

Идентификаторы ресурсов URI, URL, URN

Расшифровка аббревиатур:

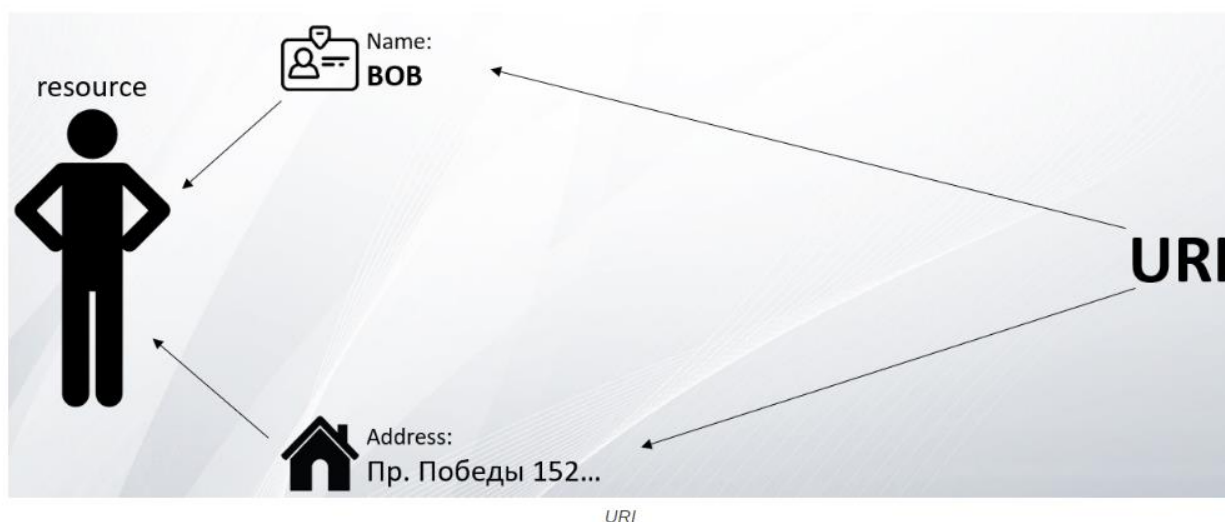
URI - Uniform Resource Identifier (унифицированный идентификатор ресурса)

URL - Uniform Resource Locator (унифицированный определитель местонахождения ресурса)

URN - Uniform Resource Name (унифицированное имя ресурса)

URI (Uniform Resource Identifier) – это строка символов, которая используется для идентификации какого-либо ресурса по его адресу или по его имени, либо по тому и тому вместе.

Чтобы стало понятнее проведем аналогию с реальным миром на примере какого-нибудь человека. У человека есть имя, например, Боб. Также у человека есть адрес проживания, например, пр. Победы 152. Предположим, нам нужно найти человека. Мы можем это сделать, начав поиск только по имени, или только по адресу, или по имени и адресу вместе.



Возвращаясь обратно к терминологии, вместо человека выступает какой-нибудь ресурс на сервере, и при помощи URI мы можем идентифицировать ресурс на сервере по его адресу или по его названию, либо по тому и тому вместе.

URL (Uniform Resource Locator) – это строка символов, которая используется для идентификации какого-либо ресурса, но только по его адресу, по его местоположению.



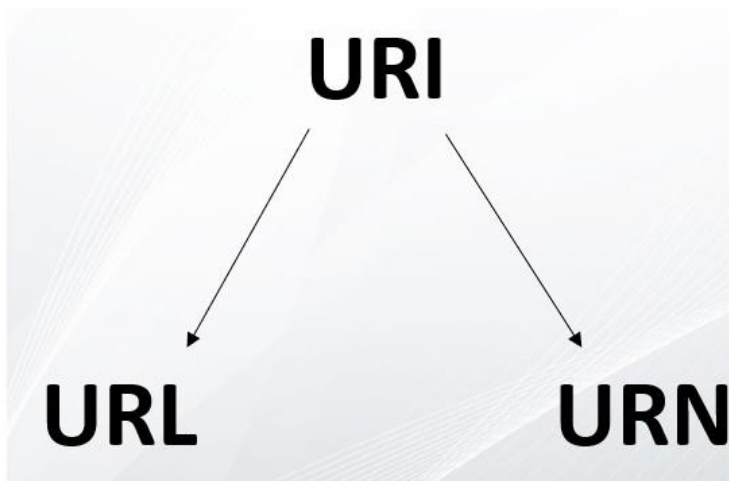
В примере с человеком это выглядит примерно так. В вебе, в сети Интернет именно URL чаще всего используется для обнаружения ресурсов на сервере.

URN (Uniform Resource Name) – это строка символов, которая используется для идентификации какого-либо ресурса, но только по его имени.



В нашем примере это выглядит так. Мы знаем этого человека, знаем, что его зовут Боб. Но мы не знаем, где он живет. Нам придется искать его только по имени.

Все эти три термина находятся в такой условной зависимости (или иерархии), как на картинке ниже. Потому что URI может использовать и адрес, и имя при идентификации ресурса. В то время как URL и URN только адрес и только имя соответственно.

