Python. Лекция 8. Разработка WEB-приложений.

Под **web-приложением** будет пониматься программа, основной интерфейс пользователя которой работает в стандартном WWW-браузере под управлением HTML и XML-документов. Для улучшения качества интерфейса пользователя часто применяют JavaScript, однако это несколько снижает универсальность интерфейса. Следует заметить, что интерфейс можно построить на Java- или Flash-апплетах, однако, такие приложения сложно назвать web-приложениями, так как Java или Flash могут использовать собственные протоколы для общения с сервером, а не стандартный для WWW протокол HTTP.

При создании web-приложений стараются отделить Форму (внешний вид, стиль), Содержание и Логику обработки данных. Современные технологии построения web-сайтов дают возможность подойти достаточно близко к этому идеалу. Тем не менее, даже без применения многоуровневых приложений можно придерживаться стиля, позволяющего изменять любой из этих аспектов, не затрагивая (или почти не затрагивая) двух других. Рассуждения на эту тему будут продолжены в разделе, посвященном средам разработки.

CGI-сценарии

Классический путь создания приложений для WWW - написание CGI-сценариев (иногда говорят - скриптов). CGI (Common Gateway Interface, общий шлюзовой интерфейс) - это стандарт, регламентирующий взаимодействие сервера с внешними приложениями. В случае с WWW, web-сервер может направить запрос на генерацию страницы по определенному сценарию. Этот сценарий, получив на вход данные от web-сервера (тот, в свою очередь, мог получить их от пользователя), генерирует готовый объект (изображение, аудиоданные, таблицу стилей и т.п.).

При вызове сценария Web-сервер передает ему информацию через стандартный ввод, переменные окружения и, для ISINDEX, через аргументы командной строки (они доступны через `sys.argv`).

Два основных метода передачи данных из заполненной в браузере формы Web-серверу (и CGI-сценарию) - `GET` и `POST`. В зависимости от метода данные передаются по-разному. В первом случае они кодируются и помещаются прямо в URL, например: `http://host/cgi-bin/a.cgi?a=1&b=3`. Сценарий получает их в переменной окружения с именем `QUERY_STRING`. В случае метода `POST` они передаются на стандартный ввод.

Для корректной работы сценарии помещаются в предназначенный для этого каталог на web-сервере (обычно он называется `cgi-bin`) или, если это разрешено конфигурацией сервера, в любом месте среди документов HTML. Сценарий должен иметь признак исполняемости. В системе Unix его можно установить с помощью команды `chmod a+x`.

Следующий простейший сценарий выводит значения из словаря `os.environ` и позволяет увидеть, что же было ему передано:

```
#!/usr/bin/python

import os
```

```
print """Content-Type: text/plain

%s""" % os.environ
```

С помощью него можно увидеть установленные Web-сервером переменные окружения. Выдаваемый CGI-сценарием web-серверу файл содержит заголовочную часть, в которой указаны поля с мета-информацией (тип содержимого, время последнего обновления документа, кодировка и т.п.).

Основные переменные окружения, достаточные для создания сценариев:

`QUERY_STRING`
Строка запроса.

`REMOTE_ADDR`
IP-адрес клиента.

`REMOTE_USER`
Имя клиента (если он был идентифицирован).

`SCRIPT_NAME`
Имя сценария.

`SCRIPT_FILENAME`
Имя файла со сценарием.

`SERVER_NAME`
Имя сервера.

`HTTP_USER_AGENT`
Название браузера клиента.

`REQUEST_URI`
Строка запроса (URI).

`HTTP_ACCEPT_LANGUAGE`
Желательный язык документа.

