Модуль HTTP (сервер http)

МОДУЛЬ HTTP

Node.js использует модульную систему. То есть вся встроенная функциональность (API NodeJS) разбита на отдельные модули. Модуль представляет блок кода, который может использоваться повторно в других модулях.

Для работы с сервером и протоколом http в Node.js используется модуль http.

Модуль http предоставляет функционал:
- создания http сервера;
- создания http клиента;
- справочник методов HTTP запросов (GET, POST, и т.п.);
- справочник кодов состояния ответа HTTP.

МОДУЛЬ HTTP

В данной презентации рассматриваются только вопросы создания http сервера.

Чтобы использовать данный модуль его необходимо подключить с помощью require:

```
const http = require('http');
```

Модуль http содержит класс реализующий http сервер и есть несколько способов создания экземпляра http сервера по заданному классу:

- конструктор класса Server;
- фабричный метод createServer.

Чтобы http сервер заработал у экземпляра сервера необходимо задать:

- функцию обработчик входящего запроса;
- номер порта на который будут приходить http запросы.

Конструктор класса Server

Вот так выглядит вариант создания http сервера через конструктор класса:

```
const http = require('http'); //подключение модуля http
//порт для прослушивания входящих сообщений
const port = 80;
//создание экземпляра сервера через конструктор
const server = new http.Server();
/*задание у сервера функции обработчик входящего запроса */
server.on('request', (req, res)=>{
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, client');
});
/*задание у сервера порта для прослушивания входящих сообщений */
server.listen(port);
```

Конструктор класса Server

Вот так выглядит вариант создания http сервера через фабричный метод createServer:

```
const http = require('http'); //подключение модуля http
//порт для прослушивания входящих сообщений
const port = 80;
/*создание экземпляра сервера через фабричный метод createServer */
const server = http.createServer();
/*задание у сервера функции обработчик входящего запроса*/
server.on('request', (req, res)=>{
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, client');
});
/*задание у сервера порта для прослушивания входящих сообщений */
server.listen(port);
```

Конструктор класса Server

Фабричный метод createServer может принимать в качестве аргумента функцию обработчик запроса:

```
const http = require('http'); //подключение модуля http
//порт для прослушивания входящих сообщений
const port = 80;
/*создание экземпляра сервера через метод createServer с заданием у сервера функции обработчика входящего запроса */
const server = http.createServer((req, res)=>{
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, client');
});
/*задание у сервера порта для прослушивания входящих сообщений */
server.listen(port);
```

Экземпляр http сервер

Экземпляр http сервера обладает свойства и методами для настройки и работы с http сервером.

Пример свойства:

- listening - логическое значение, указывающее на то обрабатывает ли сервер соединения (сервер может обрабатывать соединения только после указания серверу порта для прослушивания входящих запросов).

Примеры методов:

- on(eventName, listener) или addListener(eventName, listener) – задание функции обработчика listener на указанное событие eventName;
- close([callback]) – останавливает сервер от принятия новых подключений;
- listen([port][, hostname][, backlog][, callback]) - сервер начинает принимать соединения на указанном порту. Параметр port – номер порта, hostname – хост, backlog – размер очереди входящих сообщений (по умолчанию определяется операционной системой).

Экземпляр http сервер

Примеры событий eventName:

- 'clientError' – генерируется если соединение с клиентом генерирует событие 'error', то он будет переадресован сюда (например, обработка ошибки 404);
- 'close' – генерируется при завершении работы сервера;
- 'connection' – генерируется при создании нового TCP соединения с новым клиентов;
- 'request' – генерируется каждый раз когда приходит новый запрос;
- 'listening' – генерируется после того как сервер был привязан к порту, т.е. после вызова метода listen();
- 'error' – генерируется при возникновении ошибки.

