

Node.js

Библиотеки и фреймворки

Часть 2 - Express

**к.т.н. доцент каф. ВТ
Медведев М.С.**



Express.js (Express)

это минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений.

Express первоначально был выпущен в ноябре 2010 и текущая версия API 4.16.



Express предоставляет следующие механизмы:

- Написание обработчиков для запросов с различными HTTP-методами в разных URL-адресах (маршрутах).
- Интеграцию с механизмами рендеринга «view», для генерации ответов, вставляя данные в шаблоны.



Express предоставляет следующие механизмы:

- Установка общих параметров веб-приложения, такие как порт для подключения, и расположение шаблонов, которые используются для отображения ответа.
- «промежуточное ПО» для дополнительной обработки запроса в любой момент в конвейере обработки запросов.



Установка

```
$ mkdir myapp  
$ cd myapp
```

С помощью команды `npm init` создайте файл **package.json** для своего приложения.

```
$ npm init
```



package.json

Когда мы разрабатываем проект, и установили несколько npm пакетов, то они лежат в проекте в папке `node_modules`.

Рано или поздно нам необходимо запустить проект на другой машине, либо дать другому разработчику.

Но невозможно запустить проект, так как неизвестно какие пакеты из `node` были использованы.



package.json

package.json хранит список пакетов, необходимых для проекта с нужными версиями, и на другой машине мы можем легко установить все пакеты, которые указаны там с помощью команды:

```
npm install
```



Установка

Эта команда выдает ряд приглашений, например, приглашение указать имя и версию вашего приложения.

```
entry point: (index.js)
```

Введите `app.js` или любое другое имя главного файла по своему желанию. Если вас устраивает `index.js`, нажмите клавишу ВВОД, чтобы принять предложенное имя файла по умолчанию.



Установка

Теперь установим Express в каталоге app и сохраним его в списке зависимостей

```
$ npm install express --save
```

Для временной установки Express, без добавления его в список зависимостей, не указывайте опцию `--save`:



Установка

```
{  
  "name": "expressapp",  
  "version": "1.0.0",  
  "dependencies": {  
    "express": "^4.16.4"  
  }  
}
```



Для использования Express в начале надо создать объект, который будет представлять приложение:

```
const app = express(); // создаем объект приложения
```

Для обработки запросов в Express определено ряд встроенных функций, и одной из таких является функция `app.get()`.

Она обрабатывает GET-запросы протокола HTTP и позволяет связать маршруты с определенными обработчиками.



Для этого первым параметром передается *маршрут*, а вторым - *обработчик*, который будет вызываться, если запрос к серверу соответствует данному маршруту

```
app.get("/", function(request, response) {  
    // отправляем ответ  
    response.send("<h2>Привет Express!</h2>");  
});
```



Для этого первым параметром передается *маршрут*, а вторым - *обработчик*, который будет вызываться, если запрос к серверу соответствует данному маршруту

```
app.get("/", function(request, response) {  
    // отправляем ответ  
    response.send("<h2>Привет Express!</h2>");  
});
```

Маршрут "/" представляет корневой маршрут.



Создадим в каталоге проекта новый файл `app.js`:

```
// подключение
const express = require("express");
const app = express(); // создаем объект
приложения
// определяем обработчик для маршрута "/"
app.get("/", function(request, response){
  // отправляем ответ
  response.send("<h2>Привет Express!</h2>");
});
// прослушиваем подключения на 3000 порту
app.listen(3000);
```

Node.JS



Для запуска сервера вызывается метод `app.listen()`, в который передается номер порта.