Вот что может содержать словарь `os.environ` в CGI-сценарии:

```
{
'DOCUMENT_ROOT': '/var/www/html',
'SERVER_ADDR': '127.0.0.1',
'SERVER_PORT': '80',
'GATEWAY_INTERFACE': 'CGI/1.1',
'HTTP_ACCEPT_LANGUAGE': 'en-us, en;q=0.50',
'REMOTE_ADDR': '127.0.0.1',
'SERVER_NAME': 'rnd.onego.ru',
'HTTP_CONNECTION': 'close',
'HTTP_USER_AGENT': 'Mozilla/5.0 (X11; U; Linux i586; en-US;
rv:1.0.1) Gecko/20021003',
'HTTP_ACCEPT_CHARSET': 'ISO-8859-1, utf-8;q=0.66, *;q=0.66',
'HTTP_ACCEPT': 'text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,
image/gif;q=0.2,text/css,*/*;q=0.1',
'REQUEST_URI': '/cgi-bin/test.py?a=1',
'PATH': '/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin',
'QUERY_STRING': 'a=1&b=2',
'SCRIPT_FILENAME': '/var/www/cgi-bin/test.py',
'HTTP_KEEP_ALIVE': '300',
```

```
'HTTP_HOST': 'localhost',
'REQUEST_METHOD': 'GET',
'SERVER_SIGNATURE': 'Apache/1.3.23 Server at rnd.onego.ru Port 80',
'SCRIPT_NAME': '/cgi-bin/test.py',
'SERVER_ADMIN': 'root@localhost',
'SERVER_SOFTWARE': 'Apache/1.3.23 (Unix)  (Red-Hat/Linux)
mod_python/2.7.8 Python/1.5.2 PHP/4.1.2',
'SERVER_PROTOCOL': 'HTTP/1.0',
'REMOTE_PORT': '39251'
}
```

Следующий CGI-сценарий выдает черный квадрат (в нем используется модуль `Image` для обработки изображений):

```
#!/usr/bin/python

import sys
print """Content-Type: image/jpeg
"""

import Image
i = Image.new("RGB", (10,10))
i.im.draw_rectangle((0,0,10,10), 1)
i.save(sys.stdout, "jpeg")
```

Модуль cgi

В Python имеется поддержка CGI в виде модуля `cgi`. Следующий пример показывает некоторые из его возможностей:

```
#!/usr/bin/python
# -*- coding: cp1251 -*-
import cgi, os

# анализ запроса
f = cgi.FieldStorage()
if f.has_key("a"):
  a = f["a"].value
else:
  a = "0"

# обработка запроса
b = str(int(a)+1)
mytext = open(os.environ["SCRIPT_FILENAME"]).read()
mytext_html = cgi.escape(mytext)

# формирование ответа
print """Content-Type: text/html

<html><head><title>Решение примера: %(b)s = %(a)s + 1</title></head>
<body>
%(b)s
<table width="80%%"><tr><td>
<form action="me.cgi" method="GET">
<input type="text" name="a" value="0" size="6">
<input type="submit" name="b" value="Обработать">
</form></td></tr></table>
<pre>
%(mytext_html)s
</pre>
</body></html>""" % vars()
```

В этом примере к заданному в форме числу прибавляется 1. Кроме того, выводится исходный код самого сценария. Следует заметить, что для экранирования символов `>`, `<`, `&` использована функция `cgi.escape()`. Для формирования Web-страницы применена операция форматирования. В качестве словаря для выполнения подстановок использован словарь `vars()` со всеми локальными переменными. Знаки процента пришлось удвоить, чтобы они не интерпретировались командой форматирования. Стоит обратить внимание на то, как получено значение от пользователя. Объект `FieldStorage` "почти" словарь, с тем исключением, что для получения обычного значения нужно дополнительно посмотреть атрибут `value`. Дело в том, что в сценарий могут передаваться не только текстовые значения, но и файлы, а также множественные значения с одним и тем же именем.

Осторожно!

При обработке входных значений CGI-сценариев нужно внимательно и скрупулезно проверять допустимые значения. Лучше считать, что клиент может передать на вход все, что угодно. Из этого всего необходимо выбрать и проверить только то, что ожидает сценарий.

