World Wide Web

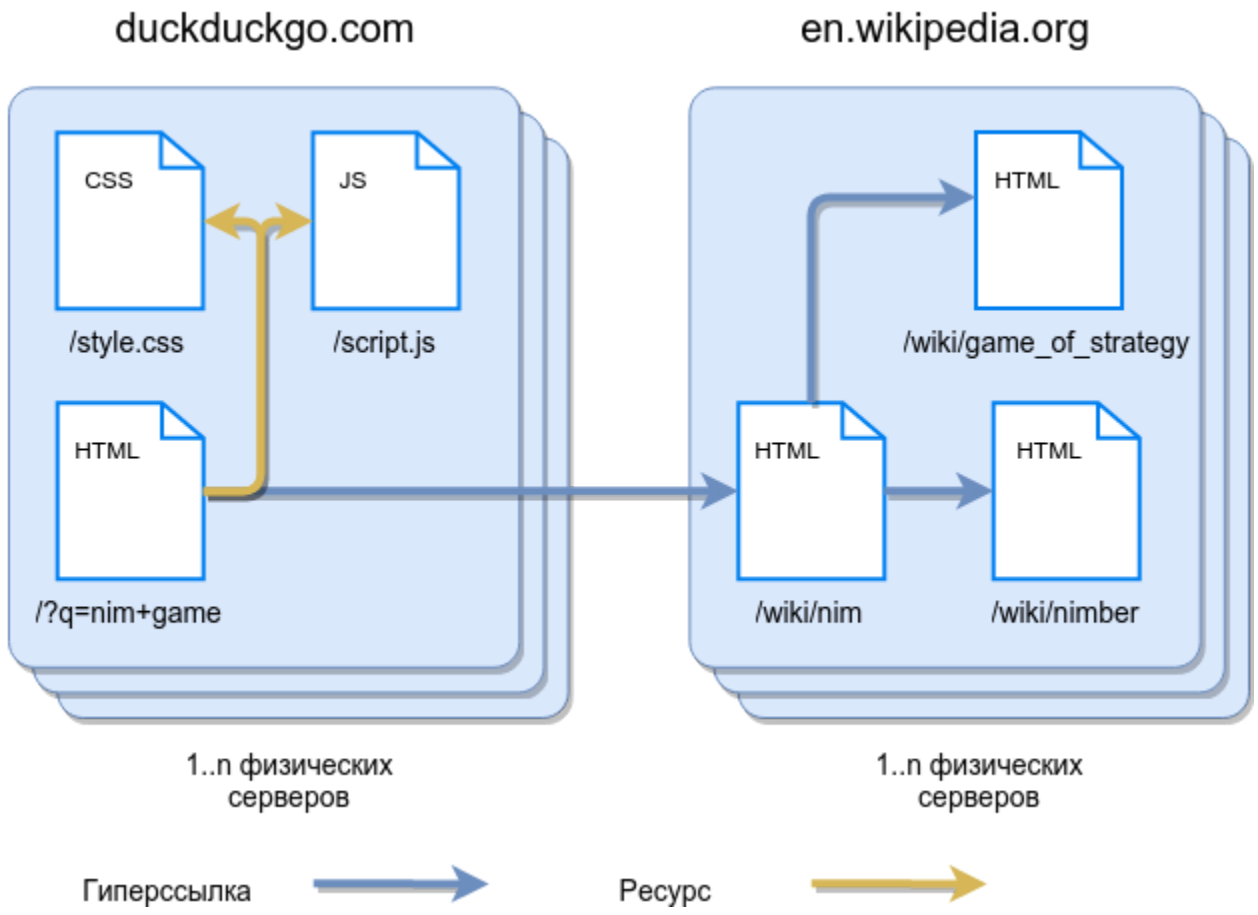
World Wide Web

WWW - множество **взаимосвязанных документов**,

располагающихся на машинах подключенных к Internet

WWW - набор протоколов, серверного и клиентского ПО,

позволяющих получать доступ к документам


duckduckgo.com
en.wikipedia.org
CSS
JS
/style.css
/script.js
HTML
/?q=nim+game
HTML
/wiki/game_of_strategy
HTML
/wiki/nim
HTML
/wiki/nimber
1..n физических серверов
1..n физических серверов
Гиперссылка
Ресурс

