



# Информатика

## введение в http-серверы

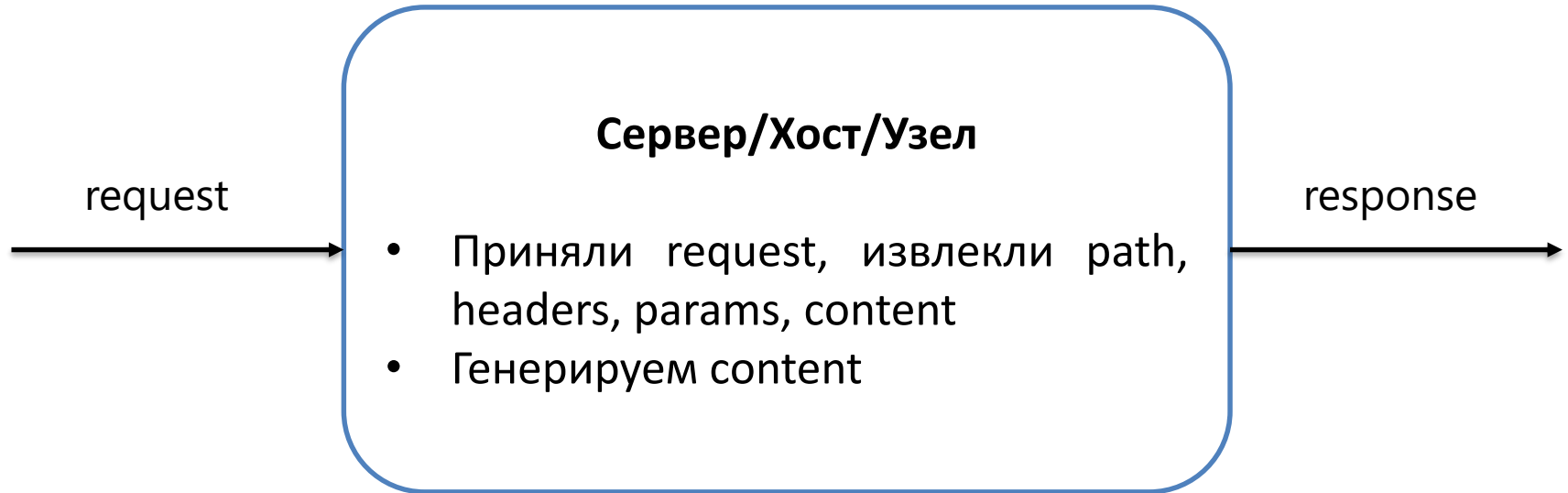
© Марченко Антон Александрович  
Абрамский Михаил Михайлович

2017 г.

# На прошлой лекции ...

- Интернет, веб
- Клиент, сервер
- Запрос, ответ
- Генерация страниц

# Принцип работы



# Вопросы

- Как обрабатываются запросы и отправляются ответы?
- Как генерируется содержимое ответа?

# Работа сервера

- Сервер работает **24/7**
  - Что вернёт http-запрос, если сервер не работает?

# Работа сервера

- Сервер работает **24/7**
  - Что вернёт http-запрос, если сервер не работает?
- Что значит «**работает**»?
  - На нём *запущены приложения* (процессы)
    - В чём разница между приложением и процессом?

# Обработка запроса

- Приложение работают постоянно
- Некоторые приложения сетевые – обрабатывают запросы
- *Если сетевых приложений несколько, как понять, кто должен получить запрос?*

# Порты

- Точки подключения к хосту (серверу, узлу)
- Записываются в заголовках протокола транспортного уровня OSI (TCP, UDP, ...)

<http://10.12.13.14:1234/>

- Определяют процесс-получатель пакета



# Открытые порты

- Процессы могут *слушать* порты (ожидая событий)
- Открытый порт – доступный *извне* порт, прослушиваемый процессом

# Порты по умолчанию:

- Набирали порты в адресной строке?
- Существуют порты по умолчанию
  - HTTP:
  - HTTPS:
  - SSH:
  - FTP:

# Номера портов

- Выдаются ОС процессам
  - Могут быть конфликты (например, со Skype, по умолчанию слушающему 80 и 443 порты)
- Порты одного протокола не пересекаются с портами другого (TCP и UDP, например)
- Во многих ОС прослушивание портов 0-1023 требует особых привилегий

# Вопросы

- ✓ Как обрабатываются вопросы и отправляются ответы?
- Как генерируется содержимое ответа?
  - Есть процесс/приложение, слушающее некий порт, получающее/отправляющее данные

# Наши программы ...

... обычно запускались при необходимости, отработывали и завершали работу

А как должно быть на сервере?

# На сервере

- Должно быть постоянно запущено приложение (процесс, программа, сервис)
- Получающее и обрабатывающее запросы
- Исполняющее некоторый код для генерации ответа
- Отправляющее содержимое клиенту

# Web-сервер

- Принимает запросы, отдаёт ответы
- Примеры: Apache, nginx, IIS
- Сам ничего не генерирует. Работает вместе с (web-)приложениями, которые могут это сделать
  - Google: CGI (Common Gateway Interface)

# Web-сервер

- Web-сервер обрабатывает задачи по обработке http запросов/ответов
- Некоторые задачи не типичны для него
  - Бесперебойность
  - Распределение нагрузки
  - Кластеризация



# Сервер приложений

- Программный фреймворк, объединяющий средства для создания web-приложений и окружения для их работы
- Позволяет разработчикам сосредоточиться на бизнес-логике
- Берёт на себя все побочные задачи

# Сервер приложений

- Традиционно Microsoft позиционирует в качестве сервера приложений связку на основе Windows Server
  - IIS в качестве web-сервера
  - .NET Framework в качестве среды приложений
  - ASP.NET для серверной обработки и скриптинга
  - WCF/Web API web-сервисы

# Не только Windows Server

- .NET Core позволяет разрабатывать и хостить приложения на Linux
- Используется http-сервер Kestrel
- Возможности серверной части беднее, чем в IIS, Apache или Nginx
  - Можно работать с Nginx

# Internet Information Server

- **Работа web-сервера IIS**

- Драйвер http.sys перехватывает запросы
- http.sys обращается к Windows Activation Service
- WAS запрашивает конфигурацию
- W3SVC получает конфигурационную информацию (данные пула приложений, параметрах приложений)
- В отдельном процессе исполняется приложение, формирующее ответ http.sys
- http.sys отправляет ответ браузеру клиента

# Работа IIS

- IIS управляет пулом web-приложений
- Разделяет адресные пространства
- Заботится об отказоустойчивости сайта и web-сервера в целом

# Разработка web-приложений

- Обычно разработка web-приложений на .NET Framework осуществляется с помощью инструментария ASP.NET
- Приложения-hostятся на web-сервере или независимо (self-hosted)

# Self-hosted http-сервер

- Рассмотрим *приложение, самостоятельно слушающее порт по некоторому адресу, обрабатывающее запросы и генерирующее ответы*
  - Такое приложение вам предстоит разработать в качестве семестровой работы
- Для чего это нужно?

# Идея (еще раз)

- Приложение слушает http запросы
- Обрабатывает запросы
- Генерирует ответ
- Отправляет ответ клиенту



# HttpListener

- Класс `System.Net` для *прослушивания* подключений по протоколу HTTP
- Реализован на основе драйвера `http.sys` ОС Windows
  - Как и IIS 6+
  - `http.sys` управляет очередью запросов, перенаправляет запросы приложениям

# Точка доступа приложения

- HttpListener перед началом работы нужно указать адреса для обращения к приложению через свойство `Prefixes`
- `Start/Stop` – запускает/останавливает прослушку

```
var listener = new HttpListener();  
listener.Prefixes.Add("http://localhost:8080");  
listener.Start();  
listener.Stop();
```

# Прослушивание подключения

- `getContext` позволяет синхронно (поток блокируется) ожидать подключение
- Результат работы – `HttpListenerContext`, позволяющий получить доступ к объектам запроса и ответа
  - `HttpRequest` и `HttpResponse` соответственно

# HttpListenerRequest

- Что было у HTTP запроса? Всё это есть!
- `Url`, `RawUrl` – куда посылался запрос
- `Headers` – заголовки запроса
- `QueryString` – строка запроса с параметрами
  - Нужно парсить (вспомним регулярки)

# HttpListenerResponse

- ContentLength24
- OutputStream

– *Еще пару свойств посмотрим на следующей лекции*

Работа с записью данных ответа сводится к  
ИСПОЛЬЗОВАНИЮ ПОТОКОВ ДАННЫХ

# Синхронное прослушивание

```
HttpListenerContext context = listener.GetContext();
```

```
var content =
```

```
"<html>
```

```
<head>
```

```
<meta charset='utf8'>
```

```
</head>
```

```
<body>Привет мир!</body>
```

```
</html>";
```

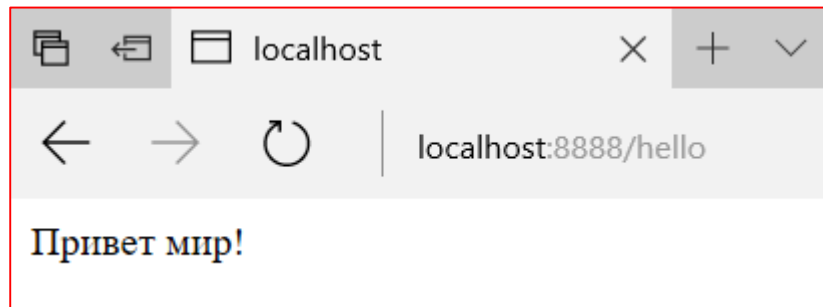
```
var bytes = Encoding.UTF8.GetBytes(content);
```

```
context.Response.ContentLength64=bytes.Length;
```

```
context.Response.OutputStream;
```

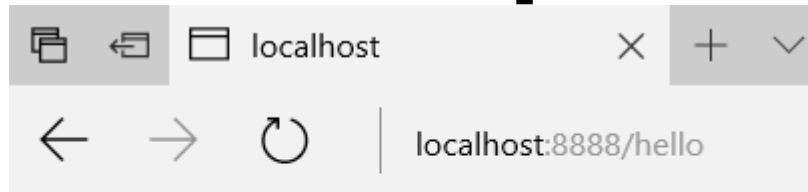
```
sw.Write(bytes,0,bytes.Length);
```

```
sw.Close();
```



# Обработка нескольких запросов

```
int count = 0;
while (true)
{
    HttpContext context = listener.GetContext();
    HttpResponse response = context.Response;
    string responseStr =
        "<html><head><meta charset='utf8'></head>
        <body>Привет мир({++count})!</body></html>";
    byte[] buffer = Encoding.UTF8.GetBytes(responseStr);
    response.ContentLength64 = buffer.Length;
    response.OutputStream.Write(buffer, 0, buffer.Length);
    response.OutputStream.Close();
}
```



Привет мир(20)!

# Асинхронная версия

```
static void Main()=>Listen().Wait();

private static async Task Listen()
{
    HttpListener listener = new HttpListener();
    listener.Prefixes.Add("http://localhost:8080/hello/");
    listener.Start();
    HttpListenerContext context = await listener.GetContextAsync();
    HttpListenerResponse response = context.Response;
    string responseStr = "<html><head><meta charset='utf8'></head>
        <body>Привет мир!</body></html>";
    byte[] buffer = Encoding.UTF8.GetBytes(responseStr);
    response.ContentLength64 = buffer.Length;
    response.OutputStream.Write(buffer, 0, buffer.Length);
    response.OutputStream.Close();
    listener.Stop();
}
```



# Асинхронная версия с циклом

```
private static async Task Listen()
{
    HttpListener listener = new HttpListener();
    listener.Prefixes.Add("http://localhost:8080/hello/");
    listener.Start();
    while (true)
    {
        HttpListenerContext context = await listener.GetContextAsync();
        Task.Run(()=>
        {
            HttpListenerResponse response = context.Response;
            string responseStr = @"<html><head><meta charset='utf8'></head>
                                   <body>Привет мир!</body></html>";
            byte[] buffer = Encoding.UTF8.GetBytes(responseStr);
            response.ContentLength64 = buffer.Length;
            response.OutputStream.Write(buffer, 0, buffer.Length);
            response.OutputStream.Close();
        });
    }
    listener.Stop();
}
```

# Что выполнять при запросах?

- Сравним запросы:
  - GET vk.com/im – страница сообщений
  - GET vk.com/friends – страница друзей

Отличаются path части у запросов

# Маршрутизация URL

Выбор обработчика по запросу

- По точному совпадению path
- По наибольшему совпадению
- По расширению
- На обработчик по умолчанию

# Пример

**Watermelon**     */fruit/summer/\**

**Garden**         */seeds/\**

**List**             */seedlist*

**Kiwi**             *\*.abc*

1. <http://host:port/mywebapp/fruit/summer/index.html>
2. <http://host:port/mywebapp/fruit/summer/index.abc>
3. <http://host:port/mywebapp/seedlist/index.html>
4. <http://host:port/mywebapp/seedlist/pear.abc>

# Как маршрутизировать запросы?

- Парсить запрос, исполнять подходящий код
  - Как сделать без хардкода?
- Установить соответствие классам обработчикам по шаблонам
  - Можно рефлексией

# Маршрутизация в ASP.NET

- В ASP.NET есть удобный механизм маршрутизации, позволяющий настраивать сопоставление адресов с файлами на веб-узле
- Google: «Маршрутизация запросов ASP.NET»

# Всё вместе

- Есть обработчик запросов (controller)
- Есть шаблоны страниц для генерации ответов (view)
- Могут быть классы данных приложения, бизнес-логика (model)

# Не только страницы в ответе

- Web-приложения могут генерировать не только web-страницы в качестве ответа на запросы
- Можно генерировать файлы с данными для их последующей обработки клиентом



# Web API

- Подход к web-разработке
  - google: современная архитектура web-приложений
- Предоставление публичного интерфейса с установленной системой сообщений request-response
- Обычно обмен данными происходит в формате JSON или XML

# Зачем?

- У нас не только клиенты-браузеры
  - Каждый может отображать результат по-своему
- Можно генерировать view на клиенте
  - Облегчив тем самым сервер
  - Лучшее разделение ответственностей

# Бонус!

- Событийно-ориентированное программирование
  - Обработка web запросов
  - Графические приложения



Вопросы?

*e-mail:* [marchenko@it.kfu.ru](mailto:marchenko@it.kfu.ru)