NodeJS. Основы создания веб-приложений

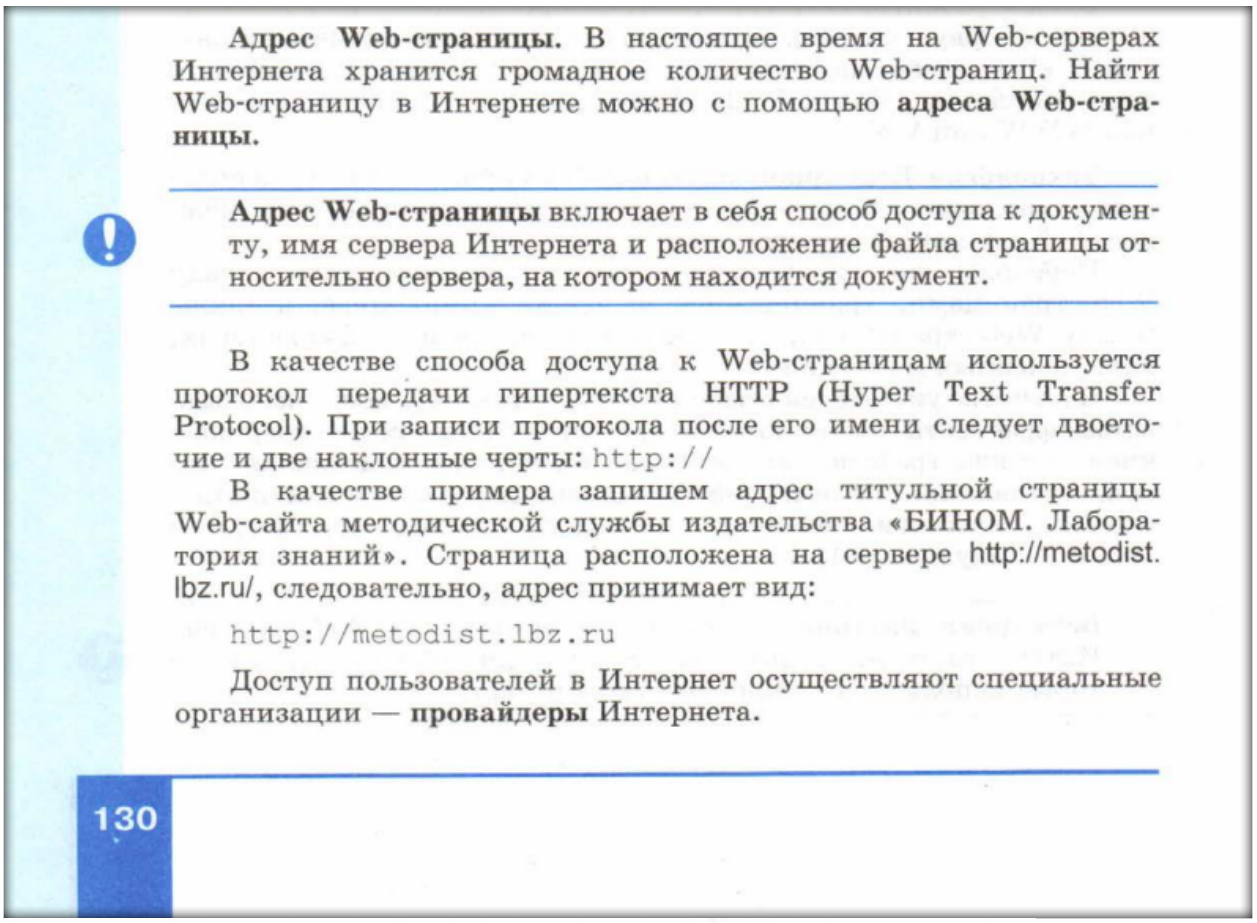
Сервинг статических файлов

Обычный веб-сервер типа Apache естественным образом обслуживает выдачу файлов.

Адреса веб-ресурсов вида

https://kodaktor.ru/api0/appver.html
https://kodaktor.ru/api2/time.php

наводят, возможно, на мысль о том, что на сервере существует некая папка api0 или api2, в которой находится файл appver.html или time.php



Адрес Web-страницы. В настоящее время на Web-серверах Интернета хранится громадное количество Web-страниц. Найти Web-страницу в Интернете можно с помощью **адреса Web-страницы.**

Адрес Web-страницы включает в себя способ доступа к документу, имя сервера Интернета и расположение файла страницы относительно сервера, на котором находится документ.

В качестве способа доступа к Web-страницам используется протокол передачи гипертекста HTTP (Hyper Text Transfer Protocol). При записи протокола после его имени следует двоеточие и две наклонные черты: `http://`

В качестве примера запишем адрес титульной страницы Web-сайта методической службы издательства «БИНОМ. Лаборатория знаний». Страница расположена на сервере http://metodist.lbz.ru/, следовательно, адрес принимает вид:

```
http://metodist.lbz.ru
```

Доступ пользователей в Интернет осуществляют специальные организации — **провайдеры** Интернета.

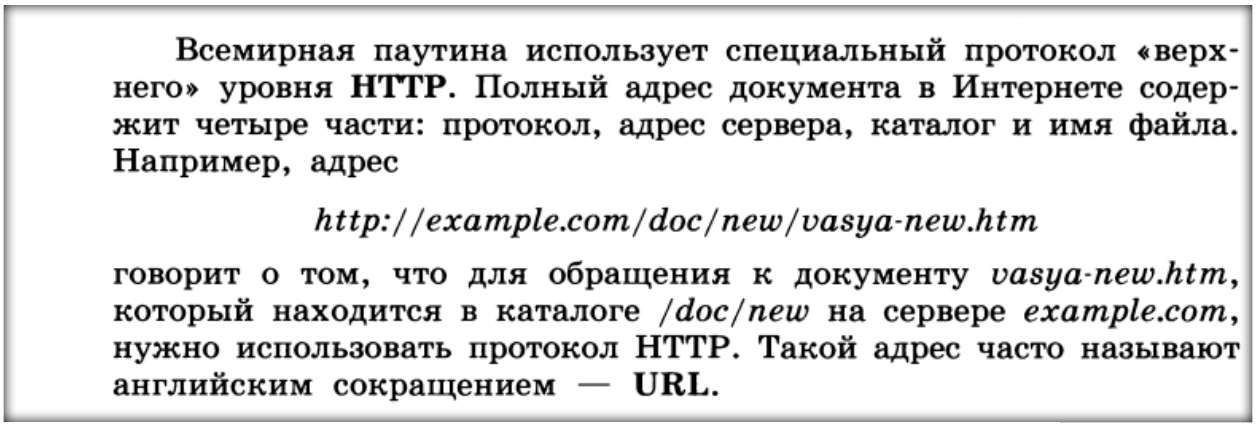
130

Угринович Н. Д.

У27 Информатика : учебник для 7 класса / Н. Д. Угринович. — 2-е изд. — М. : БИНОМ. Лаборатория знаний, 2015. — 168 с. : ил.

ISBN 978-5-9963-1824-7

Или:



Всемирная паутина использует специальный протокол «верхнего» уровня **HTTP**. Полный адрес документа в Интернете содержит четыре части: протокол, адрес сервера, каталог и имя файла. Например, адрес

http://example.com/doc/new/vasya-new.htm

говорит о том, что для обращения к документу *vasya-new.htm*, который находится в каталоге */doc/new* на сервере *example.com*, нужно использовать протокол HTTP. Такой адрес часто называют английским сокращением — **URL**.

Поляков К. Ю.
П54 Информатика. 9 класс / К. Ю. Поляков, Е. А. Еремин. — М. : БИНОМ. Лаборатория знаний, 2017. — 288 с. : ил.
ISBN 978-5-9963-3109-3

Таким образом, в школьном учебнике прошлого года (2017)
1) адрес ресурса прямо отождествляется с путём к файлу в иерархии каталогов;
2) называется сокращением URL.

В реальной практике, повторимся, ресурс определяется внутренней маршрутизацией приложения, а буквами URL чащего всего обозначается часть адреса после первого одинарного слэша включительно. Например, в адресе https://kodaktor.ru/api0/appver.html в составе URL мы находим /api0/appver.html – и в приложении нет ни каталога api0 ни файла appver.html

Кроме того, частей адреса больше – это могут быть также номер порта, строка запроса и хэш.

Устаревшее восприятие в понятийном ряду файловой системы – наследие первоначального подхода, при котором веб-сервер связывается с некой папкой и по подобным запросам ищет в ней файлы, после чего выдаёт. Сегодня это больше характерно для FTP, чем для HTTP.

Путь (URL, ресурс, машрут, эндпойнт) типа /api2/time.php рассматривается сегодня в первую очередь как команда серверу типа /сделай_это/потом_это/и_наконец_это

И сервер в этом случае становится удалённым исполнителем команд определённого формата, а не неким мостиком между желанием получить файл и каталогом, где этот файл хранится.

Мы имеем дело с динамическими ресурсами, которые генерируются автоматически по сценарию, возможно, на основе запроса к БД, или же как-то иначе.

Статический ресурс as opposed to abovesaid – это именно готовый файл. Но и для него уже не работает метафора путей в файловой системе локального компьютера. Файл с именем как в URL, возможно, существует, но его реальное расположение в файловой системе сервера не только не обязано совпадать с URL хоть в какой-то степени, но, скорее всего, наоборот, должно не совпадать – из соображений безопасности.

