**Тема 16. Фреймворк Express.js (Express)**

1. **Начало работы с Express.**
2. **Конвейер обработки запроса и middleware.**
3. **Отправка ответа.**
4. **Статические файлы.**
5. **Маршрутизация.**
6. **Переадресация.**
7. **Передача данных приложению.**
8. **POST-запросы и отправка форм.**
9. **Параметры маршрута.**
10. **Router. JSON и AJAX.**

Содержание данной темы включает материалы, доступные по адресу https://metanit.com.

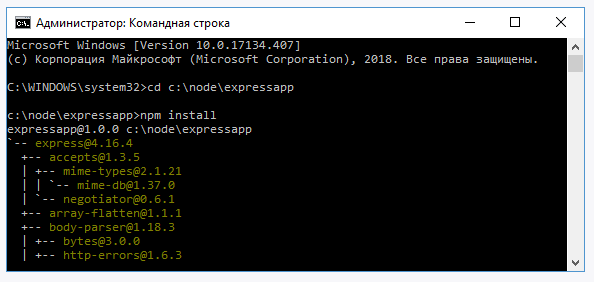
1. Начало работы с Express.

Рассмотрим создание сервера с помощью фреймворка Express. Express использует модуль http, но вместе с тем предоставляет ряд готовых абстракций, которые упрощают создание сервера и серверной логики, в частности, обработка отправленных форм, работа с куками, CORS и т.д.

Создадим для проекта новый каталог, например, expressapp. Для хранения информации обо всех зависимостях проекта определим в этом каталоге новый файл package.json:

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

Далее перейдем к этому каталогу в командной строке/терминале и для добавления всех нужных пакетов выполним команду: npm install.



Создадим в каталоге проекта новый файл app.js:

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

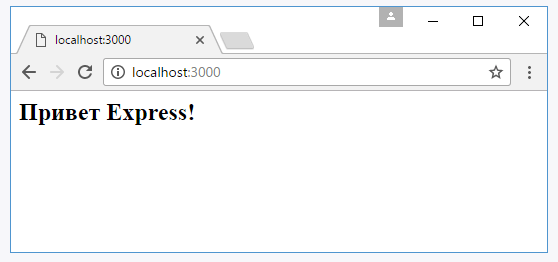
Для использования Express в начале надо создать объект, который будет представлять приложение:

|  |  |
| --- | --- |
|  | const app = express(); |

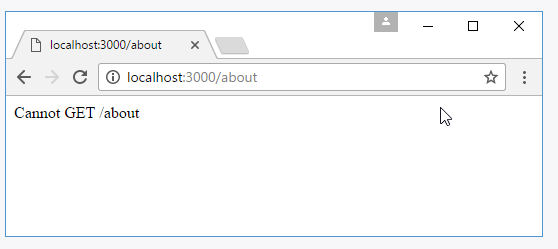
Для обработки запросов в Express определено ряд встроенных функций, и одной из таких является функция app.get(). Она обрабатывает GET-запросы протокола HTTP и позволяет связать маршруты с определенными обработчиками. Для этого первым параметром передается маршрут, а вторым – обработчик, который будет вызываться, если запрос к серверу соответствует данному маршруту:

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

Маршрут "/" представляет корневой маршрут. Для запуска сервера вызывается метод app.listen(), в который передается номер порта. Запустим проект и обратимся в браузере по адресу <http://localhost:3000/>:



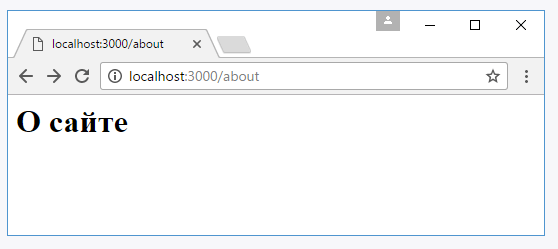
И что важно, Express опирается на систему маршрутов, поэтому все другие запросы, которые не соответствуют корневому маршруту "/", не будут обрабатываться:



Теперь изменим файл app.js:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  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); |

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



1. 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.

Так, в предыдущей теме файл приложения выглядел следующим образом:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  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); |

Здесь конвейер обработки состоял из вызовов app.get(), которые сравнивали запрошенный адрес с маршрутом, и если между адресом и маршрутом было соответствие, то данный запрос обрабатывался методом app.get().