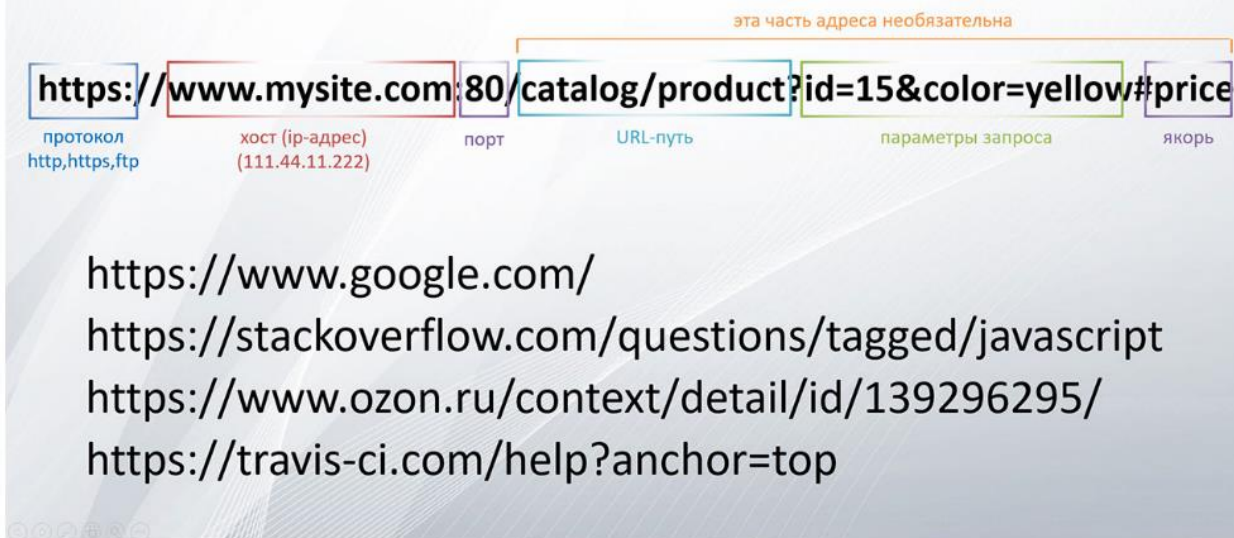


Каждый URL является URI. Каждый URN является URI. Но не каждый URI, к примеру, является URL (он может быть URN).

URL чаще всего используется в Интернете для поиска ресурсов на сервере. URL буквально точно показывает нам, как определить ресурс, именно по его адресу.

Указанный URL можно прочесть как: используя протокол https обратиться к домену www.mysite.com по стандартному порту 80, в каталоге найти товар желтого цвета с идентификатором 15, в браузере пользователя сразу же переместиться в область где указана цена.

URL



Любой URL состоит из нескольких компонентов. Протокол и хост являются обязательными, все остальные - нет.

URN служит для обозначения уникального имени ресурса, неважно, где этот ресурс располагается в данный момент времени или вообще. Такая природа URN (независимость от адреса) позволяет ресурсам перемещаться с одного места на другое.

URN



URN позволяет получить доступ к ресурсу по различным сетевым протоколам, обращаясь к одному и тому же имени.

На текущий день URN все еще считается экспериментальным и не так сильно распространен, как URL, так как для полной поддержки URN требуется поддерживающая его развитая сетевая инфраструктура.

Выводы: если мы говорим про сеть Интернет, то чаще всего используем термин URL, так как находим определенный ресурс в сети именно по его адресу на каком-то сервере. Также часто можно встретить аббревиатуру URI, подразумевающую именно URL. Хотя по факту это не совсем так, потому что URL является частью URI. В то же время в контексте веба URN практически не используется.