Например, не следует подставлять полученные от пользователя данные в путь к файлу, в качестве аргументов к функции `eval()` и ей подобных; параметров командной строки; частей в SQL-запросах к базе данных. Также не стоит вставлять полученные данные напрямую в формируемые страницы, если эти страницы будет видеть не только клиент, заказавший URL (например, такая ситуация обычна в web-чатах, форумах, гостевых книгах), и даже в том случае, если единственный читатель этой информации - администратор сайта. Тот, кто смотрит страницы с непроверенным HTML-кодом, поступившим напрямую от пользователя, рискуют обработать в своем браузере зловредный код, использующий брешь в его защите.

Даже если CGI-сценарий используется исключительно другими сценариями через запрос на URL, нужно проверять входные значения столь же тщательно, как если бы данные вводил пользователь. (Так как недоброжелатель может подать на web-сервер любые значения).

В примере выше проверка на допустимость произведена при вызове функции `int()`: если было бы задано нечисловое значение, сценарий аварийно завершился, а пользователь увидел `Internal Server Error`.

После анализа входных данных можно выделить фазу их обработки. В этой части CGI-сценария вычисляются переменные для дальнейшего вывода. Здесь необходимо учитывать не только значения переданных переменных, но и факт их присутствия или отсутствия, так как это тоже может влиять на логику сценария.

И, наконец, фаза вывода готового объекта (текста, HTML-документа, изображения, мультимедиа-объекта и т.п.). Проще всего заранее подготовить шаблон страницы (или ее крупных частей), а потом просто заполнить содержимым из переменных.

В приведенных примерах имена появлялись в строке запроса только один раз. Некоторые формы порождают несколько значений для одного имени. Получить все значения можно с помощью метода `getlist()`:

```
lst = form.getlist("fld")
```

Список `lst` будет содержать столько значений, сколько полей с именем `fld` получено из web-формы (он может быть и пустым, если ни одно поле с заданным именем не было заполнено).

В некоторых случаях необходимо передать на сервер файлы (сделать upload). Следующий пример и комментарий к нему помогут разобраться с этой задачей:

```
#!/usr/bin/env python
import cgi

form = cgi.FieldStorage()
file_contents = ""
if form.has_key("filename"):
  fileitem = form["filename"]
  if fileitem.file:
    file_contents = """<P>Содержимое переданного файла:
    <PRE>%s</PRE>""" % fileitem.file.read()

print """Content-Type: text/html

<HTML><HEAD><TITLE>Загрузка файла</TITLE></HEAD>
<BODY><H1>Загрузка файла</H1>
<P><FORM ENCTYPE="multipart/form-data"
ACTION="getfile.cgi" METHOD="POST">
<br>Файл: <INPUT TYPE="file" NAME="filename">
<br><INPUT TYPE="submit" NAME="button" VALUE="Передать файл">
</FORM>
%s
</BODY></HTML>""" % file_contents
```

В начале следует рассмотреть web-форму, которая приведена в конце сценария: именно она будет выводиться пользователю при обращении по CGI-сценарию. Форма имеет поле типа `file`, которое в web-браузере представляется полоской ввода и кнопкой "Browse". Нажимая на кнопку "Browse", пользователь выбирает файл, доступный в ОС на его компьютере. После этого он может нажать кнопку "Передать файл" для передачи файла на сервер.

Для отладки CGI-сценария можно использовать модуль `cgitb`. При возникновении ошибки этот модуль выдаст красочную HTML-страницу с указанием места возбуждения исключения. В начале отлаживаемого сценария нужно поставить

```
import cgitb
cgitb.enable(1)
```

Или, если не нужно показывать ошибки в браузере:

```
import cgitb
  cgitb.enable(0, logdir="/tmp")
```

Только необходимо помнить, что следует убрать эти строки, когда сценарий будет отлажен, так как он выдает кусочки кода сценария. Это может быть использовано злоумышленниками, с тем чтобы найти уязвимости в CGI-сценарии или подсмотреть пароли (если таковые присутствуют в сценарии).

Что после CGI?