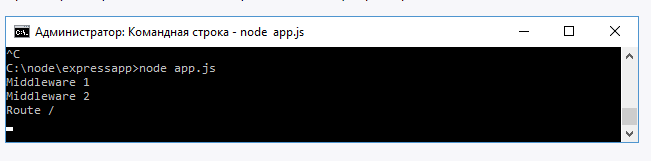
При необходимости можно встроить в конвейер обработки запроса на любом этапе любую функцию middleware. Для этого применяется метод app.use(). Так, изменим файл app.js следующим образом:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  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");  });  app.listen(3000); |

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

* request: данные запроса;
* response: объект для управления ответом;
* next: следующая в конвейере обработки функция.

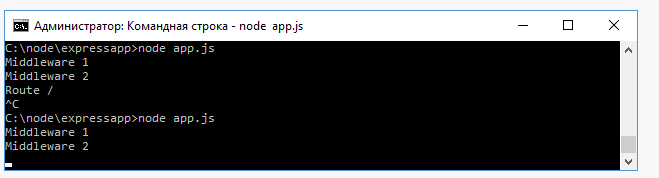
Каждая из функций middleware просто выводит на консоль сообщение и в конце вызывает следующую функцию с помощью вызова next(). При запуске приложения после обращения по адресу "http://localhost:3000/" последовательно отработают все три middleware:



Однако необязательно вызывать все последующие middleware, можно на каком-то этапе остановить обработку:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  app.use(function(request, response, next){        console.log("Middleware 1");      next();  });  app.use(function(request, response, next){        console.log("Middleware 2");      response.send("Middleware 2");  });    app.get("/", function(request, response){      console.log("Route /");      response.send("Hello");  });  app.listen(3000); |

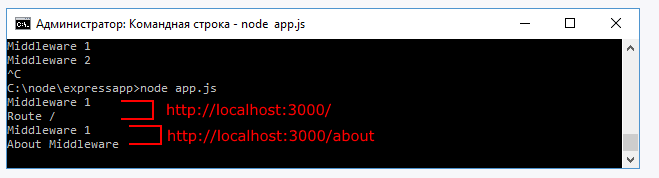
Теперь обработка завершается на Middleware 2, так как в этом методе происходит отправка ответа с помощью response.send(), а вызова следующей функции через next():



Функции middleware также могут сопоставляться с определенными маршрутами. Например:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  app.use(function(request, response, next){        console.log("Middleware 1");      next();  });  app.use("/about", function(request, response, next){        console.log("About Middleware");      response.send("About Middleware");  });    app.get("/", function(request, response){      console.log("Route /");      response.send("Hello");  });  app.listen(3000); |

В данном случае вторая функция middleware явно сопоставляется с маршрутом "/about", поэтому она будет обрабатывать только запрос "http://localhost:3000/about". Первая функция middleware по прежнему обрабатывает все запросы:



**Пример middleware**

Middleware помогают выполнять некоторые задачи, которые должны быть сделаны до отправки ответа. Стандартная задача – логгирование запросов.

Например, изменим файл app.js следующим образом:

|  |  |
| --- | --- |
|  | const express = require("express");  const fs = require("fs");    const app = express();  app.use(function(request, response, next){        let now = new Date();      let hour = now.getHours();      let minutes = now.getMinutes();      let seconds = now.getSeconds();      let data = `${hour}:${minutes}:${seconds} ${request.method} ${request.url} ${request.get("user-agent")}`;      console.log(data);      fs.appendFile("server.log", data + "\n", function(){});      next();  });    app.get("/", function(request, response){      response.send("Hello");  });  app.listen(3000); |

Здесь с помощью объекта request получаем различную информацию о запросе и добавляем ее в файл server.log, используя модуль fs.

1. Отправка ответа.

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.

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use(function (request, response) {      response.send("<h2>Hello</h2>");  });    app.listen(3000); |

Отправка объекта:

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

Отправка массива:

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

Отправка объекта:

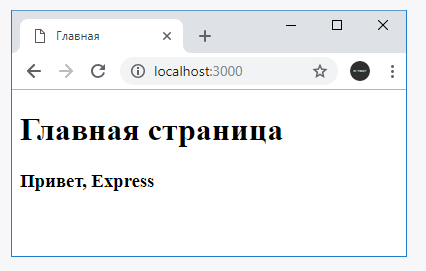
|  |  |
| --- | --- |
|  | response.send(Buffer.from("Hello Express")); |

Объект Buffer формально представляет некоторые бинарные данные. Так, в случае выше при выполнении кода по умолчанию браузер загрузит файл, в котором будет строка "Hello Express".