Документы

Типы документов (MIME-типы)

- text/html
- text/css
- text/javascript
- image/png
- video/mp4
- text/xml
- application/json
- Полный список MIME типов

Расширения файлов играют второстепенную роль

text/html

```
<html>
<body>
  <link rel="stylesheet" href="/css/style.css">
  <script src="http://code.jquery.com/jquery-2.1.4.js">
  </script>
  <p>Some text with <img src="pic/img1.png">
      and <a href="#yes">hyperlinks</a>
  </p>
</body>
</html>
```

text/css

```
.hljs-subst,
.hljs-title,
.json .hljs-value {
  font-weight: normal;
  color: #000;
}
```

text/xml

```
<response status="ok">
    <friends>
        <friend id="1" name="v.pupkin"/>
        <friend id="2" name="a.pushkin"/>
        <friend id="3" name="n.tesla"/>
    </friends>
</response>
```

application/json

```
{
    "status": "ok",
    "friends": [
        { "id": 1, "name": "v.pupkin" },
        { "id": 2, "name": "a.pushkin" },
        { "id": 3, "name": "n.tesla" }
    ]
}
```

Документы могут быть

- **Статические**
  - Это файлы на дисках сервера
  - Как правило, обладают постоянным адресом
- **Динамические**
  - Создаются на каждый запрос
  - Содержимое зависит от времени и пользователя
  - Адрес может быть постоянным или меняться

URL

URL - unified resource locator

```
http://server.org:8080/path/doc.html?a=1&b=2#part1
```

- http - протокол
- server.org - DNS имя сервера
- 8080 - TCP порт
- /path/doc.html - путь к файлу
- a=1&b=2 - опции запроса
- part1 - якорь, положение на странице

Абсолютные и относительные URL

- `http://server.org/1.html` - абсолютный
- `//server.org/1.html` - абсолютный (schemeless)
- `/another/page.html?a=1` - относительный (в пределах домена)
- `pictures/1.png` - относительный (от URL текущего документа)
- `?a=1&b=2` - относительный (от URL текущего документа)
- `#part2` - относительный (в пределах текущего документа)

Правила разрешения URL

`https://site.com/path/page.html` - основной документ

+ `http://wikipedia.org` = `http://wikipedia.org`
+ `//cdn.org/jquery.js` = **`https`**`://cdn.org/jquery.js`
+ `/admin/index.html` = **`https://site.com`**`/admin/index.html`
+ `another.html` = **`https://site.com/path/`**`another.html`
+ `?full=1` = **`https://site.com/path/page.html`**`?full=1`
+ `#chapter2` = **`https://site.com/path/page.html`**`#chaprer2`

Как документы могут ссылаться друг на друга?

HTML - гиперссылки

```
Список товаров в <a href="/cart.php">корзине</a>
```

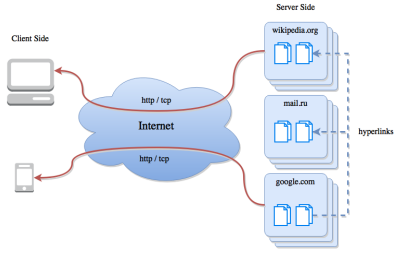
Список товаров в [корзине](/cart.php)

HTML - формы

```
<form action="https://duckduckgo.com/">
    <input type="text" name="q" value="">
    <input type="hidden" name="ia" value="images">
    <button type="submit">Найти</button>
</form>
```

HTML - ресурсы

```
<link rel="stylesheet" href="/css/index.css">
<script src="http://code.jquery.com/jquery-2.1.4.js">
</script>
<img src="pictures/network.png" width="200" >
```



CSS - ресурсы

```
.slide {
    background-image: url(../pictures/network.png)
}
@font-face {
    font-family: Terminus;
    src: url(fonts/terminus.ttf);
}
```

