Node.JS
Библиотеки и фреймворки

к.т.н. доцент каф. ВТ
Медведев М.С.

Node.JS

Для отправки ответа в express у объекта response можно использовать ряд функций.

Самый распространенный способ отправки ответа представляет функция send().

В качестве параметра эта функция может принимать объект *Buffer*, строку, в том числе с html-кодом, объект *javascript* или массив.

Node.JS

Отправка объекта:

```
response.send({id:6, name: "Tom"});
```

Отправка массива:

```
response.send(["Tom", "Bob", "Sam"]);
```

Отправка Buffer:

```
response.send(Buffer.from("Hello Express"));
```

Объект Buffer формально представляет некоторые бинарные данные. Браузер загрузит файл, в котором будет строка "Hello Express".

Node.JS

Метод **send** удобен для отправки строк, некоторого кода html небольшой длины, однако есть отправляемый код html довольно большой, то соответственно код приложения тоже становится громоздким.

Гораздо лучше определять код html в отдельных файлах и затем эти файлы отправлять с помощью функции **sendFile()**.

Node.JS

```
const express = require("express");
const app = express();

app.use(function (request, response) {
  response.sendFile(__dirname +
"/index.html");
});

app.listen(3000);
```

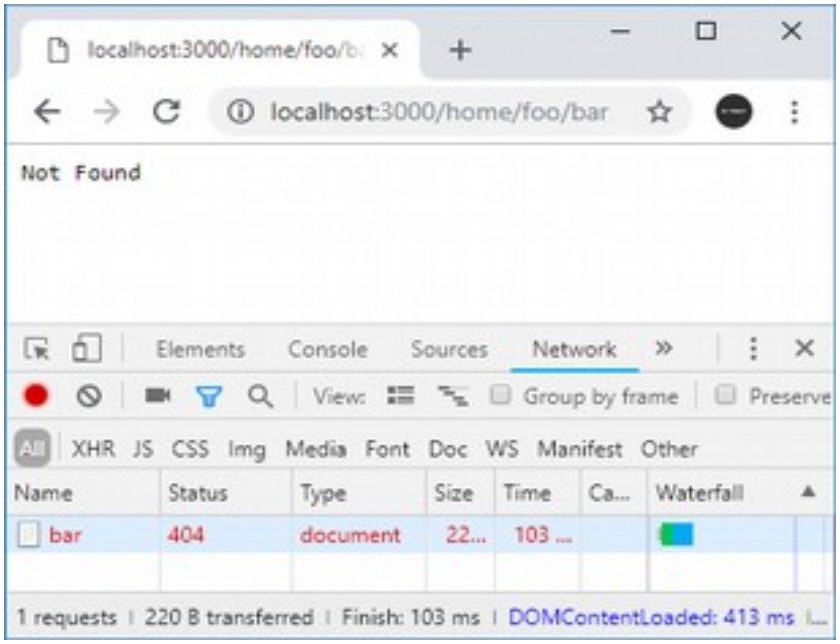
Node.JS

Функция **sendStatus()** отправляет пользователю определенный статусный код с некоторым сообщением по умолчанию.

Например, отправим статусный код 404

Node.JS

```
app.use("/home/foo/bar",function (request,
response) {
  response.sendStatus(404)
});
```



Node.JS

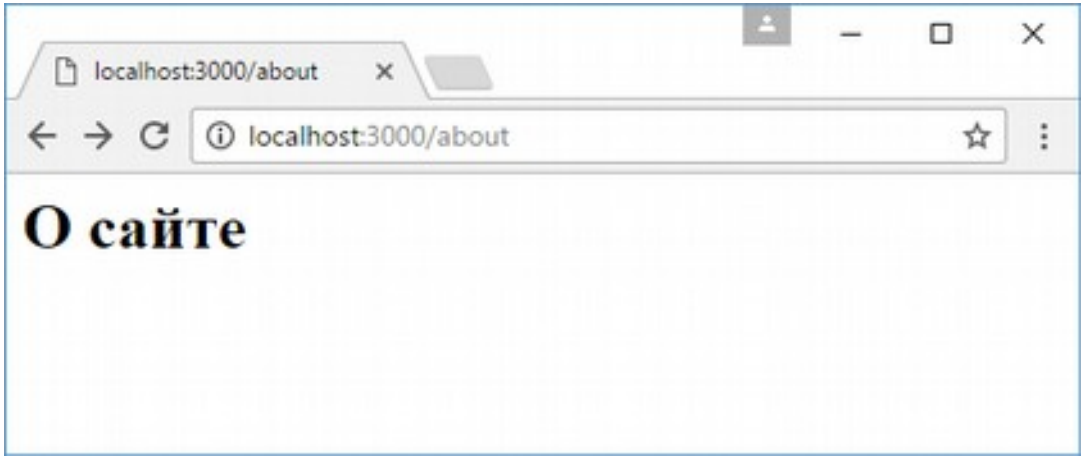
Для работы со статическими файлами в Express определен специальный компонент **express.static(),** который указывает на каталог с файлами.