**Метод sendFile**

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. Например, можно написать так:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use(function (request, response) {    response.send(`<!DOCTYPE html>    <html>    <head>        <title>Главная</title>        <meta charset="utf-8" />    </head>    <body>        <h1>Главная страница</h1>        <h3>Привет, Express</h3>    </body>    <html>`);  });    app.listen(3000); |



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

Например, определим в папке проекта новый файл index.html:

|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>    <head>        <title>Главная</title>        <meta charset="utf-8" />    </head>    <body>        <h1>Главная страница</h1>        <h3>Привет, Express</h3>    </body>  <html> |

Отправим этот файл с помощью функции sendFile:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use(function (request, response) {    response.sendFile(\_\_dirname + "/index.html");  });    app.listen(3000); |

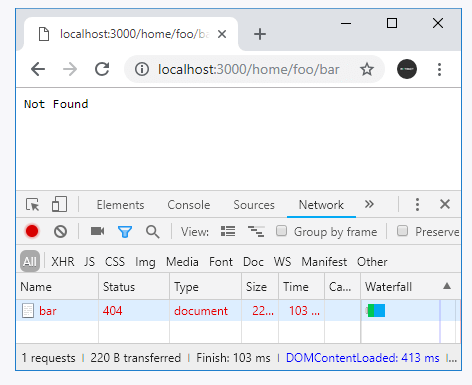
В итоге получим тот же самый результат.

Следует учитывать, что в функцию sendFile необходимо передавать абсолютный путь к файлу, именно для этого с помощью \_\_dirname получаем абсолютный путь к текущему проекту и затем добавляем к нему путь к файлу в рамках текущего проекта.

**Отправка статусных кодов**

Функция sendStatus() отправляет пользователю определенный статусный код с некоторым сообщением по умолчанию. Например, отправим статусный код 404, который говорит, что ресурс не найден:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use("/home/foo/bar",function (request, response) {    response.sendStatus(404)  });    app.listen(3000); |



Как видно из скриншота, при отправке статусного кода 404 также отправляется сообщение "Not Found". Но, возможно, надо отправлять какие-то свои более информативные сообщения. В этом случае можно использовать комбинацию функции status(), которая также отправляет статусный код, и функции send():

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use("/home/foo/bar",function (request, response) {    response.status(404).send(`Ресурс не найден`);  });    app.listen(3000); |

1. Статические файлы.

Рассмотрим, как в Node.js + Express отправлять пользователю статические файлы html. 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 назовемabout.html:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>      <title>О сайте</title>      <meta charset="utf-8" />  </head>  <body>      <h1>О сайте</h1>  </body>  <html> |

В итоге проект будет выглядеть следующим образом:

* app.js
* node\_modules
* package.json
* public
* about.html

Теперь изменим файл app.js:

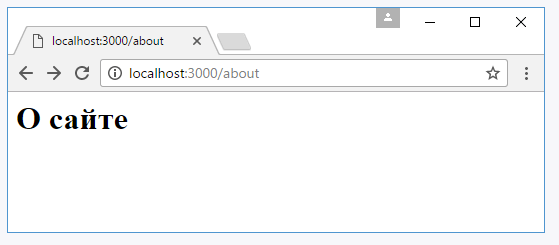
|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();    app.use(express.static(\_\_dirname + "/public"));    app.use("/", function(request, response){        response.send("<h1>Главная страница</h1>");  });    app.listen(3000); |

Чтобы встроить компонент express.static в процесс обработки запроса, вызывается функция app.use(). Эта функция позволяет добавлять различные компоненты, которые еще называются middleware, в конвейер обработки запроса:

|  |  |
| --- | --- |
|  | app.use(express.static(\_\_dirname + "/public")); |

Причем данный вызов помещается до всех остальных вызовов функции app.get().