К сожалению, строительство интерактивного и посещаемого сайта на основе CGI имеет свои ограничения, главным образом, связанные с производительностью. Ведь для каждого запроса нужно вызвать как минимум один сценарий (а значит - запустить интерпретатор Python), из него, возможно, сделать соединение с базой данных и т.д. Время запуска интерпретатора Python достаточно невелико, тем не менее, на занятом сервере оно может оказывать сильное влияние на загрузку процессора.

Желательно, чтобы интерпретатор уже находился в оперативной памяти, и были доступны соединения с базой данных.

Такие технологии существуют и обычно опираются на модули, встраиваемые в web-сервер.

Для ускорения работы CGI используются различные схемы, например, FastCGI или PCGI (Persistent CGI). В данной лекции предлагается к рассмотрению специальный модуль для web-сервера Apache, называемый `mod_python`.

Пусть модуль установлен на web-сервере в соответствии с инструкциями, данными в его документации.

Модуль `mod_python` позволяет сценарию-обработчику вклиниваться в процесс обработки HTTP-запроса сервером Apache на любом этапе, для чего сценарий должен иметь определенным образом названные функции.

Сначала нужно выделить каталог, в котором будет работать сценарий-обработчик. Пусть это каталог `/var/www/html/mywebdir`. Для того чтобы web-сервер знал, что в этом каталоге необходимо применять `mod_python`, следует добавить в файл конфигурации Apache следующие строки:

```
<Directory "/var/www/html/mywebdir">
  AddHandler python-program .py
  PythonHandler mprocess
</Directory>
```

После этого необходимо перезапустить web-сервер и, если все прошло без ошибок, можно приступать к написанию обработчика `mprocess.py`. Этот сценарий будет реагировать на любой запрос вида http://localhost/*.py.

Следующий сценарий `mprocess.py` выведет в браузере страницу со словами `Hello, world`!:

```
from mod_python import apache

def handler(req):
  req.content_type = "text/html"
  req.send_http_header()
  req.write("""<HTML><HEAD><TITLE>Hello, world!</TITLE></HEAD>
  <BODY>Hello, world!</BODY></HTML>""")
  return apache.OK
```

Отличия сценария-обработчика от CGI-сценария:

1. Сценарий-обработчик не запускается при каждом HTTP-запросе: он уже находится в памяти, и из него вызываются необходимые функции-обработчики (в приведенном примере такая функция всего одна - `handler()`). Каждый процесс-потомок web-сервера может иметь свою копию сценария и интерпретатора Python.
2. Как следствие п.1 различные HTTP-запросы делят одни и те же глобальные переменные. Например, таким образом можно инициализировать соединение с базой данных и применять его во всех запросах (хотя в некоторых случаях потребуются блокировки, исключающие одновременное использование соединения разными потоками (нитями) управления).
3. Обработчик задействуется при обращении к любому "файлу" с расширением `py`, тогда как CGI-сценарий обычно запускается при обращении по конкретному имени.
4. В сценарии-обработчике нельзя рассчитывать на то, что он увидит модули, расположенные в том же каталоге. Возможно, придется добавить некоторые каталоги в `sys.path`.

5. Текущий рабочий каталог (его можно узнать с помощью функции `os.getcwd()`) также не находится в одном каталоге с обработчиком.

6. #!-строка в первой строке сценария не определяет версию интерпретатора Python. Работает версия, для которой был скомпилирован `mod_python`.

7. Все необходимые параметры передаются в обработчик в виде Request-объекта. Возвращаемые значения также передаются через этот объект.

8. Web-сервер замечает, что сценарий-обработчик изменился, но не заметит изменений в импортируемых в него модулях. Команда `touch mprocess.py` обновит дату изменения файла сценария.

9. Отображение `os.environ` в обработчике может быть обрезанным. Кроме того, вызываемые из сценария-обработчика другие программы его не наследуют, как это происходит при работе с CGI-сценариями. Переменные можно получить другим путем: `req.add_common_vars(); params = req.subprocess_env`.

10. Так как сценарий-обработчик не является "одноразовым", как CGI-сценарий, из-за ошибок программирования (как самого сценария, так и других компонентов) могут возникать утечки памяти (программа не освобождает ставшую ненужной память). Следует установить значение параметра `MaxRequestsPerChild` (максимальное число запросов, обрабатываемое одним процессом-потомком) больше нуля.

Другой возможный обработчик - сценарий идентификации:

```
def authenhandler(req):
  password = req.get_basic_auth_pw()
  user = req.connection.user
  if user == "user1" and password == "secret":
      return apache.OK
  else:
      return apache.HTTP_UNAUTHORIZED
```

Эту функцию следует добавить в модуль `mprocess.py`, который был рассмотрен ранее. Кроме того, нужно дополнить конфигурацию, назначив обработчик для запросов идентификации (`PythonAuthenHandler`), а также обычные для Apache директивы `AuthType`, `AuthName`, `require`, определяющие способ авторизации:

```
<Directory "/var/www/html/mywebdir">
  AddHandler python-program .py
  PythonHandler mprocess
  PythonAuthenHandler mprocess
  AuthType Basic
  AuthName "My page"
  require valid-user
</Directory>
```

Разумеется, это - всего лишь пример. В реальности идентификация может быть устроена намного сложнее.

Другие возможные обработчики (по документации к `mod_python` можно уточнить, в какие моменты обработки запроса они вызываются):

`PythonPostReadRequestHandler`

Обработка полученного запроса сразу после его получения.

`PythonTransHandler`

Позволяет изменить URI запроса (в том числе имя виртуального сервера).

`PythonHeaderParserHandler`

Обработка полей запроса.

`PythonAccessHandler`

Обработка ограничений доступа (например, по IP-адресу).

`PythonAuthenHandler`

Идентификация пользователя.

`PythonTypeHandler`

Определение и/или настройка типа документа, языка и т.д.

`PythonFixupHandler`

Изменение полей непосредственно перед вызовом обработчиков содержимого.

`PythonHandler`

Основной обработчик запроса.

`PythonInitHandler`

`PythonPostReadRequestHandler` или `PythonHeaderParserHandler` в зависимости от нахождения в конфигурации web-сервера.

`PythonLogHandler`

Управление записью в логи.

`PythonCleanupHandler`

Обработчик, вызываемый непосредственно перед уничтожением Request-объекта.

Некоторые из этих обработчиков работают только глобально, так как при вызове даже каталог их приложения может быть неизвестен (таков, например, `PythonPostReadRequestHandler`).

С помощью `mod_python` можно строить web-сайты с динамическим содержимым и контролировать некоторые аспекты работы web-сервера Apache через Python-сценарии.

Среды разработки

Для создания Web-приложений применяются и более сложные средства, чем web-сервер с расположенными на нем статическими документами и CGI-сценариями. В зависимости от назначения такие программные системы называются серверами web-приложений, системами управления содержимым (CMS, Content Management System), системы web-публикации и средствами для создания WWW-порталов. Причем CMS-система может быть выполнена как web-приложение, а средства для создания порталов могут базироваться на системах web-публикации, для которых CMS-система является подсистемой. Поэтому, выбирая систему для конкретных нужд, стоит уточнить, какие функции она должна выполнять.

Язык Python, хотя и уступает PHP по количеству созданных на нем web-систем, имеет несколько достаточно популярных приложений. Самым ярким примером средства

для создания сервера web-приложений является Zope (произносится "зоп") (см. http://zope.org) (Z Object Publishing Environment, **среда публикации объектов**). Zope имеет встроенный web-сервер, но может работать и с другими Web-серверами, например, Apache. На основе Zope можно строить web-порталы, например, с помощью Plone/Zope, но можно разрабатывать и собственные web-приложения. При этом Zope позволяет разделить Форму, Содержание и Логику до такой степени, что Содержанием могут заниматься одни люди (менеджеры по содержимому), Формой - другие (web-дизайнеры), а Логикой - третьи (программисты). В случае с Zope Логику можно задать с помощью языка Python (или, как вариант, Perl), Форма может быть создана в графических или специализированных web-редакторах, а работа с содержимым организована через Web-формы самого Zope.

Zope и его объектная модель

В рамках этой лекции невозможно детально рассмотреть такой инструмент как Zope, поэтому стоит лишь заметить, что он достаточно интересен не только в качестве среды разработки web-приложений, но и с точки зрения подходов. Например, уникальная объектно-ориентированная модель Zope позволяет довольно гибко описывать требуемое приложение.

Zope включает в себя следующие возможности:

- Web-сервер. Zope может работать с Web-серверами через CGI или использовать свой встроенный Web-сервер (ZServer).
- Среда разработчика выполнена как Web-приложение. Zope позволяет создавать Web-приложения через Web-интерфейс.
- Поддержка сценариев. Zope поддерживает несколько языков сценариев: Python, Perl и собственный **DTML** (**D** ocument **T** emplate **M** arkup **L** anguage, язык разметки шаблона документа).
- База данных объектов. Zope использует в своей работе устойчивые объекты, хранимые в специальной базе данных (ZODB). Имеется достаточно простой интерфейс для управления этой базой данных.
- Интеграция с реляционными базами данных. Zope может хранить свои объекты и другие данные в реляционных СУБД: Oracle, PostgreSQL, MySQL, Sybase и т.п.

В ряду других подобных систем Zope на первый взгляд кажется странным и неприступным, однако тем, кто с ним "на ты", он открывает большие возможности.

Разработчики Zope исходили из лежащей в основе WWW объектной модели, в которой загрузку документа по URI можно сравнить с отправкой сообщения объекту. Объекты Zope разложены по папкам (folders), к которым привязаны **политики доступа** для пользователей, имеющих определенные **роли**. В качестве объектов могут выступать документы, изображения, мультимедиа-файлы, адаптеры к базам данных и т.п.

Документы Zope можно писать на языке DTML - дополнении HTML с синтаксисом для включения значений подобно SSI (Server-Side Include). Например, для вставки переменной с названием документа можно использовать

```
<!- #var document_title ->
```

Следует заметить, что объекты Zope могут иметь свои атрибуты, а также методы, в частности, написанные на языке Python. Переменные же могут появляться как из заданных пользователем значений, так и из других источников данных (например, из базы данных посредством выполнения выборки функцией SELECT).

Сейчас для описания документа Zope все чаще применяется **ZPT** (**Z** ope **P** age **T** emplates, шаблоны страниц Zope), которые в свою очередь используют **TAL** (**T** emplate

A ttribute **L** anguage, язык шаблонных атрибутов). Он позволяет заменять, повторять или пропускать элементы документа описываемого шаблоном документа. "Операторами" языка TAL являются XML-атрибуты из пространства имен TAL. Пространство имен сегодня описывается следующим идентификатором:

```
xmlns:tal=http://xml.zope.org/namespaces/tal
```

Оператор TAL имеет имя и значение (что выражается именем и значением атрибута). Внутри значения обычно записано TAL-выражение, синтаксис которого описывается другим языком - TALES (Template Attribute Language Expression Syntax, синтаксис выражений TAL).

Таким образом, ZPT наполняет содержимым шаблоны, интерпретируя атрибуты TAL. Например, чтобы Zope подставил название документа (тег `TITLE`), шаблон может иметь следующий код:

```
<title tal:content="here/title">Doc Title</title>
```

Стоит заметить, что приведенный код сойдет за код на HTML, то есть, Web-дизайнер может на любом этапе работы над проектом редактировать шаблон в HTML-редакторе (при условии, что тот сохраняет незнакомые атрибуты из пространства имен tal). В этом примере `here/title` является выражением TALES. Текст `Doc Title` служит ориентиром для web-дизайнера и заменяется значением выражения `here/title`, то есть, будет взято свойство `title` документа Zope.

Примечание:

В Zope объекты имеют свойства. Набор свойств зависит от типа объекта, но может быть расширен в индивидуальном порядке. Свойство `id` присутствует всегда, свойство `title` обычно тоже указывается.

В качестве более сложного примера можно рассмотреть организацию повтора внутри шаблона (для опробования этого примера в Zope нужно добавить объект Page Template):

```
<ul>
   <li tal:define="s modules/string"
       tal:repeat="el python:s.digits">
   <a href="DUMMY"
      tal:attributes="href string:/digit/$el"
      tal:content="el">SELECTION</a>
  </li>
</ul>
```

Этот шаблон породит следующий результат:

```
<ul>
   <li><a href="/digit/0">0</a></li>
   <li><a href="/digit/1">1</a></li>
   <li><a href="/digit/2">2</a></li>
   <li><a href="/digit/3">3</a></li>
   <li><a href="/digit/4">4</a></li>
   <li><a href="/digit/5">5</a></li>
   <li><a href="/digit/6">6</a></li>
   <li><a href="/digit/7">7</a></li>
   <li><a href="/digit/8">8</a></li>
   <li><a href="/digit/9">9</a></li>
</ul>
```

Здесь нужно обратить внимание на два основных момента:

- в шаблоне можно использовать выражения Python (в данном примере переменная `s` определена как модуль Python) и переменную-счетчик цикла `el`, которая проходит итерации по строке `string.digits`.
- с помощью TAL можно задавать не только содержимое элемента, но и атрибута тега (в данном примере использовался атрибут `href`).

Детали можно узнать по документации. Стоит лишь заметить, что итерация может происходить по самым разным источникам данных: содержимому текущей папки, выборке из базы данных или, как в приведенном примере, по объекту Python.

Любой программист знает, что программирование тем эффективнее, чем лучше удалось "расставить скобки", выведя "общий множитель за скобки". Другими словами, хорошие программисты должны быть достаточно "ленивы", чтобы найти оптимальную декомпозицию решаемой задачи. При проектировании динамического web-сайта Zope позволяет разместить "множители" и "скобки" так, чтобы достигнуть максимального повторного использования кода (как разметки, так и сценариев). Помогает этому уникальный подход к построению взаимоотношений между объектами: заимствование (**acquisition**).

Пусть некоторый объект (документ, изображение, сценарий, подключение к базе данных и т.п.) расположен в папке Example. Теперь объекты этой папки доступны по имени из любых нижележащих папок. Даже политики безопасности заимствуются более глубоко вложенными папками от папок, которые ближе к корню. Заимствование является очень важной концепцией Zope, без понимания которой Zope сложно грамотно применять, и наоборот, ее понимание позволяет экономить силы и время, повторно используя объекты в разработке.

Самое интересное, что заимствовать объекты можно также из параллельных папок. Пусть, например, рядом с папкой Example находится папка Zigzag, в которой лежит нужный объект (его наименование note). При этом в папке Example программиста интересует объект `index_html`, внутри которого вызывается note. Обычный путь к объекту `index_html` будет происходить по URI вроде `http://zopeserver/Example/`. А вот если нужно использовать note из Zigzag (и в папке Example его нет), то URI будет: `http://zopeserver/Zigzag/Example/`. Таким образом, указание пути в Zope отличается от традиционного пути, скажем, в Unix: в пути могут присутствовать "зигзаги" через параллельные папки, дающие возможность заимствовать объекты из этих папок. Таким образом, можно сделать конкретную страницу, комбинируя один или несколько независимых аспектов.

Заключение

В этой лекции были рассмотрены различные подходы к использованию Python в web-приложениях. Самый простой способ реализации web-приложения - использование CGI-сценариев. Более сложным является использование специальных модулей для web-сервера, таких как `mod_python`. Наконец, существуют технологии вроде Zope, которые предоставляют специализированные сервисы, позволяющие создавать web-приложения.