<https://alekseev74.ru/lessons/show/http-uri-url-urn>

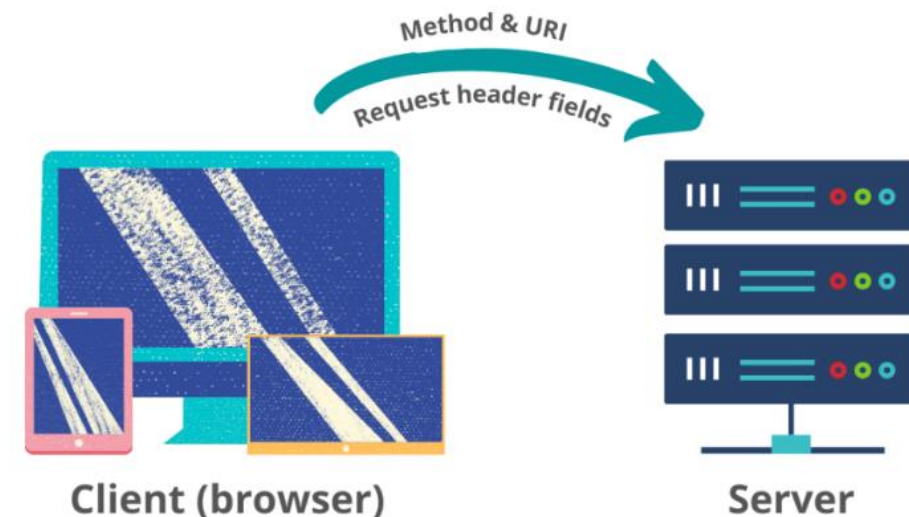
Протокол HTTP

HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

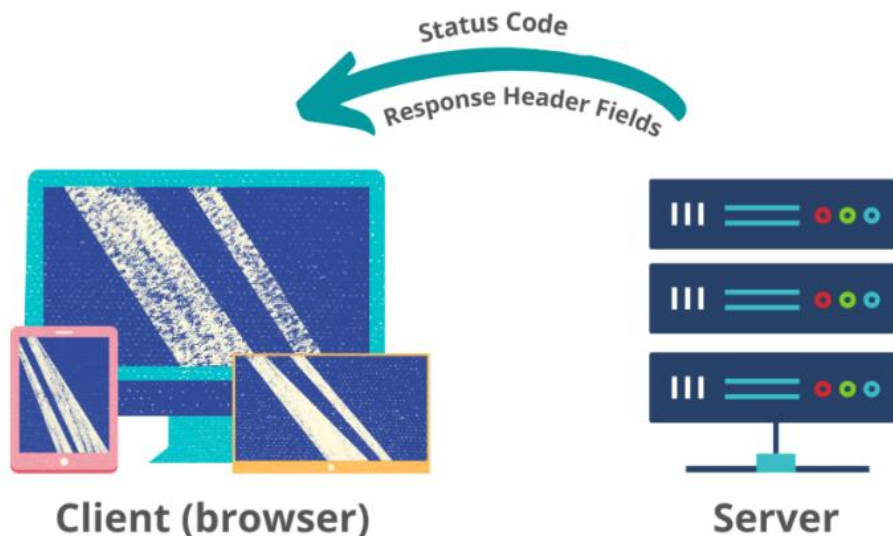
Потребителей (клиентов), которые инициируют соединение и посылают запрос;

HTTP Request



Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTP Response



Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. (в частности, для этого используется HTTP-заголовок). Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

HTTP — протокол прикладного уровня. Обмен сообщениями идёт по схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами (например, «куки» на стороне клиента, «сессии» на стороне сервера).

Структура HTTP-сообщения

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

Стартовая строка (англ. Starting line) — определяет тип сообщения;

Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;

Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа — только тело сообщения.

Для версии протокола 1.1 сообщение запроса обязательно должно содержать заголовок Host.

Методы

Метод HTTP (англ. HTTP Method) — операция над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру.

Сервер может использовать любые методы, не существует обязательных методов для сервера или клиента. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Возможные методы: OPTIONS, GET, HEAD, POST, PUT, PATCH, DELETE, TRACE, CONNECT

Наиболее часто используемые методы: GET, POST.

GET

Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

POST

Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы на сервер.

Коды состояния

Код состояния является частью первой строки ответа сервера. Он представляет собой целое число из трёх цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.

Мы стремимся к коду состояния 200, так как он означает, что всё прошло хорошо, и запрос выполнен успешно.

Код	Класс	Назначение
1xx	Информационный (англ. informational)	Информирование о процессе передачи. В HTTP/1.0 — сообщения с такими кодами должны игнорироваться. В HTTP/1.1 — клиент должен быть готов принять этот класс сообщений как обычный ответ, но ничего отправлять серверу не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. Прокси-серверы подобные сообщения должны отправлять дальше от сервера к клиенту.
2xx	Успех (англ. Success)	Информирование о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса, сервер может ещё передать заголовки и тело сообщения.
3xx	Перенаправление (англ. Redirection)	Сообщает клиенту, что для успешного выполнения операции необходимо сделать другой запрос (как правило по другому URI). Из данного класса пять кодов 301, 302, 303, 305 и 307 относятся непосредственно к перенаправлениям (редирект). Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке Location. При этом допускается использование фрагментов в целевом URI.
4xx	Ошибка клиента (англ. Client Error)	Указание ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.
5xx	Ошибка сервера (англ. Server Error)	Информирование о случаях неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.

Заголовки

Заголовки HTTP (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение. Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Все заголовки разделяются на четыре основных группы:

General Headers («Основные заголовки») — могут включаться в любое сообщение клиента и сервера;

Request Headers («Заголовки запроса») — используются только в запросах клиента;

Response Headers («Заголовки ответа») — только для ответов от сервера;

Entity Headers («Заголовки сущности») — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посылать заголовки получателю.

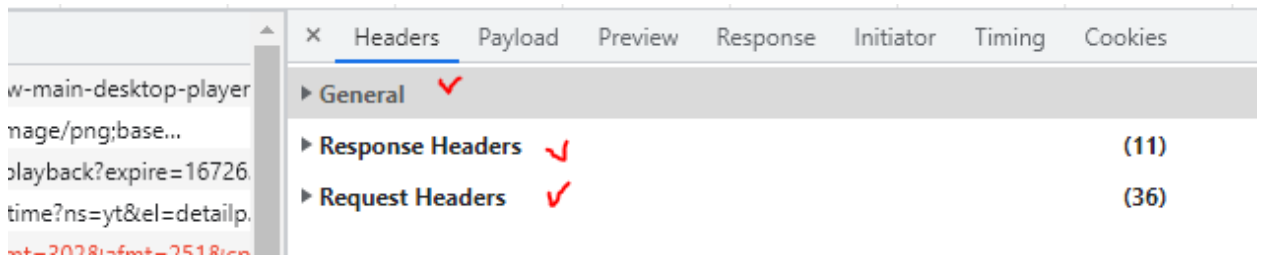
<https://habr.com/ru/company/kts/blog/669784/>
<https://ru.wikipedia.org/wiki/HTTP>
https://www.youtube.com/watch?v=URg_Q8dclzg

Упражнение «Исследование заголовков запросов» (№ 1)

1. В браузере открыть любой сайт.
2. Открыть DevTools, перейти на вкладку NetWork (сеть), которая позволяет мониторить процесс загрузки страницы и всех файлов которые подгружаются при загрузке.
3. В таблице в поле Name выбрать любую строку содержащую метод GET или POST

Name	Url	Method	Status	Domain
<input type="checkbox"/> www-main-desktop-player-skeleton.css	https://ww...	GET	200	www.y
<input type="checkbox"/> data:image/png;base...	data:image...	GET	200	
<input type="checkbox"/> videoplayback?expire=1672673723&ei=W...	https://rr2-...	POST	200	rr2---s
<input type="checkbox"/> watchtime?ns=yt&el=detailpage&cpn=Q...	https://ww...	GET	204	www.y

Строку выбирать таким образом, чтобы в появившемся окне были видны три заголовка



4. Исследуем некоторые элементы заголовка General, который содержит общую информацию о запросе:

В каком заголовке содержится адрес ресурса в запросе?