В саму же функцию express.static() передается путь к папке со статическими файлами. Специальное выражение \_\_dirname позволяет получить полный путь к папке. Запустим приложение на выполнение и обратимся в браузере по пути "http://localhost:3000/about.html":



Следует отметить, что для обращения к файлу указываем только имя файла без названия каталога public. Также не следует путать подобное обращение к файлам с действием функции sendFile: в данном случае напрямую обращаемся к статическим файлам, а функция sendFile фактически берет содержимое из файла и отсылает его пользователю. Дополнительно можно изменить путь к каталогу статических файлов:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();    app.use("/static", express.static(\_\_dirname + "/public"));    app.use("/", function(request, response){        response.send("<h1>Главная страница</h1>");  });    app.listen(3000); |

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

1. 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. Для обработки данных по определенному маршруту можно использовать ряд функций, в частности: use, get, post, put, delete.

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. Например:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();   // 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>");  });   // обработка запроса к корню веб-сайта  app.get("/", function(request, response){           response.send("<h1>Главная страница</h1>");  });  app.listen(3000); |

Когда приходит запрос Express сопоставляет запрошенный адрес с каждым из маршрутов. Затем выбирается первый совпавший маршрут. При совпадении маршрута вызывается его функция обработчика.

Но при определении функции для обработки того или иного машрута следует учитывать, что более общие маршруты должны идти после более частных. Так, в примере выше 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 node.js маршрут "/" может интерпретировть и как /contact, и как /about.

**Символы подстановок**

Используемые шаблоны адресов могут содержать регулярные выражения или специальные символы подстановок. В частности, можно использовать такие символы, как ?, +, \* и (). Например, символ ? указывает, что предыдущий символ может встречаться 1 раз или отсутствовать. И можно определить следующую функцию:

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

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

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. Символ звездочка \* 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.

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 в строке запроса. И такой маршрут будет соответствовать запросам "/book" и "/book.html".

Также вместо определения маршрутов можно указывать регулярные выражения. Например, необходимо перехватить запрос ко всем файлам html или всем путям, которые в конце имеют ".html":

|  |  |
| --- | --- |
|  | app.get(/.\*(\.)html$/, function (request, response) {      response.send(request.url)  }); |

1. Переадресация.

Для переадресации применяется метод redirect():

|  |  |
| --- | --- |
|  | redirect([status,] path) |

В качестве параметра path передается путь, на который будет перенаправляться пользователь. Дополнительный параметр status задает статусный код переадресации. Если этот параметр не задан, тогда по умолчанию отправляется статусный код 302, который предполагает временную переадресацию.

С помощью данного метода можно выполнять переадресацию как по относительным путям, так и по абсолютным, в том числе на другие домены.

Переадресация по абсолютному пути:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use("/index",function (request, response) {    response.redirect("[https://metanit.com](https://metanit.com/)")  });    app.listen(3000); |

В данном случае при обращении по пути "/index" будет идти переадресация на сайт https://metanit.com.

Переадресация по оносительным путям также не очень сложна, но здесь важно учитывать, как именно определяется адрес для редиректа. Рассмотрим редирект относительно текущего пути, с которого производится редирект. Например:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use("/home",function (request, response) {    response.redirect("about")  });  app.use("/about", function (request, response) {    response.send("<h1>About</h1>");  });    app.listen(3000); |

В данном случае будет идти перенаправление с ресурса "/home" на ресурс "/about", то есть, условно говоря, с http://localhost:3000/home на http://localhost:3000/about.

Теперь рассмотрим похожий пример:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use("/home/bar",function (request, response) {    response.redirect("about")  });  app.use("/home/about", function (request, response) {    response.send("<h1>About</h1>");  });    app.listen(3000); |

Здесь с ресурса "/home/bar" также идет переадресация на ресурс "about", однако в реальности теперь это будет не http://localhost:3000/about, а http://localhost:3000/home/about. То есть поднимаемся на один уровень вверх – с "home/bar" на "home/" и затем к нему добавляется "about".

Если необходимо выполнить переадресацию не относительно текущего ресурса, а относительно корневого каталога приложения, то в начале адреса ставится слеш:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use("/home/bar",function (request, response) {    response.redirect("/about")  });  app.use("/about", function (request, response) {    response.send("<h1>About</h1>");  });    app.listen(3000); |

Еще несколько примеров. Переадресация относительно текущего адреса на адрес на том же уровне:

|  |  |
| --- | --- |
|  | app.use("/home/foo/bar",function (request, response) {    response.redirect("./about")  }); |

Здесь переадресация с http://localhost:3000/home/foo/bar на <http://localhost:3000/home/foo/about>.