JavaScript - прямое указание URL

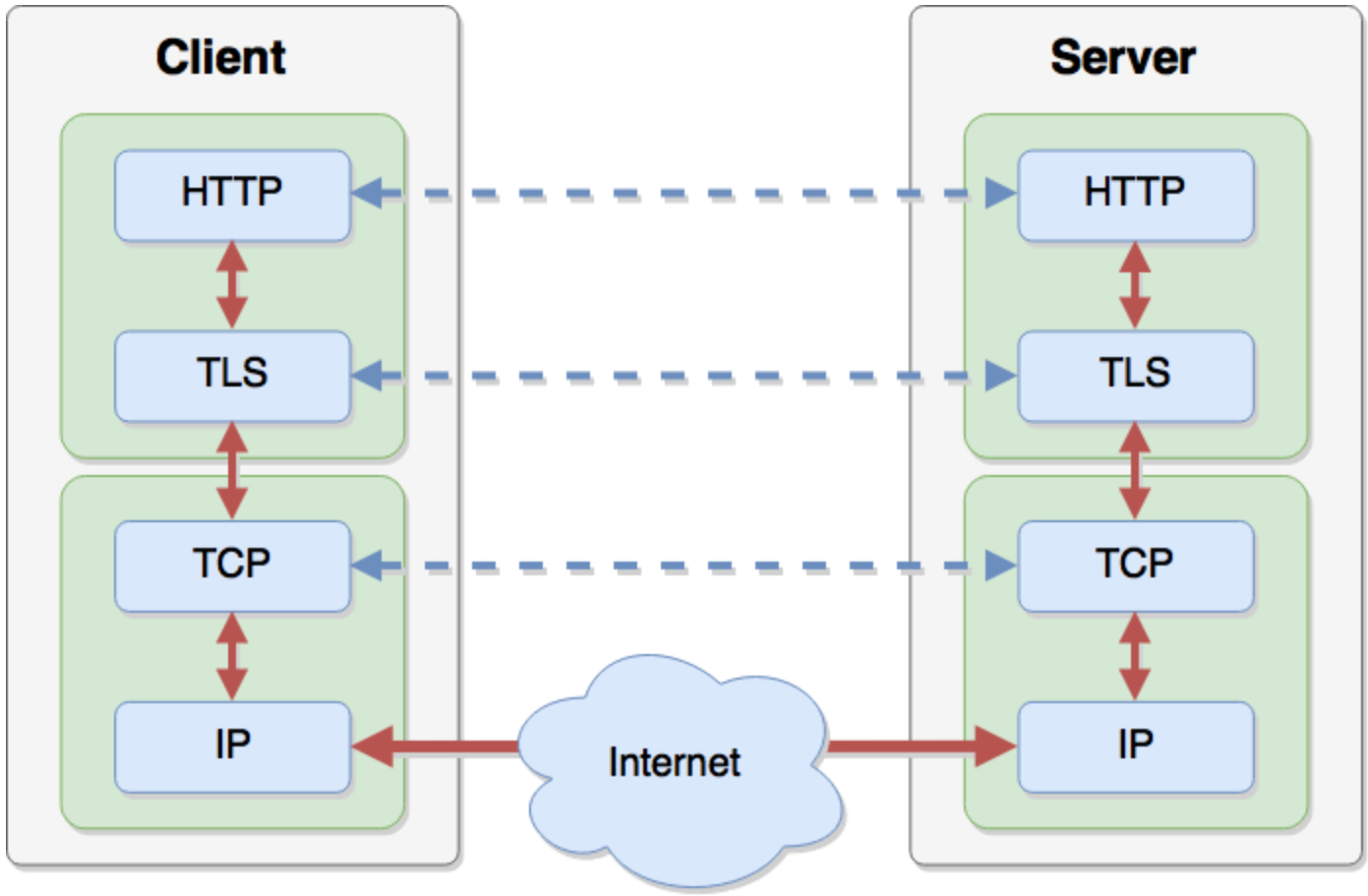
```
var saveApiUrl = '/items/save/';
var newTitle = 'Duck tales';
$.ajax({
    type: 'POST',
    url:  saveApiUrl,
    data: { id: 10, title: newTitle }
});
```

Клиент-серверная архитектура

Как происходит HTTP запрос ?

Как происходит HTTP запрос ?

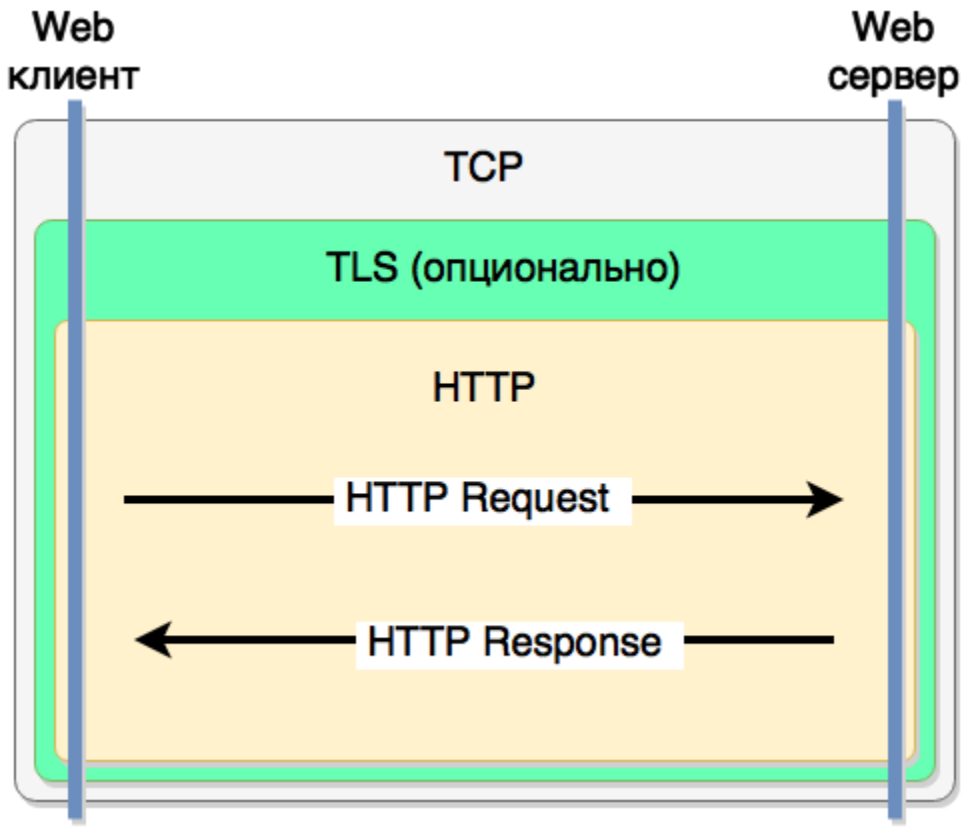
- Браузер анализирует введенный URL и извлекает имя хоста
- Используя систему DNS, браузер преобразует домен в ip адрес
- Устанавливает TCP соединение с web-сервером
- Если протокол https, устанавливает TLS соединение поверх TCP
- Формирует HTTP запрос, отправляет его, HTTP ответ
- Браузер закрывает соединение (для HTTP/1.0)
- Далее процесс парсинга и отображения документа ...


Client
Server
HTTP
HTTP
TLS
TLS
TCP
TCP
IP
Internet
IP

HTTP

Какие задачи решает HTTP?

- Передача документов
- Передача мета-информации
- Авторизация
- Поддержка сессий
- Кеширование документов
- Согласование содержимого (negotiation)
- Управление соединением


Web
клиент
Web
сервер
TCP
TLS (опционально)
HTTP
HTTP Request
HTTP Response

Ключевые особенности HTTP

- Работает поверх TCP/TLS
- Протокол запрос-ответ
- Не поддерживает состояние (соединение) - **stateless**
- **Текстовый** протокол
- Расширяемый протокол

HTTP/1.0 запрос

```
GET http://www.ru/robots.txt HTTP/1.0
Accept: text/html, text/plain
User-Agent: telnet/hands
If-Modified-Since: Fri, 24 Jul 2015 22:53:05 GMT
```

Перевод строки - `\r\n`

HTTP/1.1 запрос

```
GET /robots.txt HTTP/1.1
Accept: text/html,application/xhtml+xml
Accept-Encoding: gzip, deflate
Cache-Control: max-age=0
Connection: keep-alive
Host: www.ru
User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/39.0
```

HTTP/1.1 ответ

```
HTTP/1.1 404 Not Found
Server: nginx/1.5.7
Date: Sat, 25 Jul 2015 09:58:17 GMT
Content-Type: text/html; charset=iso-8859-1
Connection: close

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>...
```

HTTP запрос состоит из

- строка запроса
  - метод
  - URL документа
  - версия
- заголовки
- тело запроса

HTTP методы

- **GET** - получение документа
- **HEAD** - получение только заголовков
- **POST** - отправка данных на сервер
- PUT - отправка документа на сервер (*)
- DELETE - удаление документа (*)
- CONNECT, TRACE, OPTIONS - используются редко (*)
- COPY, MOVE, MKCOL - расширения WebDAV (*)

HTTP ответ состоит из

- строка ответа
  - метод
  - URL документа
  - версия
- заголовки
- тела ответа - документ

HTTP коды ответа

- 1xx - информационные
- 2xx - успешное выполнение
- 3xx - перенаправления
- 4xx - ошибка на стороне клиента
- 5xx - ошибка на стороне сервера

HTTP коды ответа (1)

- 200 OK - запрос успешно выполнен
- 204 No Content - запрос успешно выполнен, но документ пуст
- 301 Moved Permanently - документ сменил URL
- 302 Found - повторить запрос по другому URL
- 304 Not Modified - документ не изменился, использовать кеш

HTTP коды ответа (2)

- 400 Bad Request - неправильный синтаксис запроса
- 401 Unauthorized - требуется авторизация
- 403 Forbidden - нет доступа (неверная авторизация)
- 404 Not Found - документ не найден
- 500 Internal Server Error - неожиданная ошибка сервера (application)
- 502 Bad Gateway - проксируемый сервер отвечает с ошибкой
- 504 Gateway Timeout - проксируемый сервер не отвечает

Заголовки HTTP (общие)

Для управления соединением и форматом сообщения (документа)

- Content-Type - MIME тип документа
- Content-Length - длина сообщения
- Content-Encoding - кодирование документа, например gzip-сжатие
- Transfer-Encoding - формат передачи, например, chunked
- Connection - управление соединением
- Upgrade - смена протокола