!!!У разных событий в функцию обработчик будут приходить разные аргументы характеризующие событие

Обработка события request

При поступлении на http сервер нового запроса, происходит событие request. В функцию обработчик данного события поступают два объекта:

- ❖ экземпляр класса http.IncomingMessage – содержащий данные о входящем запросе и методы для работы с входящим сообщением;
- ❖ экземпляр класса http.ServerResponse – содержащий данные о формируемом ответе на запрос клиента и методы для изменения этих данных и завершения сообщения (т.е. отправки его клиенту).

!!! Каждый ответ должен завершатся, иначе он не будет отправлен клиенту

Экземпляр IncomingMessage

Пример свойств:

- headers – объект с заголовками входящего запроса вида ключ (название заголовка в нижнем регистре) / значение (значение заголовка);
- httpVersion – строка с номером версии HTTP запроса;
- method – строка с методом HTTP запроса;
- url – строка с URL адрессом, который присутствует в фактическом запросе HTTP.

Пример метода:

- on(eventName, listener) или addListener(eventName, listener) – задание функции обработчика listener на указанное событие eventName.

Экземпляр ServerResponse

Пример свойств:

- finished – логическое значение (доступно только на чтение), указывающее, завершен ли этот ответ (по умолчанию false – не завершён);
- headersSent – логическое значение (доступно только на чтение), указывающее, отправлены ли заголовки ответа клиенту или нет;
- statusCode – число, статус ответа клиенту (например, 200 или 404 и т.п.);
- statusMessage – строка, статус ответа клиенту.

Экземпляр ServerResponse

Пример методов:

- on(eventName, listener) или addListener(eventName, listener) – задание функции обработчика listener на указанное событие eventName;
- end([data][, encoding][, callback]) – завершает ответ сервера на запрос клиента (!!!Должен быть обязательно вызван для каждого ответа). Параметры data – строка или буфер с данными, encoding – кодировка, callback – функция обратного вызова сработает после полной отправки сообщения;
- getHeader(name) – возвращает значения заголовка name, который был поставлен в очередь на отправку, но ещё не отправлен клиенту;
- getHeaders(name) – возвращает копию объекта с текущими исходящими заголовками ответа вида ключ (название заголовка в нижнем регистре) / значение (значение заголовка);

Экземпляр ServerResponse

Пример методов:

- hasHeader(name) – возвращает true, если заголовок, идентифицированный по имени name, в настоящее время установлен в исходящих заголовках;
- removeHeader(name) – удаляет заголовок, который стоит в очереди на отправку;
- setHeader(name, value) – устанавливает (обновляет ранее установленный) заголовок name со значением value в очередь на отправку;
- write(chunk[, encoding][, callback]) – отправляет часть тела ответа chunk (строка в указанной кодировки encoding или буфер) клиенту. Может быть вызван несколько раз до вызова метода end. Если write предшествует вызову метода writeHead, то код статуса ответа будет вычислен автоматически и ранее установленные заголовки будут отправлены. Первый вызов write отправляет заголовки ответа и часть тела, поэтому после этого отправить новые заголовки не получится.

Экземпляр ServerResponse

Пример методов:

❖ writeHead(statusCode[, statusMessage][, headers]) – отправляет заголовок ответа серверу. Параметры: statusCode – число, статус ответа; statusMessage – строка, статус ответа; headers - объекта с исходящими заголовками ответа вида ключ (название заголовка в нижнем регистре) / значение (значение заголовка) будет объединён с ранее добавленными заголовками. Этот метод должен быть вызван только один раз на ответ, и он должен быть вызван до методов write и end.

Примеры событий eventName:

❖ 'close' – событие генерируется если происходит разрыв соединения до вызова метода end;

❖ 'finish' – событие генерируется как только ответ клиенту завершён и отправлен.

Полезные ссылки

https://nodejs.org/dist/latest-v10.x/docs/api/http.html – раздел официальной документации на английском языке про модуль HTTP;
https://js-node.ru/site/article?id=25 – раздел документации на русском языке про модуль HTTP (не полностью актуальна нужно соотносить с официальной документацией);