Переадресация на адрес, который располагается уровнем выше:

|  |  |
| --- | --- |
|  | app.use("/home/foo/bar",function (request, response) {    response.redirect("../about")  }); |

Здесь переадресация с http://localhost:3000/home/foo/bar на http://localhost:3000/home/about.

Переадресация на уровень выше:

|  |  |
| --- | --- |
|  | app.use("/home/foo/bar",function (request, response) {    response.redirect(".")  }); |

Здесь переадресация с http://localhost:3000/home/foo/bar на http://localhost:3000/home/foo.

Переадресация на два уровня выше:

|  |  |
| --- | --- |
|  | app.use("/home/foo/bar",function (request, response) {    response.redirect("..")  }); |

Здесь переадресация с http://localhost:3000/home/foo/bar на http://localhost:3000/home.

По умолчанию при редиректе передается статусный код 302, который указывает, что ресурс временно доступен по новому адресу. Но можно указать статусный код 301, чтобы сделать переадресацию постоянной:

|  |  |
| --- | --- |
|  | response.redirect(301, "/about"); |

1. Передача данных приложению.

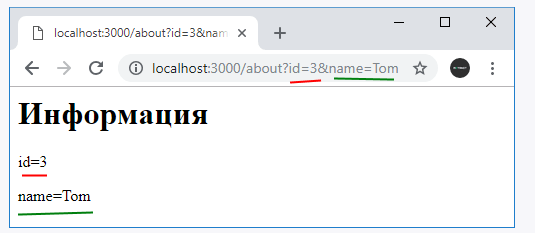
Одним из способов передачи данных в приложение представляет использование параметров строки запроса. Строка запроса (query) – фактически это часть запрошенного адреса, которая идет после знака вопроса. Например, в запросе http://localhost:3000/about?id=3&name=Tome часть id=3&name=Tome представляет строку запроса.

Строку запроса образуют параметры. После названия каждого параметра после знака равно (=) идет его значение. Друг от друга параметры отделяются знаком амперсанда. Например, в адресе выше использовалось два параметра: параметр "id" имеет значение "3" и параметр "name" имеет значение "Tom".

В express можно получить параметры строки запроса через свойство query объекта request, который передается в функцию обработки запроса. Например:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  app.get("/", function(request, response){        response.send("<h1>Главная страница</h1>");  });  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>");  });    app.listen(3000); |

С помощью выражения request.query можно получить все параметры строки запрос в виде объекта javascript, а с помощью выражения request.query.название\_параметра  можно обратиться к каждому отдельному параметру.

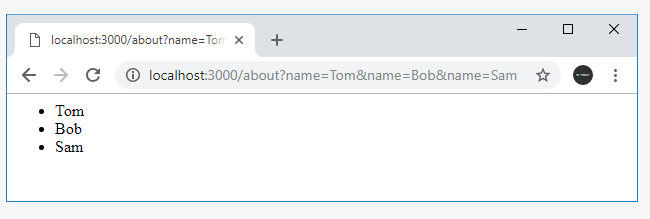


**Передача массивов**

Подобным образом можно передавать массивы данных:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  app.get("/", function(request, response){        response.send("<h1>Главная страница</h1>");  });  app.use("/about", function(request, response){        console.log(request.query);      let names = request.query.name;      let responseText = "<ul>";      for(let i=0; i < names.length; i++){          responseText += "<li>" + names[i] + "</li>";      }      responseText += "</ul>";      response.send(responseText);  });    app.listen(3000); |

В данном случае в приложение будет передаваться параметр name, который представляет массив.