Каким методом выполнен запрос?

Какой статус вернул сервер? Что это значит?

В каком заголовке содержится удаленный адрес сервера?

5. Исследуем некоторые элементы заголовка Response Headers – это информация, присылаемая сервером в ответ на запрос пользователя:

Content-type – тип возвращаемого ответа. Сервер может ответить несколькими способами и форматами: обычный текст, формат json (это специальный формат данных для понимания языком JS), формат XML и др.

Date

Server – название сервера

6. Исследуем некоторые элементы заголовка Request Headers, в котором указаны заголовки, отправленные серверу. Часть заголовков формируется автоматически браузером, например, user-agent – браузер, из которого пользователь выполняет запросы

Формат JSON

JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми.

Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

JSON обладает двумя основными функциональными возможностями:

- является форматом данных, передаваемых между клиентом и сервером;
- используется для определения конфигураций.

Синтаксис JSON прост и обладает ограниченной поддержкой типов данных, к которым относятся object ({}), array ([]), number, string, boolean и null. Однако функции,

NaN, Infinity, undefined и Symbol не являются допустимыми значениями JSON. JSON не имеет поддержки пространств имен, комментариев или атрибутов. Он не может поддерживать сложные конфигурации. Эти ограничения делают JSON простым и доступным, и поэтому он быстро усваивается и легко интерпретируется.

Объект в формате JSON имеет несколько важных отличий от объектного литерала:

- строки используют двойные кавычки. Никаких одинарных кавычек или обратных кавычек в JSON. Так 'John' становится "John".

- имена свойств объекта также заключаются в двойные кавычки. Это обязательно. Так age:30 становится "age":30.

JSON предлагает два статических метода — JSON.parse() и JSON.stringify().

JSON.parse()

JSON.parse(text) парсит строку JSON для создания значения или объекта JavaScript. Для объектов имена свойств JSON должны быть строками с двойными кавычками, а запятые в конце строки запрещены. Для примитивных типов JSON.parse() возвращает примитивные значения. Для чисел запрещены начальные нули, а за десятичной точкой должна следовать хотя бы одна цифра. Любые нарушения синтаксиса JSON приводят к ошибке SyntaxError.

JSON.stringify()

JSON.stringify(value) возвращает JSON-строку, соответствующую указанному value. boolean, number и string преобразуются в соответствующие примитивные значения. Функции, undefined и Symbol являются недопустимыми значениями JSON, которые опускаются в объекте или заменяются на null в массиве. NaN, Infinity и null заменяются на null. Полученная строка json называется JSON-форматированным или сериализованным объектом. Его можно отправить по сети или поместить в обычное хранилище данных.

```
<script>
  let json1 = '[12, 45, 20, 135, 7]'; // массив
  console.log(json1);

  let json2 = '{"firstName":"Иванов","lastName":"Иван","age": 20}'; // простой
  объект

  let data2 = JSON.parse(json2); //метод parse объекта JSON распарсил строку
  json2 в объект data2 с набором атрибутов
  console.log(data2);
  console.log(data2.age);
  console.log(data2.firstName);
  console.log(data2.lastName);

  let obj = {
    firstName:"Петров",
    lastName:"Петр",
    age: 21
  }
  console.log(obj);

  let json3 = JSON.stringify(obj); //метод stringify объекта JSON вернул объект
  obj в строку json3
  console.log(json3);

  // объект со сложной структурой
  let json4 = `{
```



```

        "book": {
            "number1": {
                "author": "Достоевский",
                "title": "Идиот"
            },
            "number2": {
                "author": "Чехов",
                "title": "Палата № 6"
            }
        }
    }`
let data4 = JSON.parse(json4);
console.log(data4);
console.log(data4.book.number1.title);

</script>
</body>
</html>

```

Упражнение № 2

Имеется строка

```

{
  "firstName": "Иванов",
  "lastName": "Иван",
  "age": 20,
  "address": {
    "streetAddress": "пл. Гагарина, 1",
    "city": "Ростов-на-Дону",
    "postalCode": 344000
  },
  "phoneNumbers": [
    {
      "type1": "home",
      "number1": "634-5625-45-63"
    },
    {
      "type2": "fax",
      "number2": "634-5625-45-64"
    }
  ]
}

```

Выполнить следующие действия:

1. Распарсить строку в объект.
2. Вывести полученный объект в консоль.
3. Вывести в консоль фамилию, возраст, город.
4. Вывести в консоль все номера.
5. Вывести в консоль номер 634-5625-45-64.
6. Из имеющейся строки создать объект. Затем полученный объект преобразовать в строку JSON

Решение

//Упражнение

```

let json5 = `{
  "firstName": "Иванов",

```

```
"lastName": "Иван",
"age": 20,
"address": {
  "streetAddress": "пл. Гагарина, 1",
  "city": "Ростов-на-Дону",
  "postalCode": 344000
},
"phoneNumbers": [
  {
    "type1": "home",
    "number1": "634-5625-45-63"
  },
  {
    "type2": "fax",
    "number2": "634-5625-45-64"
  }
]
}`

data5 = JSON.parse(json5);
console.log(data5);
console.log(data5.address.postalCode);
console.log(data5.phoneNumbers[1].number2);
```

<https://ru.wikipedia.org/wiki/JSON>

<https://learn.javascript.ru/json>

<https://medium.com/nuances-of-programming/углубленное-изучение-json-json5-и-циклических-зависимостей-4ec6c7f53a45>

<https://habr.com/ru/post/554274/>

Node.js

Node.js - это технология, которая позволяет запускать код, написанный на JS вне браузера. Технология позволяет создавать back-end часть сайта, в то время как написание js-кода для выполнения в браузере – это front-end часть сайта.