Создадим для статических файлов в проекте каталог public, в который добавим новую html-страницу, которую назовем about.html

```
app.use(express.static(__dirname +
"/public"));
```

Node.JS

В саму же функцию express.static() передается путь к папке со статическими файлами. Специальное выражение __dirname позволяет получить полный путь к папке.



Node.JS

Дополнительно можно изменить путь к каталогу статических файлов

```
app.use("/static",
express.static(__dirname + "/public"));
```

Теперь чтобы обратиться к файлу about.html, необходимо отправить запрос http://localhost:3000/static/about.html.

Node.JS

Маршрутизация

При обработке запросов фреймворк Express опирается на систему маршрутизации.

В приложении определяются маршруты, а также обработчики этих маршрутов. Если запрос соответствует определенному маршруту, то вызывается для обработки запроса соответствующий обработчик.

Node.JS

Для обработки данных по определенному маршруту можно использовать ряд функций, в частности:

- *use*
- *get*
- *post*
- *put*
- *delete*

Node.JS

Маршрутизация

В качестве первого параметра эти функции могут принимать шаблон адреса, запрос по которому будет обрабатываться.

Второй параметр функций представляет функцию, которая будет обрабатывать запрос по совпавшему с шаблоном адресу.

Node.JS

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

// обработка запроса по адресу /contact
app.use("/contact", function(request,
response){
    response.send("<h1>Контакты</h1>");
});
```

Node.JS

При определении функции для обработки того или иного машрута следует учитывать, что более общие маршруты должны идти после более частных.

Сначала идут функции для обработки маршрутов "/contact" и "/about" и лишь затем функции для обработки корневного маршрута "/", поскольку маршруты "/contact" и "/about" содержат маршрут "/".

Поэтому маршрут "/" может интерпретироваться и как /contact, и как /about.

Символы подстановок

Используемые шаблоны адресов могут содержать регулярные выражения или специальные символы подстановок. В частности, мы можем использовать такие символы, как ?, +, * и ().

Пример

символ *?* указывает, что предыдущий символ может встречаться 1 раз или отсутствовать.

Node.JS

Символы подстановок

```
app.get("/bo?k", function (request,
response) {
    response.send(request.url)
});
```

Такой маршрут будет соответствовать строке запроса "/bk" или "/bok".

Символы подстановок

Символ + указывает, что предыдущий символ может встречаться 1 и более раз:

```
app.get("/bo+k", function (request,
response) {
    response.send(request.url)
});
```

Такой маршрут будет соответствовать запросам "/bok", "/book", "/boook" и так далее.

Node.JS

Символы подстановок

Символ * указывает, что на месте данного символа может находиться любое количество символов:

```
app.get("/bo*k", function (request,
response) {
    response.send(request.url)
});
```

Такой маршрут будет соответствовать запросам "/bork", "/bonk", "/bor.dak", "/bor/ok" и так далее.

Node.JS

Символы подстановок

Скобки () позволяют оформить группу символов, которые могут встречаться в запросе:

```
app.get("/book(.html)?", function
(request, response) {
    response.send(request.url)
});
```

Выражение (.html)? указывает, что подстрока ".html" может встречаться или отсутствовать. Такой маршрут будет соответствовать запросам "/book" и "/book.html".

Node.JS

Передача данных приложению. Параметры строки запроса

Одним из способов передачи данных в приложение представляет использование параметров строки запроса.