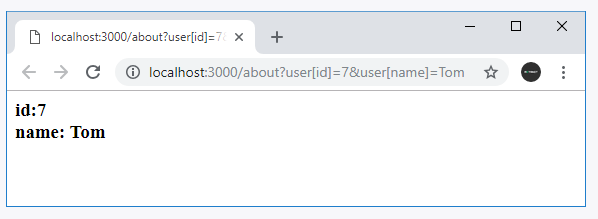


**Передача сложных объектов**

Также можно передавать более сложные объекты, которые состоят из множества свойств:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();    app.use("/about", function(request, response){        console.log(request.query);      let id = request.query.user.id;      let name = request.query.user.name;        response.send("<h3>id:" + id + "<br>name: " + name +"</h3>");  });    app.listen(3000); |

В данном случае получаем объект user, который содержит два свойства id и name, например, user: { id: '7', name: 'Tom' }.



При передаче в строке запроса свойства объекта помещаются в квадратные скобки: user[id].

1. POST-запросы и отправка форм.

При оправке каких-то сложных данных обычно используются формы. Рассмотрим, как получать отправленные данные в Express. Для получения данных форм из запроса необходимо использовать специальный пакет [body-parser](https://www.npmjs.com/package/body-parser). Поэтому вначале добавим его в проект с помощью команды

npm install body-parser --save

Определим в папке проекта новый файл register.html:

|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>  <head>      <title>Регистрация</title>      <meta charset="utf-8" />  </head>  <body>      <h1>Введите данные</h1>      <form action="/register" method="post">          <label>Имя</label><br>          <input type="text" name="userName" /><br><br>          <label>Возраст</label><br>          <input type="number" name="userAge" /><br><br>          <input type="submit" value="Отправить" />      </form>  </body>  </html> |

Здесь определены два поля для ввода имени и возраста пользователя. После нажатия на кнопку данные будут уходить по адресу "/register".

В файле app.js определим следующий код:

|  |  |
| --- | --- |
|  | const express = require("express");  const bodyParser = require("body-parser");    const app = express();    // создаем парсер для данных application/x-www-form-urlencoded  const urlencodedParser = bodyParser.urlencoded({extended: false});    app.get("/register", urlencodedParser, function (request, response) {      response.sendFile(\_\_dirname + "/register.html");  });  app.post("/register", urlencodedParser, function (request, response) {      if(!request.body) return response.sendStatus(400);      console.log(request.body);      response.send(`${request.body.userName} - ${request.body.userAge}`);  });    app.get("/", function(request, response){      response.send("Главная страница");  });    app.listen(3000); |

Прежде всего для получения отправленных данных необходимо создать парсер:

|  |  |
| --- | --- |
|  | const urlencodedParser = bodyParser.urlencoded({extended: false}); |

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

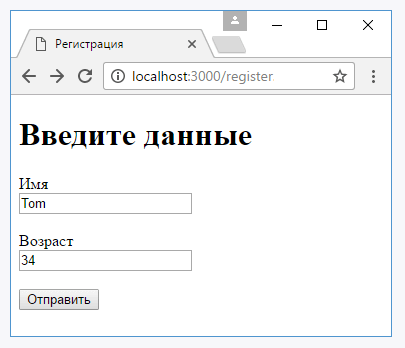
Так как данные отправляются с помощью метода POST, то для обработки определяем функцию app.post("/register",...). Первый параметр функции – адрес, на который идет отправка – "/register". Стоит отметить, что в данном случае с одним адресом "/register" связаны две функции, только одна обрабатывает запросы get, а другая – запросы post. Второй параметр – выше созданный парсер. Третий параметр – обработчик:

|  |  |
| --- | --- |
|  | app.post("/register", urlencodedParser, function (request, response) {      if(!request.body) return response.sendStatus(400);      console.log(request.body);      response.send(`${request.body.userName} - ${request.body.userAge}`);  }); |

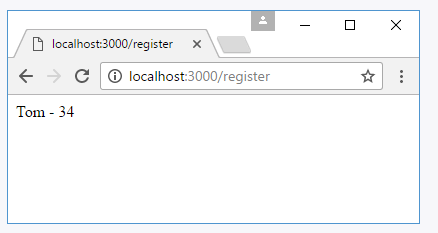
Для получения самих отправленных данных используем выражения типа request.body.userName, где request.body инкапсулирует данные формы, а userName – ключ данных, который соответствует значению атрибута name поля ввода на html-странице:

|  |  |
| --- | --- |
|  | <input type="text" name="userName" /> |

Запустим приложение и обратимся к странице register.html:



После отправки введенные данные будут получены в функции app.post("/register",...), которая выведет результат обработки:



1. Параметры маршрута.