Весь технологический стек разработки можно разделить на две части:

- front-end отвечает за видимую часть сайта, т.е. то что видит пользователь и с чем он может взаимодействовать
- back-end – это то, что работает на сервере. Он нужен для обработки событий, которые приходят с front-end и дальнейших манипуляций с ними. Это может быть взаимодействие с БД, роутинг, редирект и т.п.

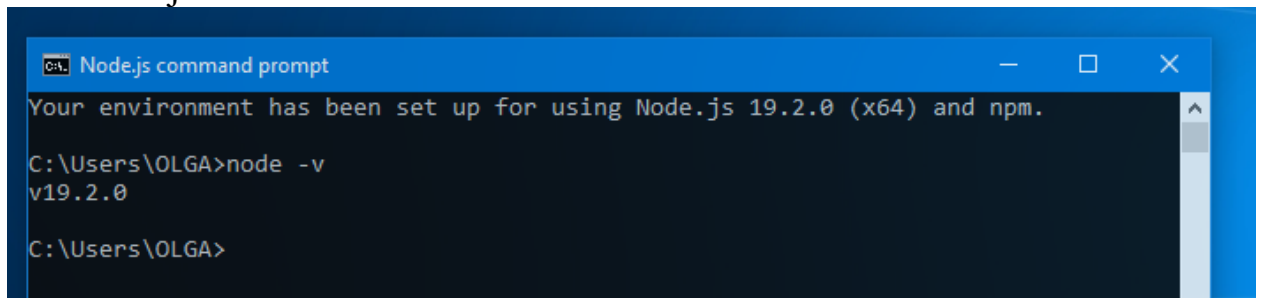
Чтобы создать серверную часть сайта используются серверные языки: Python, Java, PHP, Ruby и т.п.

В основе все базовой логики front-end лежит только JS. Со временем появилась идея создания back-end технологии, которая возьмет на себя создание серверной части приложения и будет использовать синтаксис JS. Так появился Node.js.

Для того чтобы код заработал на ПК его нужно компилировать в машинный код. В браузере это делает «под капотом» движок V8 (он написан на C++). В Node.js реализован этот же движок, что и позволяет выполнять JS-код вне браузера.

Практическое занятие 1

1. Запустить на ПК Node.js command prompt. Проверить версию Node.js

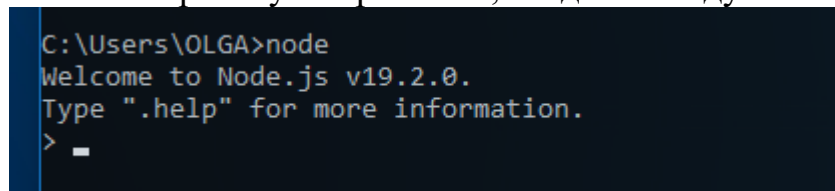


```
Node.js command prompt
Your environment has been set up for using Node.js 19.2.0 (x64) and npm.

C:\Users\OLGA>node -v
v19.2.0

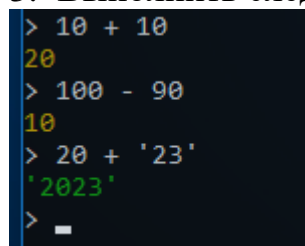
C:\Users\OLGA>
```

2. Начать работу в терминале, введя команду node



```
C:\Users\OLGA>node
Welcome to Node.js v19.2.0.
Type ".help" for more information.
> _
```

3. Выполнить следующие операции



```
> 10 + 10
20
> 100 - 90
10
> 20 + '23'
'2023'
> _
```

```
> console.log("Hello Node JS")  
Hello Node JS
```

```
> !!'true'  
true
```

4. Для выхода из терминала – Ctrl+C

Локальный хост

localhost или <http://127.0.0.1:8080> (протокол 8080 можно не писать)

Комп, подключенный к сети, называют хостом. К нему обращаются по уникальному адресу или имени. Одному имени (т.е. одному хосту) могут соответствовать несколько разных адресов.

Когда к хосту обращаются по имени, это имя сначала разрешается в адрес хоста, по которому, собственно, и происходит обращение. В сети этим обычно занимается DNS, но если DNS нет, соответствие имен-адресов можно прописать в файле hosts на самом хосте. Если некое имя хоста и там не прописано, обратиться к нему по имени не удастся... хотя прямое обращение по адресу будет работать.

Сервер - это программа, отвечающая на запросы из сети. Комп, на котором она выполняется, также называют "сервером". Причем, даже если выполнение программы приостановлено (например, проводят профилактику или программа-сервер упала), этот комп все равно будут называть "сервером", ибо он предназначен, в основном, для выполнения этой программы.

На одном компе (=хосте, сервере) может одновременно выполняться несколько разных программ-серверов. Для того, чтоб обратиться к конкретной из них (адрес-то у всех один и тот же!), в протоколе TCP/IP используются разные номера портов.

Если на компе запущен, например, HTTP сервер (= Webserver, например, Nginx или Apache), он "слушает" порт 80, а если не запущен, порт 80 никто не слушает, и, если обратиться к такому хосту (= серверу, компу) по его адресу в порт 80, никакого ответа не придет... хотя сам хост и будет доступен.