Заголовки HTTP запросов

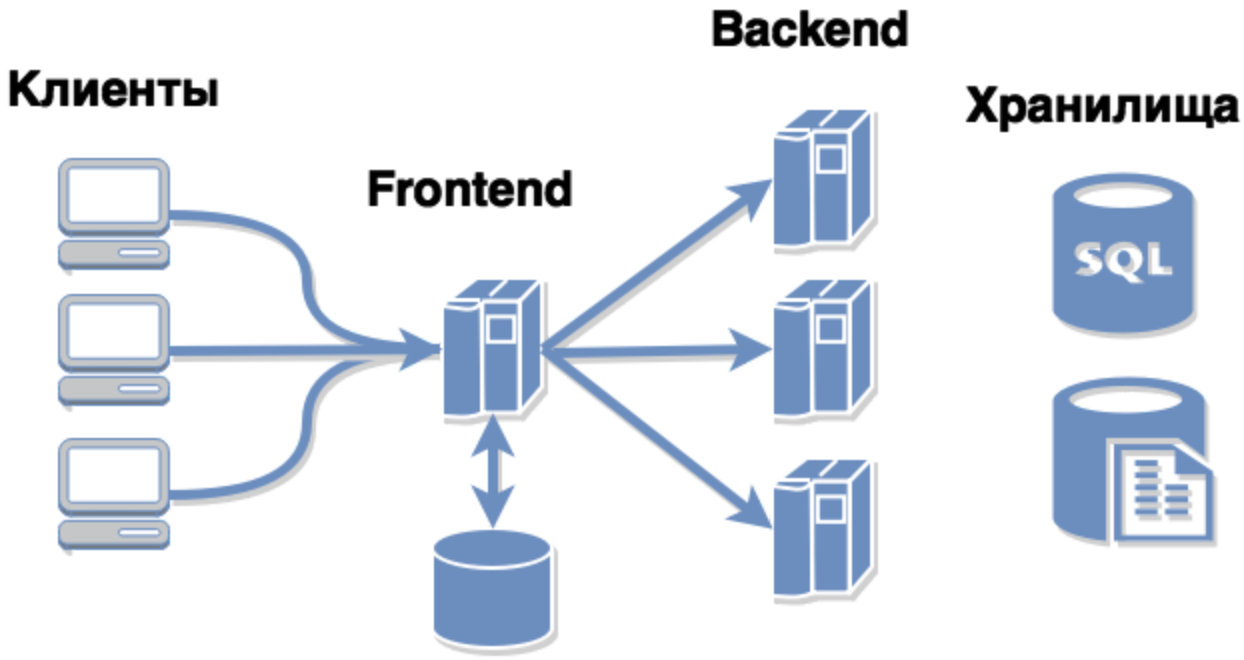
- Authorization - авторизация, чаще всего логин/пароль
- Cookie - передача состояния (сессии) на сервер
- Referer - URL предыдущего документа, контекст запроса
- User-Agent - описание web-клиента, версия браузера
- If-Modified-Since - условный GET запрос
- Accept-* - согласование (negotiation) содержимого

Заголовки HTTP ответов

- Location - новый URL документа при перенаправлениях
- Set-Cookie - установка состояния (сессии) в браузере
- Last-Modified - дата последнего изменения документа
- Date - Дата на сервере, для согласования кешей
- Server - описание web-сервера, название и версия

Трехзвенная архитектура

Общая архитектура



Задачи Frontend (web) сервера

- отдача статических документов
- проксирование (reverse proxy)
- балансировка нагрузки
- кеширование
- сборка SSI
- авторизация, SSL, нарезка картинок, gzip

Reverse proxy

- frontend (медленно) читает запрос от клиента
- frontend (быстро) передает запрос свободному backend
- backend генерирует страницу
- backend (быстро) возвращает ответ frontend серверу
- frontend (медленно) возвращает ответ клиенту

Результат: backend занят минимально возможное время.

Web сервер


NGINX
http server


APACHE
HTTP SERVER


LIGHTTPD
fly light.


IIS7
internet information services

Запуск web сервера

- Команда на запуск

  ```
  sudo /etc/init.d/nginx start
  ```

- Чтение файла конфигурации
- Получение порта 80
- Открытие (создание) логов
- Понижение привилегий
- Запуск дочерних процессов/потоков (*)
- Готов к обработке запроса

Файлы web сервера

- Конфиг `/etc/nginx/nginx.conf`

  `include /etc/nginx/sites-enabled/*`
- Init-скрипт `/etc/init.d/nginx [start|stop|restart]`
- PID-файл `/var/run/nginx.pid`
- Error-лог `/var/log/nginx/error.log`
- Access-лог `/var/log/nginx/access.log`

Процессы web сервера

- Master (root, 1 процесс)
  - Чтение и валидация конфига
  - Открытие сокета (ов) и логов
  - Запуск и управление дочерними процессами (worker)
  - Graceful restart, Binary updates
- Worker (www-data, 1+ процессов)
  - Обработка входящих запросов

