<https://github.com/GossJS/gosshttpresearch>

13.09.2019 на II курсе были рассмотрены основы реализации HTTP на Node.js

<https://kodaktor.ru/13092019>

Были написаны сервер

**const** **{** Server **}** **=** require**(**'http'**);**

**const** s **=** **new** Server**();**

s**.**on**(**'request'**,** **(**req**,** res**)** **=>** **{**

**const** **{** url **}** **=** req**;**

**const** hu **=** **{**

'Content-Type'**:** 'text/html; charset=utf-8'

**};**

res**.**writeHead**(200,** **{** **...**hu **});**

res**.**end**(**`Привет мир с URL **=** $**{**url**}!**`**);**

**});**

s**.**on**(**'connection'**,** **()** **=>** console**.**log**(**process**.**pid**));**

s**.**listen**(4321,** '127.0.0.1'**);**

и клиент

**const** **{** request**:** r **}** **=** require**(**'http'**);**

**const** options **=** **{**

hostname**:** 'localhost'**,**

port**:** **4321,**

method**:** 'POST'**,**

path**:** '/'

**};**

r**(**options**,** res **=>** res**.**on**(**'data'**,** d **=>** console**.**log**(**String**(**d**))))**

**.**end**();**

Задание на дом – добавить поддержку метода POST.

**Цель исследования** – глубже погрузиться в механизм веб-протоколов. В Node.js есть модуль net и модуль http. В первом имеется объект Сокет, через который полнодуплексно реализуется всё взаимодействие (это поток для чтения и записи), а во втором реализуется более высокий уровень абстракции: вводятся Запрос и Ответ. Это тоже потоки.

На уровне net то, что мы отсылаем методом POST представляет собой единый «кусок» данных. С более высокой точки зрения там находятся заголовк и тело, между которыми два символа перевода строки. Модуль http оперирует ими по отдельности: заголовки доступны серверу через req.headers и req.url, а тело – через поток.

Метод http.request() возвращает объект класса ClientRequest. Он абстрагирует находящийся в ходе выполнения запрос. С помощью метода setHeader(name, value), мы можем изменять его заголовки. С помощью метода end() мы можем отправить запрос.

В файле sender.js мы как раз посылаем такой запрос. Ответ сервера тоже рассматривается по отдельности: мы можем посмотреть заголовки ответа в событии request.on('response') и мы можем прочитать тело сообщения через поток.

Файлы sender-net-get-with-body.js и server-net.js позволяют создать примитивную реализацию взаимодействия веб-клиента и веб-сервера на низком уровне сокетов. В частности, server-net может вместо HTTP /1.1 отправить XXXX /6.66 и клиент это благополучно получит и покажет, никаких ошибок не возникнет. А вот если запустить server-net в компании с sender.js, то тут плохой ответ сервера вызовет ошибку request.on('error').

Например, если в одном окне терминала выполнить ./server-net.js XXXX

а в другом ./sender.js

то получим срабатывание on('error') с сообщением Parse Error: Expected HTTP/

HPE\_INVALID\_CONSTANT означает, что начало ответа не соответствует протоколу (i.e. не начинается с "HTTP".)

Это так же означает, что посылаемый заголовок Content-Length не соответствует (меньше) реально посланного ответа.

Сервер index.js

**const** **{** createServer**:** cS **}** **=** require**(**'http'**);**

**const** **{** table**,** log **}** **=** console**;**

cS**((**req**,** res**)** **=>** **{**

**const** **{** url**,** method**,** headers **}** **=** req**;**

**const** hu **=** **{**

'Content-Type'**:** 'text/html; charset=utf-8'

**};**

table**(**headers**);** log**(**method**);**

**const** init **=** 'Привет мир с URL = '**;**

**let** pssst **=** ' GET '**;**

**if** **(**method **!==** 'GET'**)** pssst **=** ' NOGET '**;**

**let** b **=** ''**;**

req

**.**on**(**'data'**,** d **=>** b **+=** d**)**

**.**on**(**'end'**,** **()** **=>** **{**

pssst **+=** b**;**

res**.**writeHead**(200,** **{** **...**hu **});**

res**.**end**(**`$**{**init**}**$**{**url**}**$**{**pssst**}!**`**);**

**});**

**})**

**.**listen**(4321,** '127.0.0.1'**,** **()** **=>** log**(**process**.**pid**));**

При любом методе запроса сработает end, поэтому мы можем поместить отправку ответа туда.

Клиент sender.js

Можно вызывать ./sender.js POST haha

Или ./sender.js GET

И разумеется **curl localhost:4321/poop -i -d dgdg -XGET**

#**!**/usr/local/bin/node

**const** **{** request**:** r **}** **=** require**(**'http'**);**

**const** **{** table**,** log **}** **=** console**;**

**const** options **=** **{**

hostname**:** 'localhost'**,**

port**:** **4321,**

method**:** process**.**argv**[2]** **||** 'GET'**,**

path**:** '/kkk'

**};**

**const** body **=** process**.**argv**[3]** **||** ''**;**

**const** rq **=** r**(**options**,** res **=>** **{**

res**.**on**(**'data'**,** d **=>** log**(**'>>>' **+** String**(**d**))** **||** table**(**res**.**headers**));**

**})**

**.**on**(**'error'**,** e **=>** log**(**e**))**

**.**on**(**'response'**,** d **=>** table**(**d**.**rawHeaders**)** **||** log**(**d**.**statusCode **+** ' ' **+** d**.**statusMessage**));**

rq**.**setHeader**(**'Content-Length'**,** body**.**length**);** *// без этого не получится послать GET+body*

rq**.**end**(**body**);**

// при посылке методами POST, PUT, PATCH

// body посылается без всяких проблем

// чтобы послать GET и непустое body нужно также послать Content-Length

// php devserver на GET с body без Content-Length отвечает сбросом соединения

// (ECONNRESET) – это будет on.error

//

// а index.js ответит 400 Bad Request – это будет on.response

// ./sender.js DELETE hey – пример вызова из командной строки

Спуск на уровень ниже к модулю net

server-net.js

#**!**/usr/local/bin/node

**const** prot **=** process**.**argv**[2]** **||** 'HTTP'**;**

**const** resp **=**

`$**{**prot**}/1.1** **200** OK

Content**-**Type**:** text**/**plain**;** charset**=**utf**-8**

Content**-**Length**:** **12**

Привет

`**;**

require**(**'net'**)**

**.**Server**(sock** **=>** **sock.**on**(**'data'**,** d **=>** console**.**log**(**String**(**d**))** **||** **sock.**end**(**resp**)))**

**.**listen**(4321);**

sender-net-get-with-body.js

#**!**/usr/local/bin/node

require**(**'net'**)**

**.**connect**(4321,** 'localhost'**)**

**.**on**(**'data'**,** d **=>** console**.**log**(**String**(**d**)))**

**.**end**(**'GET / HTTP/1.1\nContent-Length:4\n\nkaka'**);**

Запускать можно в разных комбинациях, например сервер на net и клиент на http и исследовать. ./server-net.js XXXX – для подстановки XXXX вместо HTTP и моделирования ошибки ответа

Разумеется здесь нет всех деталей и тонкостей, типа keep-alive, но это отводится на дальнейшее самостоятельное исследование.