Для разных общеизвестных типов программ-серверов (в этом случае также говорят о "сервисах" или же "протоколах", что в данном контексте практически одно и то же) принято использовать общеизвестные номера портов, а для наиболее распространенных (как тот же HTTP) можно даже не указывать номер порта при обращении, как мы обычно и делаем в строке браузера, т.к. клиент автоматически использует номер порта по умолчанию, в данном случае 80. Но, в принципе, любой сервис можно (переконфигурировав) использовать на любом порту... если, конечно, в этом есть смысл. Единственно, что нельзя - одновременно использовать разные серверы на одном порту.

И, наконец, было бы совсем глупо, если бы для обращения к какому-то серверу на одном хосте в сети обязательно был бы нужен еще и другой комп, с которого обращаться. Вот и придумали возможность обратиться к программе-серверу с того же хоста, на котором она выполняется, т.е. локально, а чтоб не гадать, по какому адресу или имени это делать, ввели понятие localhost.

localhost - "общеизвестное" имя компа для самого себя и ему соответствует IP адрес 127.0.0.1. Это - общепринятая договоренность, которую просто нужно знать. Если говорят "установить сервер на localhost", это означает "установить на тот самый комп, с которого и обращаться к этому серверу".

<https://qna.habr.com/q/228439>
<https://www.youtube.com/watch?v=10KGuTZ8oio>

Практическое занятие 2

1. В VS code создать новую папку (пустой проект) с именем WWW.
2. Выполнить инициализацию проекта в node. Для этого создать новый терминал (Terminal → New Terminal). В окне терминала набрать команду

```
\www> npm init
```

и ответить на вопросы, предложенные node (ответ или его отсутствие подтверждать Enter). package name писать только в нижнем регистре.

```
package name: (www) training
version: (1.0.0) 0.0.1
description: учебный проект начат 05.02.2023
entry point: (index.js)
test command:
git repository:
keywords:
author: Manakova
license: (ISC)
```

Получаемый результат:

```
{
  "name": "training",
  "version": "0.0.1",
  "description": "учебный проект начат 05.02.2023 ",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Manakova",
  "license": "ISC"
}
```

Затем node сформирует файл зависимостей, который появится в структуре проекта (файл package.json) с примерным содержимым (см. ниже). Такой файл node формирует всегда при создании node.js – проекта. В нем содержатся все характеристики проекта: название, версия, автор и т.п. В дальнейшем в этот файл будут внесены все пакеты (библиотеки для работы проекта).

```
{ } package.json > ...
1  {
2    "name": "training",
3    "version": "0.0.1",
4    "description": "учебный проект начат 05.02.2023 ",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Manakova",
10   "license": "ISC"
11 }
12
```


Свойство `scripts` данного файла содержит скрипты, которые можно будет запускать через пакетный менеджер `npm`.

3. Запустим скрипт, который есть в проекте по умолчанию — `test`. Данный скрипт выведет в консоль свое содержимое. Для этого выполнить в терминале команду

```
npm run test
```

Данная команда запускает на выполнение любой `.js` файл

Получаем результат

```
> training@0.0.1 test
> echo "Error: no test specified" && exit 1

"Error: no test specified"
```

Теперь можно удалить данный скрипт

```
"scripts": {},
```

4. После инициализации проекта в нем можно создавать и выполнять файлы на платформе `node.js`.

5. Создадим следующий файл (`index.js`) и выполним его на платформе `node.js`.

```
let a = 3;
let b = 4;
console.log(a+b);

let i = 0;
setInterval(function(){
  console.log(++i);
}, 1000);
```

Для запуска: в терминале набрать `node index`

6. Вернемся в файл `package.json` и создадим скрипт, который будет выполняться каждый раз при запуске проекта.

Внести в файл `package.json` следующее:

```
"scripts": {
  "start": "node start.js"
},
```

Содержимое файла `start.js`

```
let num = 5500;
console.log(`localhost, port: ${num}`);
```

Затем выполнить в терминале:

```
npm run start
```

Результат:

```
> training@0.0.1 start
> node start.js

Localhost, port: 5500
```

7. Создадим в проекте первый сервер (файл server1.js):

```
const http = require('http')
http.createServer(function(request, response){
  response.end('Hello NodeJS')
}).listen(5500, "127.0.0.1", function(){
  console.log("Сервер начал прослушивание запросов на порту 5500")
})
```

Сначала запрашивается необходимый модуль, затем создаётся сервер и запускается на заданном порту. Метод `createServer()` объекта `http` принимает в качестве аргумента анонимную функцию-колбэк, аргументами которой, в свою очередь, служат объекты `request` и `response`. Они соответствуют поступавшему HTTP-запросу и отдаваемому HTTP-ответу.

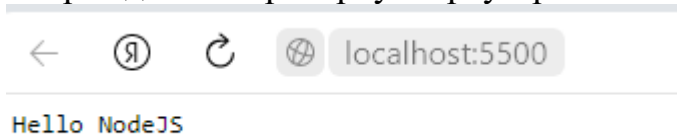
Для запуска сервера набрать в терминале

```
node server1.js
```

Результат:

```
Сервер начал прослушивание запросов на порту 5500
```

Теперь сделаем проверку в браузере:



The screenshot shows a web browser address bar with the URL `localhost:5500`. Below the address bar, the text `Hello NodeJS` is displayed in a monospace font.

Что бы остановить работу сервера достаточно нажать `Ctrl + C`.

Пакетный менеджер npm

Кроме встроенных и кастомных модулей Node.js существует огромный пласт различных библиотек и фреймворков, разнообразных утилит, которые создаются сторонними производителями и которые также можно использовать в проекте, например, express, grunt, gulp и так далее. И они тоже нам доступны в рамках Node.js. Чтобы удобнее было работать со всеми сторонними решениями, они распространяются в виде пакетов. Пакет по сути представляет набор функциональностей.