Конфигурация web сервера

Терминология

virtual host, **вирт. хост** - секция конфига web сервера, отвечающая за обслуживание определенного домена

location - секция конфига, отвечающая за обслуживание определенной группы URL

```
user  www www;
error_log  /var/log/nginx.error_log  info;
http {
  include       conf/mime.types;
  default_type  application/octet-stream;
  log_format    simple '$remote_addr $request $status';
  server {
    listen        80;
    server_name   one.example.com www.one.example.com;
    access_log    /var/log/nginx.access_log simple;
    location / {
      root        /www/one.example.com;
    }
    location ~* ^.+\.(jpg|jpeg|gif)$ {
      root        /www/images;
      access_log  off;
      expires     30d;
    }
  }
}
```

Секции и директивы

- `http` — конфигурация для HTTP сервера
- `server` — конфигурация домена (вирт. Хоста)
- `server_name` — имена доменов
- `location` — локейшен, группа URL
- `root`, `alias` — откуда нужно брать файлы
- `error_log` — лог ошибок сервера
- `access_log` — лог запросов

Приоритеты location в nginx

- `location = /img/1.jpg`
- `location ^~ /pic/`
- `location ~* \.jpg$`
- `location /img/`

При одинаковом приоритете используется тот location, что находится **выше** в конфиге.

Отдача статических документов

```
location ~* ^.+\.(jpg|jpeg|gif|png)$ {
    root         /www/images;
}
location /sitemap/ {
    alias /home/www/generated/;
}
```

`/2015/10/ae2b5.png` → `/www/images/2015/10/ae2b5.png`

`/sitemap/index.xml` → `/home/www/generated/index.xml`

Атрибуты файлов и процессов

У процесса есть

- пользователь
- группа

У файла (или директории) есть

- пользователь (владелец)
- группа
- права доступа (read/write/execute)

Как узнать атрибуты ?

```
$ ps -o pid,euser,egroup,comm,args -C nginx
  PID EUSER    EGROUP   COMMAND
29731 root     root     nginx: master process /usr/sbin/nginx
29732 www-data www-data nginx: worker process
29733 www-data www-data nginx: worker process
29734 www-data www-data nginx: worker process
29737 www-data www-data nginx: worker process

$ ls -lah www/index.html
-rw-r--r-- 1 nuf users 156K Feb  6 21:15 www/index.html
```

Проверка доступа

Для того, чтобы открыть файл, необходимо иметь права на чтение `r` самого файла и на исполнение `x` директорий, в которых он находится. Наличие прав проверяется следующим образом:

- Если совпадает пользователь `-rw-r--r--`
- Если совпадает группа `-rw-r--r--`
- Иначе `-rw-r--r--`

Application сервер

Backend (application) сервер

Роль application сервера заключается в исполнении бизнес-логики приложения и генерации динамических документов.

На каждый HTTP запрос application сервер запускает некоторый обработчик в приложении. Это может быть функция, класс или программа, в зависимости от технологии.

Протоколы запуска приложения

- Servlets и др. специализированные API
- mod_perl, mod_python, mod_php
- CGI
- FastCGI
- SCGI
- PSGI, **WSGI**, Rack

WSGI

WSGI - актуальный протокол

WSGI, PSGI, Rack - протоколы вызова функции обработчика из application сервера. Сам application server при этом может выполняться в отдельном процессе или совпадать с web сервером. Как правило, при использовании этих протоколов в качестве application сервера выступает отдельный легковесный процесс.

WSGI - обработчик

```
def wsgi_application(environ, start_response):
    # бизнес-логика
    status = '200 OK'
    headers = [
        ('Content-Type', 'text/plain')
    ]
    body = 'Hello, world!'.encode('utf-8')
    start_response(status, headers)
    return [ body ]
```

В файле `app/main.py`

Web Server Gateway Interface

- Обработчик - функция или класс (callable)
- Метод, QueryString, заголовки запроса - через аргумент **environ**
- Тело запроса передается через file-handle **wsgi.input**
- HTTP код ответа и заголовки ответа передаются через вызов функции **start_response**
- Тело ответа возвращается в виде списка (iterable) из обработчика
- Поток ошибок должен быть направлен в file-handle **wsgi.stderr**

