

Web клиенты

Разновидности web-клиентов

- Библиотеки в ЯП: libcurl, urllib, requests и т.д.
- Консольные утилиты: wget, curl, telnet!
- Роботы: поисковики, вредоносные скрипты
- Браузеры:
 - Полноценные: firefox, chrome и т.д.
 - Встроенные: web-view, webkit и т.д.

Особенности библиотек web-клиентов

- Предоставляют максимум опций для работы с HTTP
- Осуществляют кодирование / декодирование данных
- Перенаправления, куки - опционально

Назначение: используются внутри других программ для простой работы с HTTP

Пример использования requests

```
# pip install requests
import requests
url = 'https://api.site.com/method/'
params = {'argument1': 'value1',
          'argument2': 'value2'}
headers = {'User-Agent': 'python requests'}
response = requests.get(url, params=params, headers=headers)
response.text
# '{"type": "User", "name": "Pupkin" ...'
response.json()
# {'type': 'User', 'name': 'Pupkin', ...}
```

Назначение консольных клиентов

- Автоматизация в shell-скриптах
- Создание "статической копии сайта"
- Отладка web-приложений

Telnet

Telnet - это простейшее средство отладки. telnet открывает tcp соединение и связывает его с консолью, позволяя общаться с web-сервером напрямую с клавиатуры.

ПОРТ, НА КОТОРОМ ОТКРЫТО
СОЕДИНЕНИЕ

```
[nuf@nuftop tmp]$ telnet www.ru 80
```

```
Trying 217.112.35.75...
```

```
Connected to www.ru.
```

```
Escape character is '^]'.
```

```
GET /index.html HTTP/1.1
```

```
Host: www.ru
```

```
HTTP/1.1 404 Not Found
```

```
Server: nginx/1.5.7
```

```
Date: Sun, 26 Jul 2015 14:43:18 GMT
```

```
Content-Type: text/html; charset=iso-8859-1
```

```
Transfer-Encoding: chunked
```

```
Connection: keep-alive
```

```
Keep-Alive: timeout=20
```

Еще примеры отладки

GET запрос к серверу с отображением всех заголовков:

```
curl -vv 'http://api.site.com/method/?arg=1'
```

POST запрос к серверу с авторизацией и передачей доп.
заголовков:

```
curl -vv -d 'arg=1' -H 'X-Token: 123'
```

```
'http://api.site.com/method/'
```


Браузер

Основное назначение - отображение HTML страниц.

Однако, возможности современных браузеров огромны.

Существуют операционные системы и 3D-игры, работающие внутри браузеров!