Для автоматизации установки и обновления пакетов, как правило, применяются системы управления пакетами или менеджеры. Непосредственно в Node.js для этой цели используется пакетный менеджер NPM (Node Package Manager). NPM по умолчанию устанавливается вместе с Node.js, поэтому ничего доустанавливать не требуется.

Официальный сайт библиотек модулей Node.js - <https://www.npmjs.com/>

Практическое занятие 3

1. Зайти на сайт <https://www.npmjs.com/>
2. В строке поиска набрать cowsay. Перейти на страницу этой библиотеки, ознакомиться с информацией о библиотеке. Скопировать команду на установление

Install

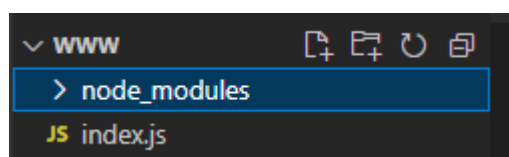
```
> npm i cowsay
```

и выполнить ее в терминале проекта.

Результат

```
added 41 packages, and audited 42 packages in 4s
3 packages are looking for funding
run `npm fund` for details
```

В структуре проекта появилась новая папка node_modules с набором библиотек, необходимых проекту. Как правило, для работы программиста она не требуется, но удалять ее нельзя, т.к. она необходима нашему проекту.



3. Найти установленную библиотеку cowsay, просмотреть содержимое ее файла package.json. Так же в проекте появился файл package-lock.json, в котором указано более полное описание проекта (посмотрите его). Нам для

работы он не потребуется, т.к. он больше нужен системе для информации о проекте. Для прямого общения с проектом у нас есть файл package.json.

4. Начнем работу с библиотекой cowsay.

В терминале выполнить команду, предложенную на сайте

Usage

```
cowsay JavaScript FTW!
```

Результат

```
строка:1 знак:1
cowsay : Имя "cowsay" не распознано как имя командлета, функции, файла сценария или выпол-
няемой программы. Проверьте правильность написания имени, а также наличие и правильность
пути, после чего повторите попытку.
строка:1 знак:1
+ cowsay JavaScript FTW!
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (cowsay:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException
```

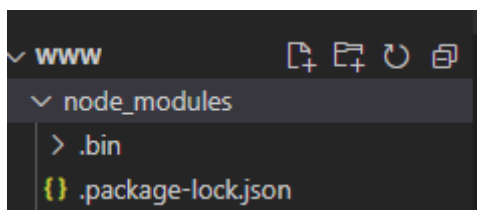
Ошибка возникла из-за того, что пакет установлен локально, т.е. только в нашем проекте. Удалим его:

```
npm uninstall cowsay
```

Результат

```
removed 41 packages, and audited 1 package in 722ms
```

Изменилась структура проекта



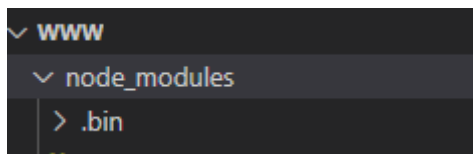
Установим его снова, но глобально, т.е. на ПК (а не в проект).

```
npm install -g cowsay
```

Результат

```
added 41 packages, and audited 42 packages in 4s
```

Структура проекта не изменилась

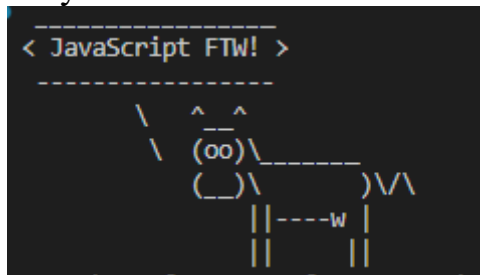


5. Снова выполним

Usage

```
cowsay JavaScript FTW!
```

Результат



Библиотека работает!

6. Самостоятельно придумать текст и для нескольких команд. Выполнить их в терминале. Показать преподавателю.

7. Найдем место расположения пакета на ПК пользователя. Как правило, пакет устанавливается в папке, например, C:\Users\OLGA\AppData\Roaming\npm\node_modules). Что бы убедиться в этом введем команду

```
npm root -g
```

8. Установим в наш проект еще один пакет lodash

Lodash помогает в работе с массивами, строками, объектами, числами и т.д. Он предоставляет нам различные встроенные функции и использует подход функционального программирования, который облегчает понимание программирования на JavaScript, поскольку вместо написания повторяющихся функций задачи могут выполняться с помощью одной строки кода

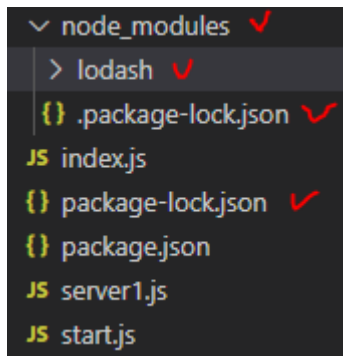
```
npm i lodash
```

Проверим содержимое файла package.json. В нем появился объект «dependencies», в котором указываются библиотеки (зависимости), которые нужны для корректной работы проекта. В нашем случае – установленный пакет lodash и его версия.

```
"dependencies": {  
  "lodash": "^4.17.21"
```


Объект «dependencies» позволяет установить в проект все необходимые ему библиотеки: как правило, при пересылке проекта кому-либо не пересылается папка node_modules, т.к. она может занимать много места. Получатель проекта на своем ПК выполнит команду `npm i` и все необходимые библиотеки установятся на его ПК. Убедимся в этом:

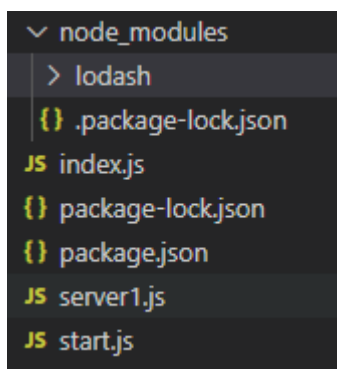
- удалим из проекта помеченные папки и файлы.