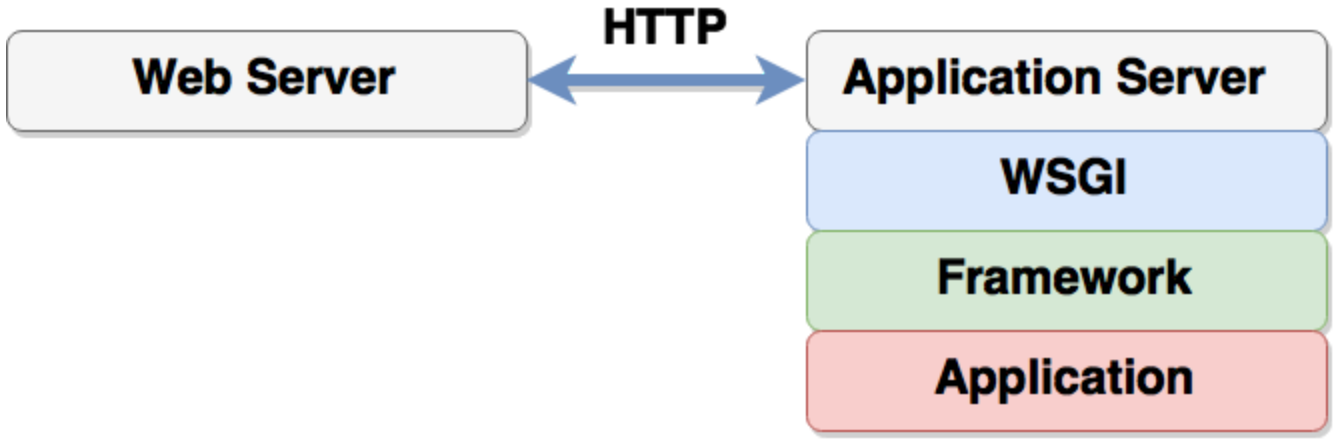
Переменные environ

- CGI-like переменные: `REQUEST_URI`, ...
- `wsgi.version` - версия WSGI протокола
- `wsgi.url_scheme` - схема текущего URL: https или http
- `wsgi.input` - file-handle для чтения тела запроса
- `wsgi.errors` - file-handle для вывода ошибок
- `wsgi.multithreaded` - ...
- `wsgi.multiprocess` - ...

Переменные окружения CGI

- `REQUEST_METHOD` - метод запроса
- `PATH_INFO` - путь из URL
- `QUERY_STRING` - фрагмент URL после `?`
- `REMOTE_ADDR` - IP адрес пользователя
- `CONTENT_LENGTH` - длина тела запроса
- `HTTP_COOKIE` - Заголовок `Cookie`
- `HTTP_ANY_HEADER_NAME` - любой другой HTTP заголовок

Развертывание WSGI



Что ложится на приложение ?

- Анализ `PATH_INFO` и выбор конкретного обработчика
- Разбор конкретных заголовков, например `Cookie`
- Разбор `QUERY_STRING`
- Разбор тела запроса:
  - x-www-form-urlencoded
  - multipart/form-data
- Вывод правильных заголовков ответа

Установка и запуск Gunicorn

```
# Внутри virtualenv
pip install gunicorn
pip freeze > requirements.txt

# находясь в директории проекта и в virtualenv
gunicorn --chdir app main:wsgi_application

# или из любого состояния
/home/nuf/quack/venv/bin/gunicorn \
    --chdir /home/nuf/quack/app main:wsgi_application
```

Настройка проксирования в nginx

Настройка проксирования в nginx

```
proxy_set_header Host      $host;
proxy_set_header X-Real-IP $remote_addr;
location / {
    proxy_pass http://backend;
}
location /partner/ {
    proxy_pass http://www.partner.com;
}
location ~ \.\w\w\w?\w?$ {
    root /www/static;
}
```

Настройка upstream в nginx

```
upstream backend  {
    server back1.example.com:8080 weight=1 max_fails=3;
    server back2.example.com:8080 weight=2;
    server unix:/tmp/backend.sock;
    server backup1.example.com:8080 backup;
    server backup2.example.com:8080 backup;
}
```