Параметры маршрута представляют именованные сегменты URL-адреса. Не стоит их путать с параметрами строки запроса. Например:

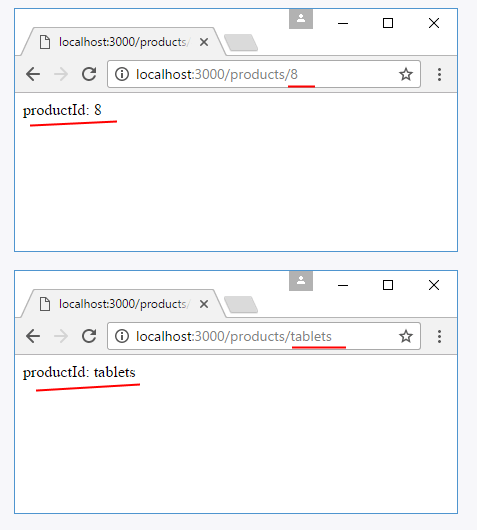
|  |  |
| --- | --- |
|  | localhost:3000/about/user?id=3&name=Tome |

Здесь параметры строки запроса – это то, что идет после вопросительного знака – id=3&name=Tome. Остальная часть, которая идет до вопросительного знака может содержать параметры маршрута.

Название параметра должно включать символы из диапазона [A-Za-z0-9\_]. В определении маршрута параметры предваряются знаком двоеточия:

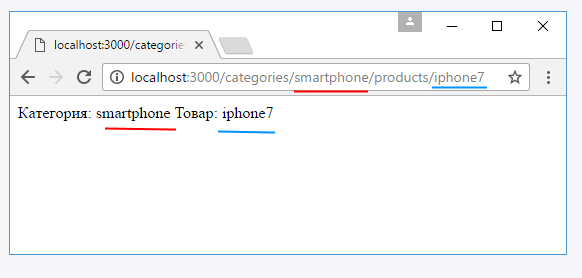
|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.get("/products/:productId", function (request, response) {    response.send("productId: " + request.params["productId"])  });    app.listen(3000); |

В данном случае параметр называется "productId". Через коллекцию request.params можно получить все параметры и, в частности, значение параметра productId. Если потребуется передать для этого параметра значение, то оно указывается в качестве последнего сегмента в строке запроса:



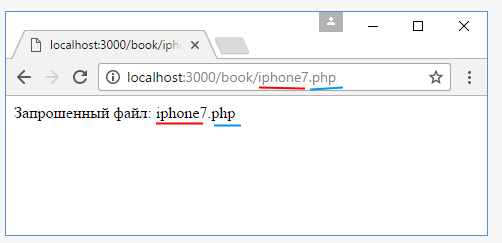
Можно использовать более сложные комбинации параметров:

|  |  |
| --- | --- |
|  | app.get("/categories/:categoryId/products/:productId", function (request, response) {      let catId = request.params["categoryId"];      let prodId = request.params["productId"];      response.send(`Категория: ${catId}  Товар: ${prodId}`);  }); |



Еще пример:

|  |  |
| --- | --- |
|  | app.get("/book/:pageName.:pageExt", function (request, response) {      let pageName = request.params["pageName"];      let pageExt = request.params["pageExt"];      response.send(`Запрошенный файл: ${pageName}.${pageExt}`);  }); |



1. Router. JSON и AJAX.

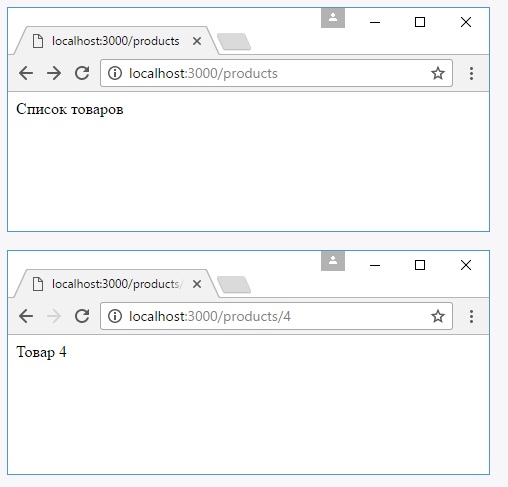
Router позволяет определить дочерние подмаршруты со своими обработчиками относительно некоторого главного маршрута. Например, определим следующее приложение:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    app.use("/about", function (request, response) {    response.send("О сайте");  });    app.use("/products/create",function (request, response) {    response.send("Добавление товара");  });  app.use("/products/:id",function (request, response) {    response.send(`Товар ${request.params.id}`);  });  app.use("/products/",function (request, response) {    response.send("Список товаров");  });    app.use("/", function (request, response) {    response.send("Главная страница");  });  app.listen(3000); |