- выполним команду

```
npm i
```

Результат — все файлы и пакеты «вернулись» в проект:



Работа с модулями

Практическое занятие № 4

В процессе работы с node.js можно использовать встроенные и пользовательские (кастомные) модули, а также множество библиотек. Рассмотрим общие принципы работы с ними.

1. Создадим новый файл index_module.js.

2. Подключим встроенный модуль os, который предоставляет информацию о пользователе ПК: его имя, платформа и др.

Для подключения модулей используют функцию require.

```
1 const os = require('os')
```

3. Обратимся к функции platform модуля os

```
1 const os = require('os')
2 let plat = os.platform()
3 console.log(plat);
```

Выполним программу -

```
node index2_module.js
```

Результат -

```
win32
```

Самостоятельно вывести в консоль: имя пользовательского ПК, информация о пользователе, о версии ОС, о пользовательской директории.

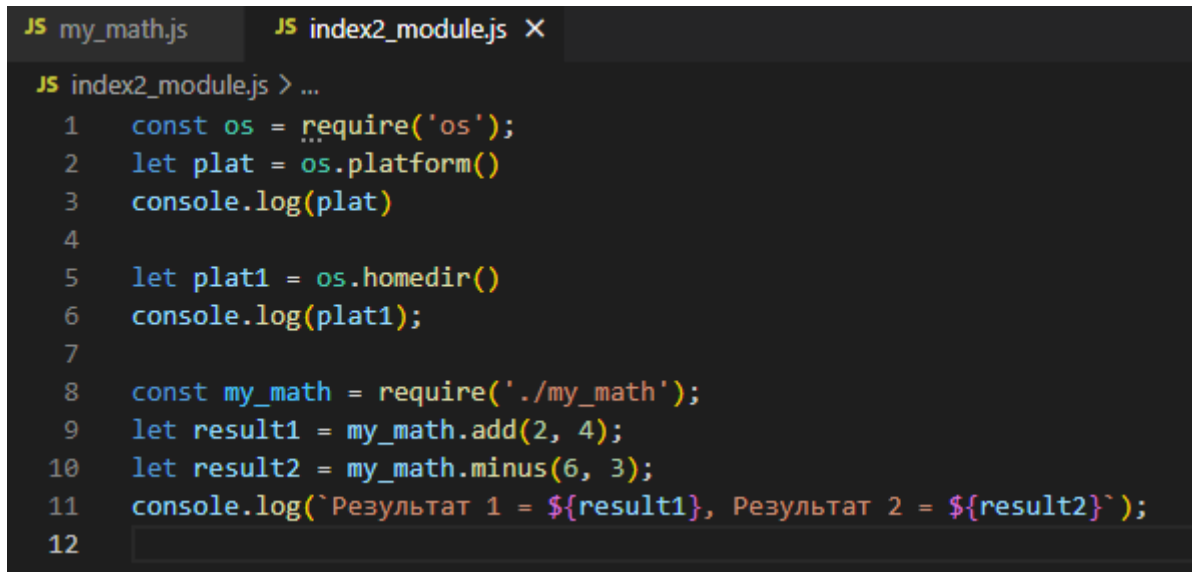
4. Создание собственного модуля, что означает создание либо собственной папки, либо собственного модуля и затем его подключение внутри других файлов.

В корне проекта создадим новый файл my_math.js для работы с объектом Math. В этом файле будем описывать различные математические функции, которые затем будем экспортировать в нужный файл.

```
JS my_math.js • JS index2_module.js
JS my_math.js > [?] minus
1 const add = (a, b) => {
2   return a + b
3 }
4
5 const minus = (a, b) => {
6   return a - b
7 }
8
9 module.exports = {
10   add: add,
11   minus: minus
12 }
```

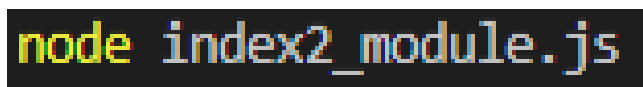
Команда `module.exports` обеспечивает экспортирование нужных функций, переменных, объектов и т.п.

Дополним имеющийся файл `index2_module.js` вызовами функций из `my_math.js`



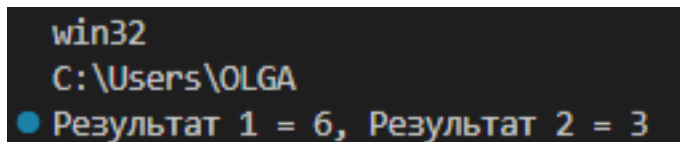
```
JS my_math.js JS index2_module.js X
JS index2_module.js > ...
1  const os = require('os');
2  let plat = os.platform()
3  console.log(plat)
4
5  let plat1 = os.homedir()
6  console.log(plat1);
7
8  const my_math = require('./my_math');
9  let result1 = my_math.add(2, 4);
10 let result2 = my_math.minus(6, 3);
11 console.log(`Результат 1 = ${result1}, Результат 2 = ${result2}`);
12
```

Выполним файл `index2_module.js`



```
node index2_module.js
```

Результат



```
win32
C:\Users\OLGA
● Результат 1 = 6, Результат 2 = 3
```

5. Самостоятельно составить функции для расчета произведения и деления двух чисел, обеспечить вывод результатов в консоль.