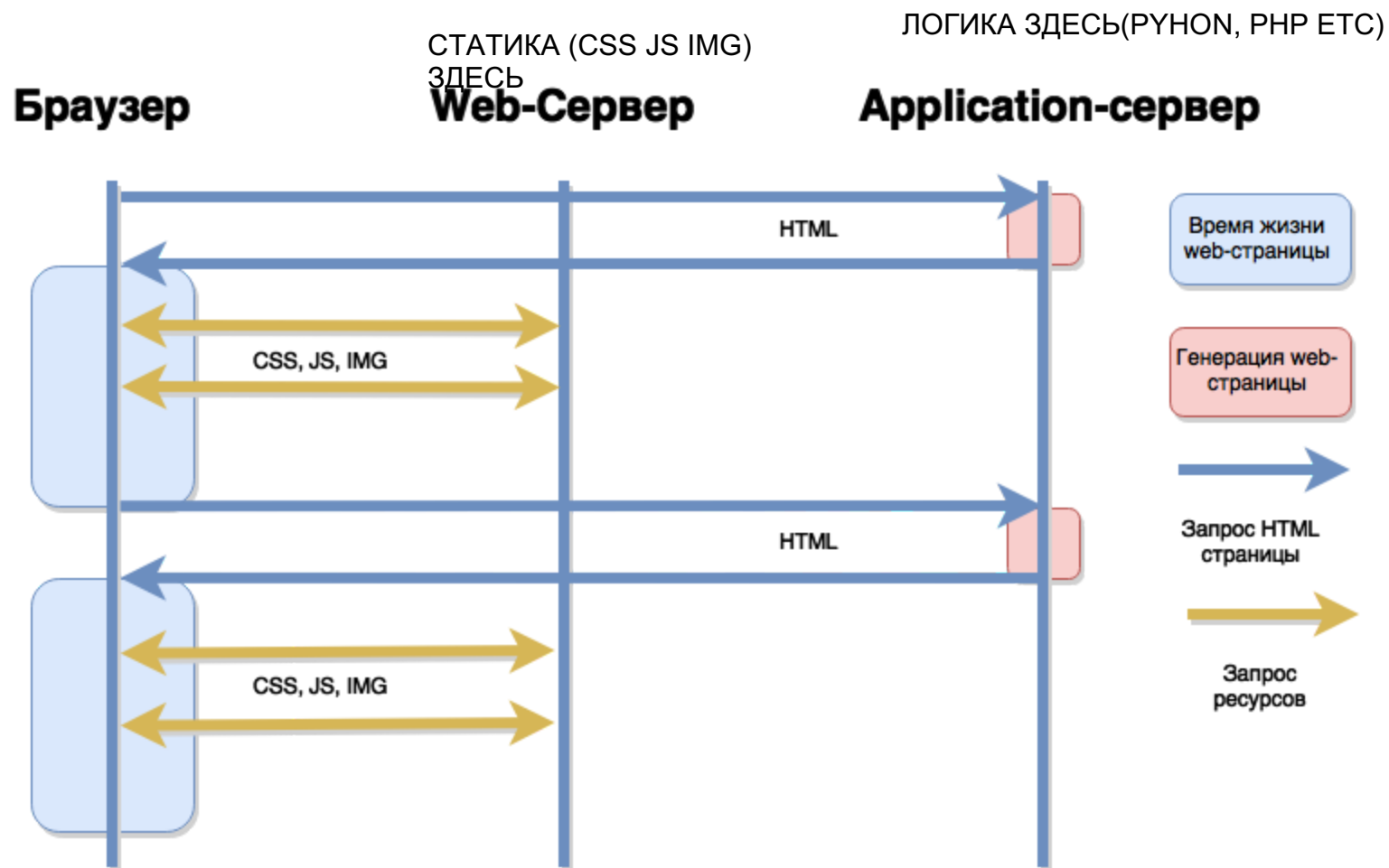
www.evolutionoftheweb.com

Сценарий работы web приложения

(КЛАССИЧЕСКИЙ)

- Пользователь вводит URL
- Браузер загружает Web страницу - HTML документ ЗАГРУЖАЕТ ВСЕГ HTML ДОКУМЕНТ
- Браузер анализирует (parse) HTML и загружает доп. ресурсы IMG, JS, CSS
- Браузер отображает (rendering) HTML страницу ОТРИСОВКА ДОКУМЕНТА
- Пользователь переходит по гиперссылке или отправляет форму АНАЛОГИЧНО
ВВОДУ НОВОГО
URL
- Цикл повторяется