```
http://localhost:3000/about?id=3&name=Tome
```

Node.JS

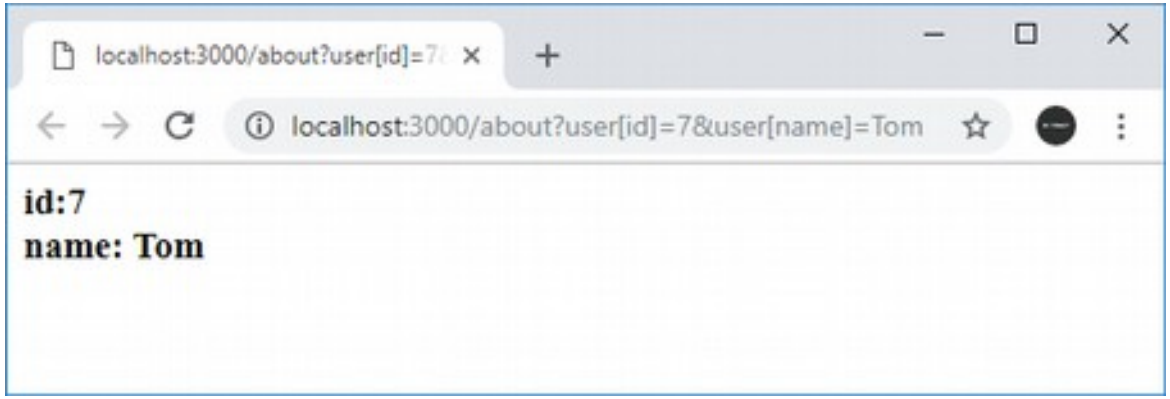
В express мы можем получить параметра строки запроса через свойство query объекта request, который передается в функцию обработки запроса.

```
app.use("/about", function(request,
response){
    let id = request.query.id;
    let userName = request.query.name;

response.send("<h1>Информация</h1><p>id="
+ id +"</p><p>name=" + userName + "</p>");
});
```

Node.JS

Можно передавать более сложые объекты, которые состоят из множества свойств:



В данном случае мы получае объект user, который содержит два свойства id и name

Node.JS

При оправке каких-то сложных данных обычно используются формы.

Для получения данных форм из запроса необходимо использовать специальный пакет *body-parser*.

```
npm install body-parser --save
```

Node.JS

Определим в папке проекта новый файл register.html:

```
<form action="/register" method="post">
  <label>Имя</label><br>
  <input type="text" name="userName" />
  <label>Возраст</label><br>
  <input type="number" name="userAge" />
  <input type="submit" value="Отправить" />
>
</form>
```

Node.JS

Поскольку данные отправляются с помощью формы, то для создания парсера применяется функция *urlencoded().*

В эту функцию передается объект, устанавливающий параметры парсинга.

Значение *extended*: *false* указывает, что объект - результат парсинга будет представлять набор пар ключ-значение, а каждое значение может быть представлено в виде строки или массива.

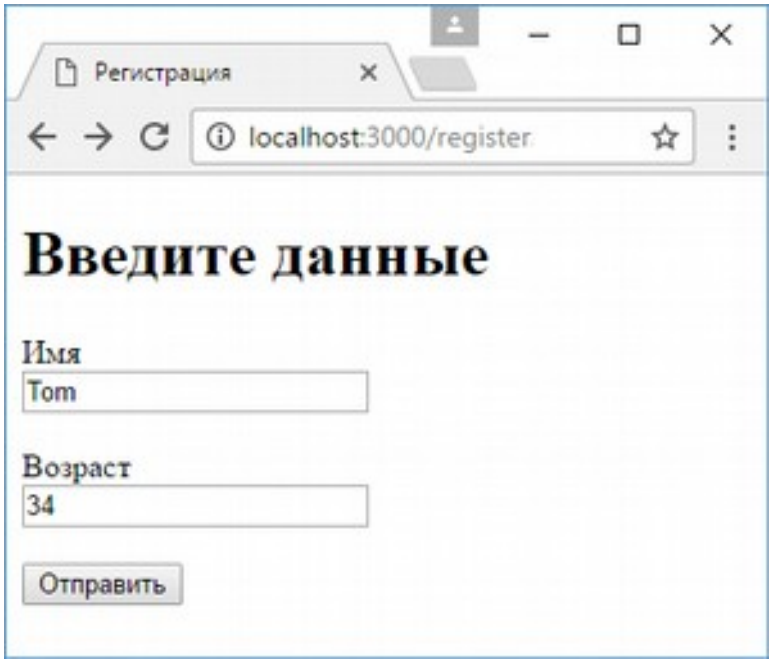
Node.JS

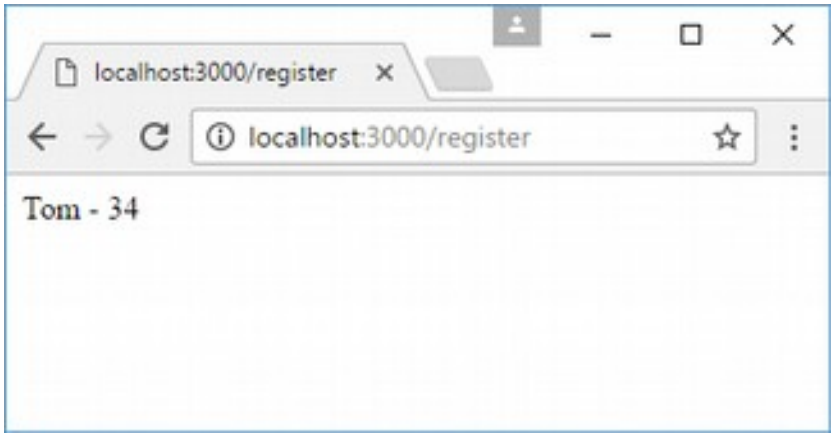
При переходе по адресу "/register" будет срабатывать метод app.get, который отправит пользователю файл *register.html*.

Для получения самих отправленных данных используем выражения типа *request.body.userName,*

где *request.body* инкапсулирует данные формы, а *userName* - ключ данных, который соответствует значению атрибута name поля ввода на html-странице.

Node.JS


Регистрация
localhost:3000/register
Введите данные
Имя
Tom
Возраст
34
Отправить


localhost:3000/register
localhost:3000/register
Tom - 34

Node.JS

Router позволяет определить дочерние подмаршруты со своими обработчиками относительно некоторого главного маршрута. Например:

```
app.use("/about", function (request,
response) {
  response.send("О сайте");
});
app.use("/products/create",function
(request, response) {
  response.send("Добавление товара");
});
```

Node.JS

```
app.use("/products/:id",function (request,
response) {
   response.send(`Товар
$request.params.id}`);
});

app.use("/products/",function (request,
response) {
   response.send("Список товаров");
});
```

Node.JS

Объект Router позволяет связать подобный функционал в одно целое и упростить управление им.

```
// определяем Router
const productRouter = express.Router();
```

Объект productRouter обрабатывает все запросы по маршруту "/products". Это главный маршрут. Однако в рамках этого маршрута может быть подмаршрут "/" со своим обработчиком, а также подмаршруты "/:id" и "/create", которые также имеют свои обработчики.

Node.JS

JSON представляет один из самых популярных форматов хранения и передачи данных, и Express имеет все возможности для работы с JSON.

Пусть в папке проекта имеется файл index.html со следующим кодом (json_ex.html).

Node.JS

Определена форма с двумя полями для ввода имени и возраста пользователя. Но теперь с помощью обработчика перехватывается отправка этой формы.

Мы получаем значения ее полей и сериализум в объект json, который затем отправляется на сервер с помощью ajax на адрес "/user".

Node.JS

Для получения данных в формате json необходимо создать парсер с помощью функции json:

```
const jsonParser = express.json();
```

поскольку с клиентом мы взаимодействуем через формат json, то данные клиенту отправляются с помощью метода response.json().

Node.JS

```
app.post("/user", jsonParser, function
(request, response) {
    console.log(request.body);
    if(!request.body) return
response.sendStatus(400);

    response.json(request.body); // отправляем
пришедший ответ обратно
});
```

Node.JS

```
app.get("/", function(request, response){

    response.sendFile(__dirname +
"/index.html");
});

app.listen(3000);
```

Node.JS

В реальности метод response.json() устанавливает для заголовка "Content-Type" значение "application/json", серилизует данные в json с помощью функции JSON.stringify() и затем отправляет данные с помощью response.send().

Для получения данных, как и в случае с формами, используются выражения типа request.body.userName, где request.body инкапсулирует данные формы, а userName - ключ данных.

Node.JS

При обращении к корню веб-приложения пользователю будет отправляться содержимое файла index.html с формой ввода данных.