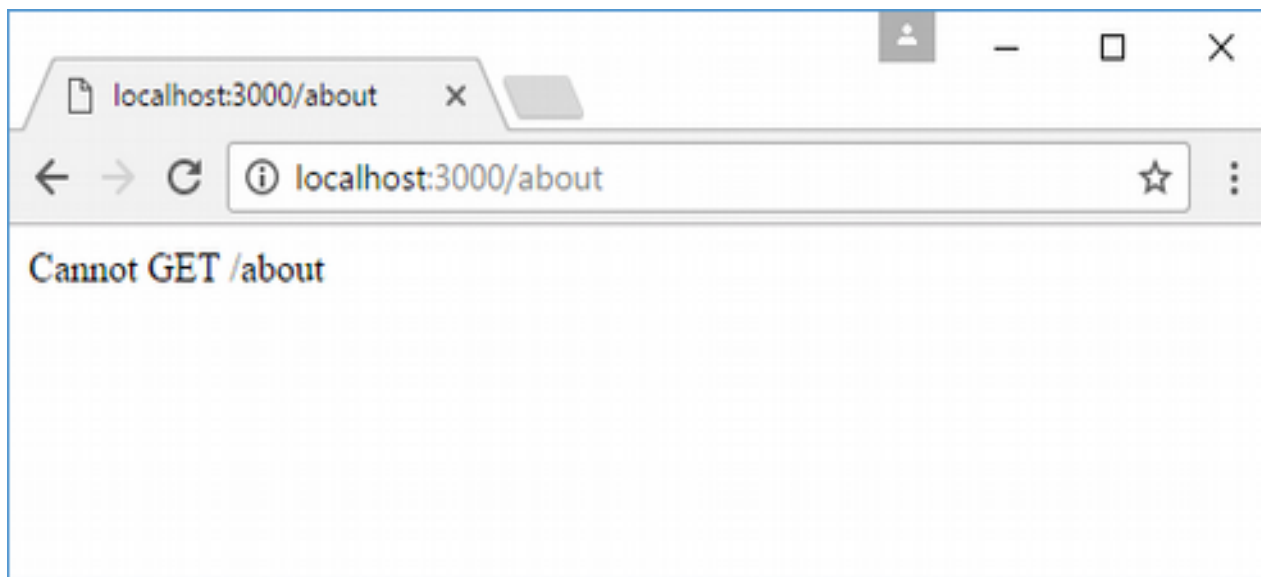
Запустим проект и обратимся в браузере по адресу `http://localhost:3000/`:



Node.JS



Express опирается на систему маршрутов, поэтому все другие запросы, которые не соответствуют корневому маршруту "/", не будут обрабатываться:



Node.JS



```
app.get("/", function(request, response) {  
    response.send("<h1>Главная страница  
</h1>"); });
```

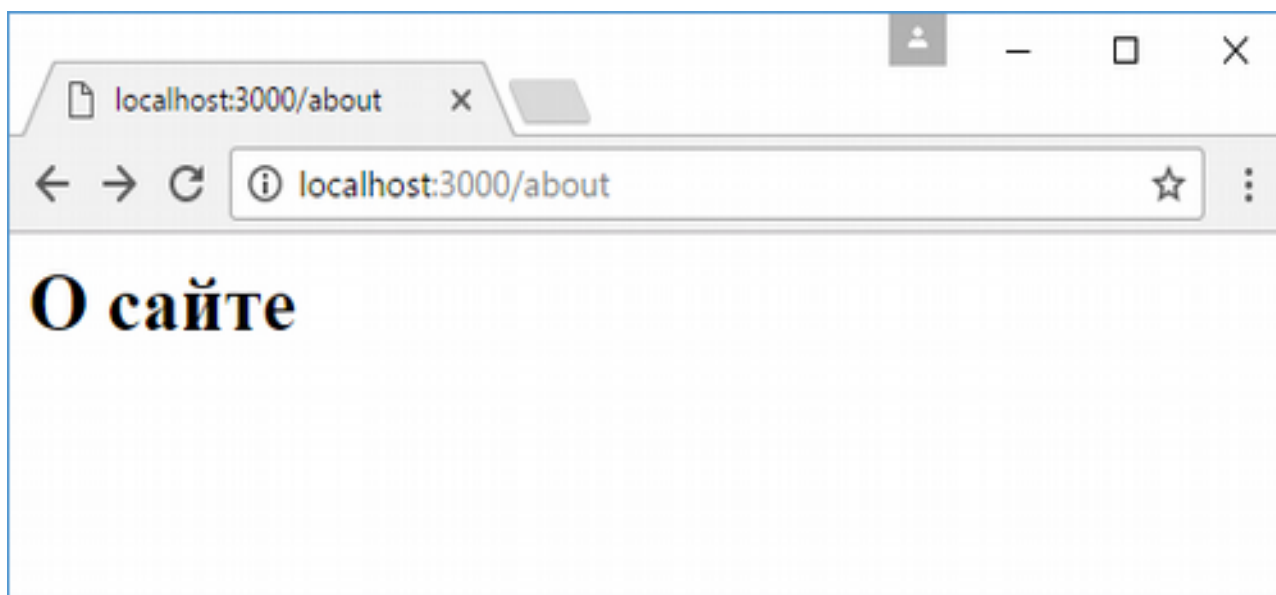
```
app.get("/about", function(request,  
response) {  
    response.send("<h1>О сайте</h1>");  
});
```

```
app.get("/contact", function(request,  
response) {  
    response.send("<h1>Контакты</h1>"); }  
);  
app.listen(3000);
```

Node.JS



Теперь в приложении определено три маршрута, которые будут обрабатываться сервером:





Конвейер обработки запроса и middleware

Когда фреймворк Express получает запрос, этот запрос передается в *конвейер обработки*.