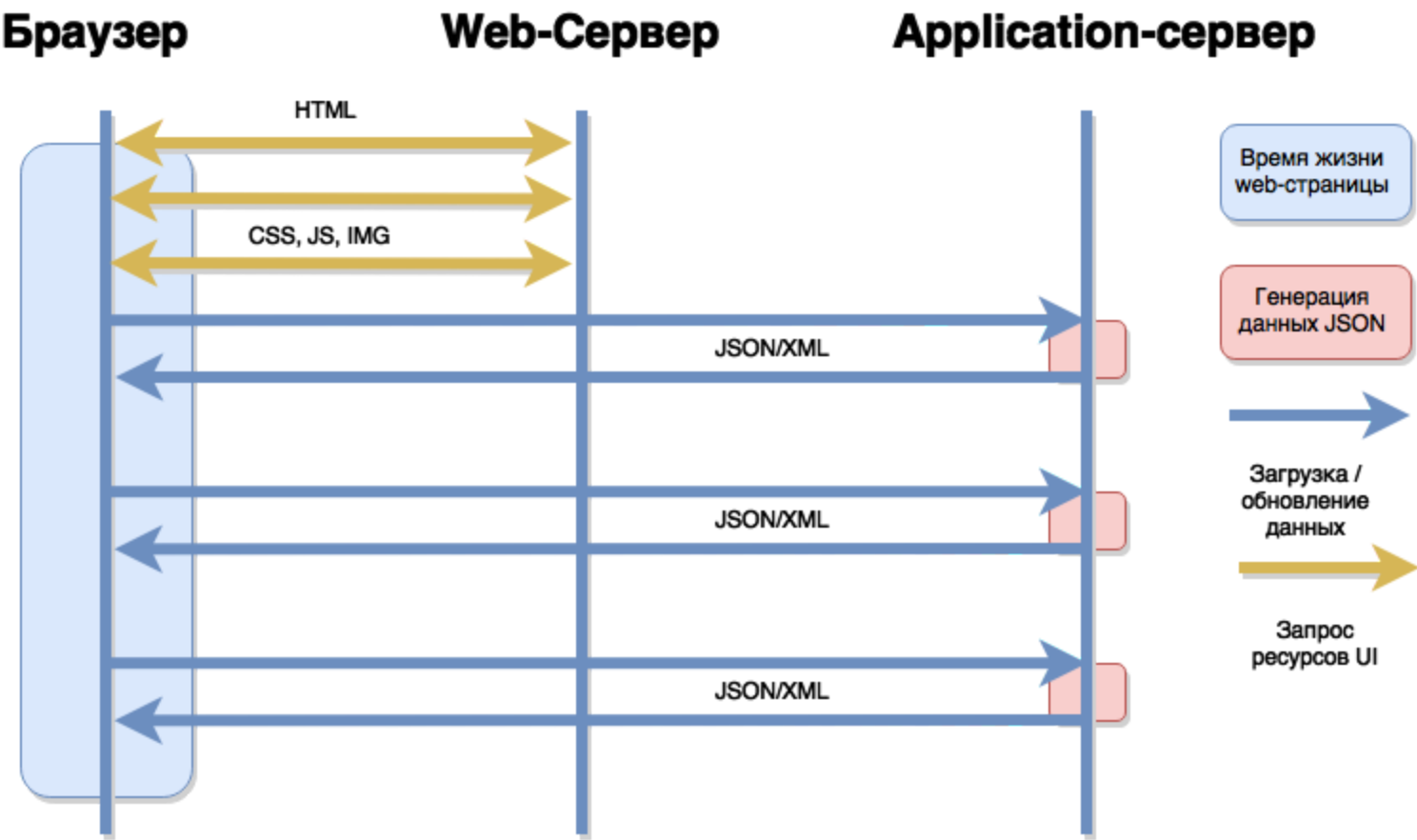
ТАКЖЕ ПРЕДПОЛАГАЮТСЯ АЛГОРИТМЫ КЕШИРОВАНИЯ В
БРАУЗЕРЕ



Сценарий работы современного приложения

- Браузер загружает Web страницу, ресурсы и отображает ее СТРАНИЦА ПОЧТИ ПУСТАЯ, ОНА СТАТИЧЕСКАЯ
- JavaScript загружает данные с помощью AJAX запросов ПОСЛЕ ЗАПРОСА JS ОН ЗАГРУЖАЕТСЯ И НАЧИНАЕТ ЗАПРАШИВАТЬ ДАННЫЕ С СЕРВЕРА
- JavaScript обеспечивает полноценный UI на странице ШАБЛОНИЗАЦИЯ НА КЛИЕНТЕ С ПОМОЩЬЮ JS
- Пользователь взаимодействует с UI, что приводит к вызову JavaScript обработчиков
- JavaScript обновляет данные на сервере или загружает новые данные, используя AJAX ДОЗАПРОС И ДООТРИСОВКА НОВЫХ ДАННЫХ ПРИ ПЕРЕХОДЕ НА ДРУГУЮ СТРАНИЦУ

В РАМКАХ ОДНОГО САЙТА ПРИНЦИП SPA ОБЕСПЕЧИВАЕТ МИНИМУМ ПЕРЕЗАГРУЗОК(ХОДИМ ЗА ДАННЫМИ(JSON/XML) НА БЭКЕНД, ПАРСИМ ИХ И ИСПОЛЬЗУЕМ В js ДЛЯ ПРОРИСОВКИ UI)



Особенности современных Web-приложений

- UI находится на 1 или нескольких страницах (single page application / SPA)
- UI полностью статичен: HTML, CSS, JS - статические файлы
- Логика UI полностью работает на стороне клиента
- Используется шаблонизация в JavaScript
- Application сервер возвращает чистые данные (JSON или XML, а не HTML)