Выдача статических файлов таким образом превращается в проблему данного конкретного приложения: как сделать это безопасно, удобно и быстро. Следует учесть MIME-типы, выдачу соответствующих заголовков браузеру, чтобы выданный файл мог быть адекватно обработан.

Для некоторых языков свойственны встроенные средства быстрого запуска таких веб-серверов прямо в текущей папке.

```
Начиная с версии 5.4, дистрибутив PHP снабжается встроенным сервером, который позволяет отлаживать скрипты локально, без развертывания стороннего веб-сервера.

Находясь в папке со сценарием, можно запустить Development Server

php -S localhost:4321


Он отчасти похож на webpack-dev-server. Он запускается как однопоточный процесс.

  Близкий аналог - python -m SimpleHTTPServer 4321
```

Для node.js это не так характерно. Сервинг файлов является возможностью, которая реализуется как отдельная функция веб-сервера, сосредоточенного на обработка цикла запросов-ответов.

Вручную реализовывать эту функцию на сегодняшний день нет особого смысла – разве что в учебно-тренировочных целях. Она давно превратилась в подключаемую.

```
const PORT = 3336;

const serve = require('serve-static')('public', {
  'index': ['index.html', 'index.htm']
});

require('http').Server((req, res) => {
  if (req.url === '/hello') res.writeHead(200) || res.end('Hello!');
  serve(req, res, () => res.writeHead(404) || res.end('not found'));
})

.listen(PORT, () => console.log(`running ${process.pid} at port ${PORT}`));
```

https://kodaktor.ru/serve.static.vm

require('serve-static') – это функция от точки монтирования (например public) и некоторых настроек, включая индексный документ. В дальнейшем при работе этой функции то, что находится в req.url трактуется как путь относительно этой указанной точки монтирования.

Например, мы указали public – и у нас в папке приложения есть папка public, а в ней папка docs, в которой файл author.pdf. Тогда путь /docs/author.pdf как раз будет отсчитываться от этой папки-как-корня.

Функция serve здесь – это middleware, которая принимает объекты req, res и функцию next. Т.е. если файл не найден, она вызывается и тем самым управление передаётся дальше по middleware-стеку. В примере выше это функция, которая указана прямо литералом вместо ссылки, она выдаёт статус 404 и завершает соединение.

В примере выше сначала обрабатывается какой-то динамический маршрут (которых может быть сколько угодно). Если ни один из указанных клиентом маршрутов не подошёл, управление передаётся нашему middleware.

Понятие finalhandler

Поскольку при поступлении запроса на соответствие req.url проверяются последовательно идущие варианты (как бы шаблоны маршрутов), то возникает вопрос, что должно происходить, если не подошёл ни один вариант.

Довольно логично, что в этом случае нужно выдать код 404 и соответствующее сообщение. Но это требует написания кода в сыром Node, который превращается в boilerplate code.

404 – это в некотором смысле пользовательская ошибка, т.е. клиент неправильно набрал адрес или хочет неисполнимого. С точки зрения логики приложения это вообще не ошибка, а своего рода default case. А есть ещё и реальные ошибки приложения, которые должны быть обработаны.

В Express принято соглашение, что за обработку 404 отвечает предпоследний middleware, а за обработку ошибок приложения отвечает последний middleware, у которого в сигнатуре указаны 4 аргумента. Error first. (В Koa принят кардинально другой подход.)

И вот этот последний шаг в жизненном цикле обработки запроса можно обобщить, например, под названием **finalhandler**. Пусть это функция, которая принимает на вход аргумент err. Если err ложностен (falsy = false, null, undefined и т.д.), то такая функция должна сработать как default case, т.е. выдать 404. Иначе в res должно быть выдано 5xx.

Например, написать

```
require('finalhandler')(req, res)()
```

означает передать Последнему Обработчику undefined

Если раскрыть подробнее:

```
const finalhandler = require('finalhandler');

require('http').Server((req, res) => {
  const done = finalhandler(req, res);
  done();
})
.listen(1234);
```

такой веб-сервер будет всегда выдавать 404.

Пусть мы хотим, чтобы по запросу /hey осуществлялось приветствие, а во всех остальных случаях выдавалось 404.

```
const finalhandler = require('finalhandler');

require('http').Server((req, res) => {
  if (req.url === '/hey') res.end('hey');
  finalhandler(req, res)();
})
.listen(1234);
```

Отсюда следует, что выражение из примера с serve-static

```
serve(req, res, () => res.writeHead(404) || res.end('not found'));
```

эквивалентно

```
serve(req, res, finalhandler(req, res));
```

При этом в Express, Koa и других фреймворках существуют более высокоуровневые решения, приспособленные к этим системам, например, express.static – middleware, подключаемое с помощью app.use. Но лежащая вглуби идея всё равно в целом та же.