Конвейер состоит из набора компонентов или middleware, которые получают данные запроса и решают, как его обрабатывать.



При необходимости можно встроить в конвейер обработки запроса на любом этапе любую функцию *middleware*, используя метод *app.use()*.

```
app.use(function(request, response, next) {  
    console.log("Middleware 1");  
    next();  
});
```

```
app.use(function(request, response, next) {  
    console.log("Middleware 2");  
    next();  
});
```



```
app.get("/", function(request, response) {  
  
    console.log("Route /");  
    response.send("Hello");  
});
```



Функция, которая передается в `app.use()`, принимает три параметра:

request: данные запроса

response: объект для управления ответом

next: следующая в конвейере обработки функция

Node.JS



Каждая из функций `middleware` просто выводит на консоль сообщение и в конце вызывает следующую функцию с помощью вызова `next()`.

A screenshot of a Windows command prompt window titled "Администратор: Командная строка - node app.js". The window shows the following text:

```
^C
C:\node\expressapp>node app.js
Middleware 1
Middleware 2
Route /
_
```



мы можем на каком-то этапе остановить обработку.

```
app.use(function(request, response, next) {  
    console.log("Middleware 2");  
    response.send("Middleware 2");  
});
```

A screenshot of a Windows command prompt window titled "Администратор: Командная строка - node app.js". The window shows the execution of a Node.js application. The first run shows "Middleware 1", "Middleware 2", and "Route /" before being interrupted by a Ctrl-C (^C). The second run shows "Middleware 1" and "Middleware 2" before being interrupted by a Ctrl-C (^C).

```
Администратор: Командная строка - node app.js  
C:\node\expressapp>node app.js  
Middleware 1  
Middleware 2  
Route /  
^C  
C:\node\expressapp>node app.js  
Middleware 1  
Middleware 2  
^C
```



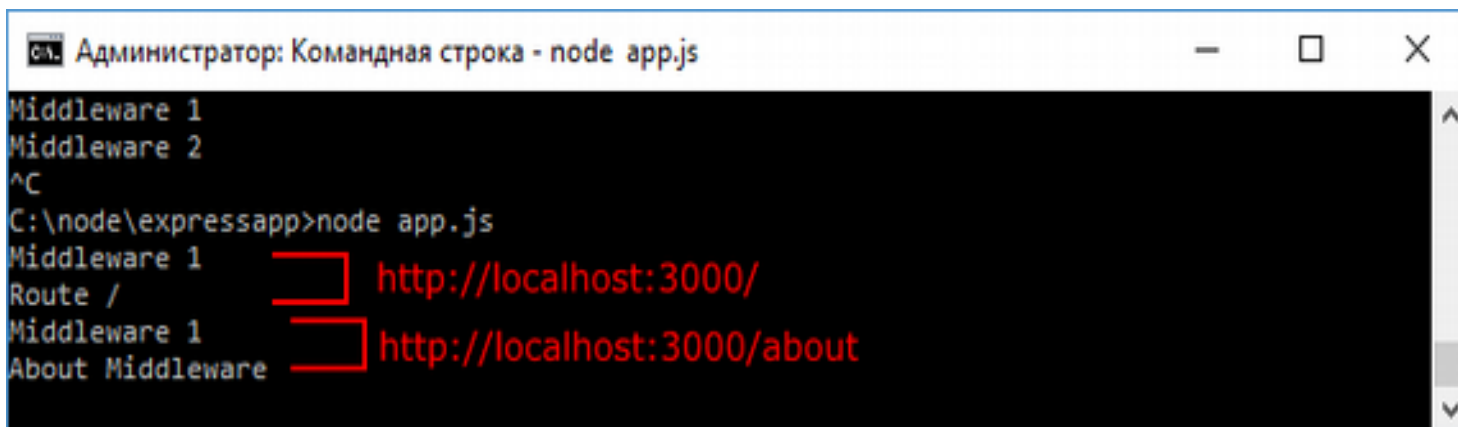

Функции middleware также могут сопоставляться с определенными маршрутами.

```
app.use("/about", function(request,  
response, next) {  
  
    console.log("About Middleware");  
    response.send("About Middleware");  
});
```

Node.JS



В данном случае вторая функция middleware явно сопоставляется с маршрутом `"/about"`, поэтому она будет обрабатывать только запрос *`"http://localhost:3000/about"`*.



```
Администратор: Командная строка - node app.js
Middleware 1
Middleware 2
^C
C:\node\expressapp>node app.js
Middleware 1      http://localhost:3000/
Route /
Middleware 1      http://localhost:3000/about
About Middleware
```



Middleware помогают выполнять некоторые задачи, которые должны быть сделаны до отправки ответа.