Здесь есть пять маршрутов, которые обрабатываются различными обработчиками. Но три из этих маршрутов начинаются с "/products" и условно относятся к некоторому функционалу по работе с товарами (просмотр списка товаров, просмотр одного товара по id и добавление товара). Объект Router позволяет связать подобный функционал в одно целое и упростить управление им. Например, перепишем предыдущий пример с использованием объекта Router:

|  |  |
| --- | --- |
|  | const express = require("express");  const app = express();    // определяем Router  const productRouter = express.Router();    // определяем маршруты и их обработчики внутри роутера  productRouter.use("/create", function(request, response){    response.send("Добавление товара");  });  productRouter.use("/:id", function(request, response){    response.send(`Товар ${request.params.id}`);  });  productRouter.use("/", function(request, response){    response.send("Список товаров");  });  // сопотавляем роутер с конечной точкой "/products"  app.use("/products", productRouter);    app.use("/about", function (request, response) {    response.send("О сайте");  });    app.use("/", function (request, response) {    response.send("Главная страница");  });  app.listen(3000); |

Здесь определен объект productRouter, который 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.



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:

|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>  <head>      <title>Регистрация</title>      <meta charset="utf-8" />  </head>  <body>      <h1>Введите данные</h1>      <form name="registerForm">          <label>Имя</label></br>          <input type="text" name="userName" /></br></br>          <label>Возраст</label></br>          <input type="number" name="userAge" /></br></br>          <button type="submit" id="submit">Отправить</button>      </form>      <script>            document.getElementById("submit").addEventListener("click", function (e) {               e.preventDefault();              // получаем данные формы              let registerForm = document.forms["registerForm"];              let userName = registerForm.elements["userName"].value;              let userAge = registerForm.elements["userAge"].value;              // сериализуем данные в json              let user = JSON.stringify({userName: userName, userAge: userAge});              let request = new XMLHttpRequest();              // посылаем запрос на адрес "/user"               request.open("POST", "/user", true);               request.setRequestHeader("Content-Type", "application/json");               request.addEventListener("load", function () {                  // получаем и парсим ответ сервера                   let receivedUser = JSON.parse(request.response);                   console.log(receivedUser.userName, "-", receivedUser.userAge);   // смотрим ответ сервера               });               request.send(user);           });      </script>  </body>  <html> |

Здесь, как и в прошлой теме, определена форма с двумя полями для ввода имени и возраста пользователя. Но теперь с помощью обработчика нажатия перехватывается отправка этой формы. Получаем значения ее полей и сериализум в объект json, который затем отправляется на сервер с помощью ajax на адрес "/user". Далее определим в главном файле приложения – в app.js код, который бы принимал отправленные данные:

|  |  |
| --- | --- |
|  | const express = require("express");    const app = express();  // создаем парсер для данных в формате json  const jsonParser = express.json();    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  });    app.get("/", function(request, response){        response.sendFile(\_\_dirname + "/index.html");  });    app.listen(3000); |

Прежде всего для получения данных в формате json необходимо создать парсер с помощью функции json:

|  |  |
| --- | --- |
|  | const jsonParser = express.json(); |

В реальности данный парсер будет использовать модуль bodyParser, который использовался нами ранее для парсинга данных отправленной формы.

И поскольку с клиентом взаимодействие осуществляется через формат json, то данные клиенту отправляются с помощью метода response.json():

|  |  |
| --- | --- |
|  | 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  }); |

Следует отметить, что в принципе можно отправить объект и с помощью стандартного метода response.send(request.body). 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. При обращении к корню веб-приложения пользователю будет отправляться содержимое файла index.html с формой ввода данных.

Запустим приложение и обратимся к корню веб-сайта. Введем какие-нибудь данные и после отправки в консоли браузера отобразится ответ сервера:

