Язык программирования

Лекция № 8

Владимир Владимирович Руцкий
rutsky.vladimir@gmail.com

План занятия

- Чат на Python (работа с сокетами)
- Основы WWW: HTML, CSS, JavaScript
- Написание простого веб-сервера
- Установка библиотек в Python
- Введение в Django

Чат на Python

Основы web-программирования

- Браузер
  - HTML (https://ru.wikipedia.org/wiki/HTML)

    Самоучитель HTML4: http://htmlbook.ru/samhtml/vvedenie-v-html

    Самоучитель HTML5: http://htmlbook.ru/samhtml5
  - CSS (https://ru.wikipedia.org/wiki/CSS)

    Самоучитель: http://htmlbook.ru/samcss
  - JavaScript (https://ru.wikipedia.org/wiki/Javascript)

HTML

- DOM-модель
- Отображение статического контента
- URL

CSS

- Гибкие возможности по управлению *стилями* отображения

JavaScript

- Обработка событий и модификация DOM

HTTP

- Протокол HTTP (https://ru.wikipedia.org/wiki/HTTP).
- Методы GET, POST, HEAD
- Формы
- Cookie

Веб-сервер на Python

```
import http.server
import socketserver

class MyHandler(http.server.BaseHTTPRequestHandler):
    def do_HEAD(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()

    def do_GET(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()

        self.wfile.write("""\
            <!DOCTYPE html>
            <html>
            <head>
              <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
              <title>Заголовок окна</title>
            </head>
            <body>
              <h1>Заголовок страницы</h1>
              <p>Параграф с <b>жирным</b> и <i>наклонным</i> текстом.</p>
            </body>
            </html>
            """.encode('utf-8'))

if __name__ == "__main__":
    server = http.server.HTTPServer(('localhost', 8080), MyHandler)
    print('Started http server')
    server.serve_forever()
```

WSGI

- В примере веб-сервер — это обёртка над сокетами, реализующая протокол HTTP
- Интерфейс HTTP достаточно простой ("запрос — ответ"), но его реализация содержит много непростых деталей
  - `http.server.HTTPServer` — упрощённая реализация, реализует не все возможности HTTP
- Чтобы связать существующие веб-сервера (Apache, Nginx, IIS) с различными веб-фреймворками на Python (Django, TurboGears) разработали протокол WSGI
- WSGI — Web Server Gateway Interface

Пример клиента WSGI

```
# Точка входа в web-приложение на Python – Python модуль
# Модуль должен создать глобальный объект `application',
# который будет вызываться при поступлении запроса

def application(environ, start_response):
    # В качестве объекта используем просто функцию

    # Вызываемый объект принимает словарь с специально сформированными
    # переменными окружения (содержит запрашиваемый путь, переданные параметры
    # в GET и POST и т.п.)

    # В соответствии с тем, какой ресурс или действие были запрошены,
    # подготавливаем ответ.

    # Выводим заголовки ответа
    start_response('200 OK', [('Content-Type', 'text/plain')])

    # Выводим ответ
    yield b'Hello World!\n'

    # Выводим доступные переменные
    import pprint
    yield pprint.pformat(environ).encode('utf-8')
```

Пример сервера WSGI

```
# В Python есть реализация WSGI-совместимого сервера
# Также можно использовать модули Apache/Nginx/IIS для запуска WSGI клиентов
from wsgiref.validate import validator
from wsgiref.simple_server import make_server

# "Запускаем" клиента
import wsgi_client

# Получаем из клиента объект application и оборачиваем его в валидатор
validator_app = validator(wsgi_client.application)

# Создаём и запускаем сервер
httpd = make_server('', 8010, validator_app)
print("Listening on port 8010...")
httpd.serve_forever()
```

HTTP **клиенты на** Python

- Библиотека https://docs.python.org/3/library/urllib.request.html

http.server

- При исполнении модуля "расшаривает" текущую директорию:

```
python -m http.server 8080
```

Назначение директорий Python

- Установленный дистрибутив Python в Windows:

```
C:\Python33\
    python.exe          - интерпретатор (main)
    pythonw.exe         - интерпретатор без консоли (WinMain)
    Tools\
        Scripts\        - Python скрипты (часто добавляется в %PATH%)
            2to3.py
            ...
    Lib\                - стандартная библиотека
        site-packages\ - сторонние библиотеки
            pygame\
            pygame-1.9.2a0-py3.3.egg-info
            ...
    ...
```

Установка библиотек в Python

- Установка через инсталлятор (Windows):

```
C:\>pygame-1.9.2a0-py3.3.msi
```

- Установка через пакетный менеджер (GNU/Linux):

```
sudo apt-get install python-pygame
```

- Установка из исходных кодов (Setuptools, Distribute — низкоуровневый Python-way):

```
# 1. скачать и распаковать исходные коды
# 2. запустить из директории с исходными кодами:
python.exe setup.py install
# А можно собрать msi из исходных кодов (или другие "пакеты")
# python.exe setup.py bdist_msi
# python.exe setup.py bdist_rpm
```

- Использовать пакетный менеджер Python (сначала нужно его поставить):

```
# pip:
C:\Python33\Tools\Scripts\pip install pygame

# setuptools:
C:\Python33\Tools\Scripts\easy_install install pygame
```

Конфликты версий библиотек

- Установка библиотек "в систему" может привести к конфликту версий
  - Программа A зависит от библиотеки C-Lib v.1.0
  - Программа B зависит от библиотеки C-Lib v.2.0
  - Программы A и B нельзя поставить одновременно
- В Windows такая проблема с динамически подгружаемыми библиотеками была названа *DLL Hell*
- Выход: ставить библиотеки не "в систему", а для каждого приложения отдельно

Virtualenv

- В Python существует проблема с конфликтами версий библиотек при установке "в систему"
- В качестве решения можно использовать virtualenv — виртуальная установка Python:

```
C:\scripts> C:\Python33\python.exe C:\Python33\Tools\Scripts\pyvenv.py env

или

C:\scripts> C:\Python33\python.exe -m venv env
```

- Директория env будет содержать виртуальную установку Python:

```
C:\scripts\env
    Include\
    Lib\
        site-packages\
            ...
        ...
    Scripts\
        python.exe
        activate.bat
        ...
    pyenv.cfg
```

Установка библиотек в Virtualenv

- В сессии shell можно "активировать" виртуальный Python:

```
C:\scripts> env\Scripts\activate.bat
(env) C:\scripts> python.exe # - будет использовать "виртуальный" python.exe
(env) C:\scripts> pip install Django # сначала нужно поставить pip, в Python 3.4 он уже
```

- Или можно запускать скрипты, используя `C:\scripts\env\Scripts\python.exe`:

```
C:\pygame-src> C:\scripts\env\Scripts\python.exe setup.py install
```

- Обычно для каждого проекта делается отдельный virtualenv с зависимостями только этого проекта

Virtualenv **в** PyCharm

- `File` -> `Settings` -> `Python Interpreter` -> `Python Interpreters` -> кнопка `Create virtual environment`
- Выберите директорию для `env`

В PyCharm можно ставить библиотеки через кнопку `Install` в том же меню

- Нажмите `Install` -> наберите в поиске `Django` -> `Install package`

Назначение веб-приложений

- Суть — передача информации
- Предоставление общей информации (сайт-визитка) — **статический контент**
  - Всем клиентом отдаётся *одна и та же* информация
- Предоставление персонализированной информации (веб-интерфейс к почте) — **динамический контент**
  - Для каждого клиента *генерируется* своя информация
- Обработка запросов клиентов (добавить фотографию, пост, и т.п.)
  - Обрабатывается запрос *с данными от клиента*

Серверов должно быть много

- Клиентов много. *Очень много*
- Один сервер может обрабатывать десятки тысяч "холостых" запросов в секунду

  Реальных запросов — на порядки меньше
- Один сервер имеет ограниченный сетевой канал
- Отклик сервера (или задержка, latency) зависит от взаимного географического положения сервера и клиента
- Один сервер может сломаться — сложно реализовать отказоустойчивость
- Вывод: одного сервера недостаточно

Масштабирование

- Масштабируемость (scalability) — способность системы, сети или процесса справляться с увеличением рабочей нагрузки при добавлении ресурсов
- **Горизонтальное** — увеличение мощности процессоров, увеличение памяти
- **Вертикальное** — увеличение количества обработчиков (CPU, компьютеров)

Статический контент

- Статический контент можно раскопировать на множество серверов по всему миру
  - при запросе клиент будет соединяться с географически ближайшим (позволяет DNS)
- Такие системы называются Content Delivery Network (CDN)
- Обычно статические картинки, CSS-файлы, JavaScript-файлы распространяются через CDN

Динамический контент

- Информация получается от пользователя, обрабатывается и отдаётся пользователю/лям (часто в модифицированном виде)
- Серверов для приёма запросов должно быть много, но хранилище информации должно быть одним
- В качестве хранилищ используют различные **базы данных** (БД)
- На серверах используются фреймворки **без состояния** (stateless), которые запрашивают информацию (состояние) из БД

Базы данных

- В промышленности сейчас используются **реляционные БД**
  - Хорошо изученная и проработанная область
  - Плохо масштабируются
  - "Табличные" данные. Сложно представлять графы и деревья с нефиксированной структурой
- В интернет-проектах всё чаще используются **распределённые** (distributed) хранилища ("облачные")
  - Не реляционные. Чаще всего с интерфейсом словаря (получение значения по ключу)

Кеширование

- Сгенерированный динамический контент чаще всего **кешируется** (сохраняется) на промежуточном сервере
  - При повторном запросе в ограниченный промежуток времени используется "закешированная" копия
- Между клиентов и сервером также обычно ставится **балансировщик нагрузки** (load balancer)
  - Перенаправляет запросы от клиенту к одному из множества серверов-обработчиков запросов

Реляционные БД

- Грубо:
  - Данные хранятся в наборе таблиц
  - У каждой таблицы фиксированное количество столбцов и тип значений в каждом столбце
  - Строка в таблице представляет собой запись

Таблица классов:

ID **класса**	**Название**	**Специализация**
1	"11-1"	"физико-математический"
2	"11-2"	"математико-физический"

Таблица учеников:

ID **ученика**	**Имя**	**Фамилия**	ID **класса**
1	"Вася"	"Петров"	1
2	"Маша"	"Иванова"	2
3	"Коля"	"Сидоров"	1

- Для работы с данными в реляционных БД используется язык SQL

Введение в Django

- Django (https://www.djangoproject.com/) — фреймворк для разработки веб-приложений на Python
- лицензия BSD
- Используется: Instagram, Disqus, Mozilla, The Washington Times, Pinterest и др.
- Изначально разрабатывался программистами газеты Lawrence Journal-World
  - требовалась быстрая и лёгкая публикация контента, быстрое добавление новой функциональности

Туториал Django

На основе туториала https://docs.djangoproject.com/en/1.6/

- Создайте новый проект в PyCharm
- Создайте virtualenv для этого проекта
- Установите Django в virtualenv
- Создайте проект Django в командной строке:

```
C:\scripts> env\Scripts\activate.bat
(env) C:\scripts> env\Scripts\django-admin.py startproject mysite
```

- Запускаем сервер разработчика:

```
(env) C:\scripts> cd mysite
(env) C:\scripts\mysite> python.exe manage.py runserver
Validating models...

0 errors found
April 26, 2014 - 08:31:31
Django version 1.6.3, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

- Сервер доступен по адресу http://127.0.0.1:8000/

Структура проекта на Django

- Приложение на Django состоит из **проекта** (project) и **приложений** (applications)
  - В одном проекте ("моя домашняя страница") может быть несколько приложений ("гостевая книга", "каталог ссылок")
  - Приложение может распространяться независимо от проекта и использоваться в нескольких проектах
- `python django-admin.py startproject mysite` создаёт проект
- `python manage.py startapp polls` создаёт приложение внутри проекта

Паттерн MVC

- В интерактивных приложениях часто используют паттерн Model-View-Controller (**Модель**-**Представление**-**Контроллер**, MVC) или его производные

  Приложение структурно и логически делится на:

  - Модель — структурированные данные приложения (например, список с объектами "пользователь")
  - Представление — графический интерфейс, отображающий данные из Модели (виджет со списком и кнопками)
  - Контроллер — обработчик ввода пользователя, по событиям изменяющий Модель

MVC **в** Django

- В Django:
  - Model — Object-relational mapper (ORM), связывает данные в БД и объекты классов в Python
  - View — система шаблонов HTML (и не только) документов
  - Контроллер — вызов методов в зависимости от запрашиваемых URL