="1">1</py:when>
  <py:otherwise>2</py:otherwise>
</py:choose>
```

Dyrektywy c.d.

for

```
<ul>
  <li  py:for="item␣in␣items">${item}</li>
</ul>
<ul>

<py:for  each="item␣in␣items">
    <li>${item}</li>
  </py:for>
</ul>
```

Dyrektywy c.d.

def

```
<div>
  <p py:def="greeting(name)" class="greeting">
    Hello, ${name}!
  </p>
  ${greeting('world')}
  ${greeting('everyone else')}
</div>

<div>
  <py:def function="greeting(name)">
    <p class="greeting">Hello, ${name}!</p>
  </py:def>
</div>
```

Przykład – Geddit?

http://genshi.edgewall.org/wiki/GenshiTutorial