Урок N3

Сетевые запросы

Как уже упоминалось ранее в курсе, в Node.js прекрасно организована работа с сетевыми запросами. Node.js легко обрабатывает большое количество соединений и быстро выполняет входящие и исходящие запросы. В связи с этим классическая ниша применения Node.js - серверные приложения с простой бизнес-логикой, которые должны работать с большим количеством соединений (чаты, онлайн-игры, системы статистики и т.п.)

Содержание урока

- Исходящие запросы с помощью модуля http
- Исходящих запросы с помощью модуля request
- Отправка почты с помощью модуля nodemailer
- Простейший веб-сервер для обработки входящих запросов
- Пример: программа-переводчик (с помощью Google Translate)
- Инструменты для отладки сетевого взаимодействия

Стандартный модуль **http** содержит функцию **get** для отправки GET запросов и функцию **request** для отправки POST и прочих запросов.

Пример отправки GET запроса:

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

http.get("http://www.google.com/index.html", function(res) {
        console.log("Got response: " + res.statusCode);
}).on('error', function(e) {
        console.log("Got error: " + e.message);
});
```

Пример отправки POST запроса:

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

var options = {
        hostname: 'www.google.com',
        port: 80,
        path: '/upload',
        method: 'POST'
};

var req = http.request(options, function(res) {
        console.log('STATUS: ' + res.statusCode);
        console.log('HEADERS: ' + JSON.stringify(res.headers));
        res.setEncoding('utf8');
        res.on('data', function (chunk) {
                console.log('BODY: ' + chunk);
        });
});

req.on('error', function(e) {
        console.log('problem with request: ' + e.message);
});

req.write('data\n');
req.write('data\n');
req.end();
```

Один из самых популярных и удобных npm модулей для работы с исходящими сетевыми запросами - request.

Пример отправки GET запроса:

```
var request = require('request');
request('http://www.google.com', function (error, response, body) {
        if (!error && response.statusCode == 200) {
                console.log(body) // Print the google web page.
        }
});
```

Пример отправки POST запроса:

```
var request = require('request');
request({
        method: 'POST',
        uri: 'http://google.com',
        form:{key: 'value'},
}, function (error, response, body) {
        if (error) {
                console.error(error);
        } else {
                console.log(body);
                console.log(response.statusCode);
        }
});
```

Полезные возможности модуля:

- автоматическая обработка JSON
- работа с учетом редиректов или без них через опцию followRedirects
- поддержка Basic Auth через опцию auth
- поддержка OAuth через опцию oauth
- поддержка прокси через опцию proxy
- поддержка cookies через опцию jar: true

Таким образом **request** является универсальным "швейцарским ножом" для удобной отправки любых исходящих сетевых запросов и поэтому настоятельно рекомендуется к изучению.

Для отправки email крайне удобно использовать популярный npm модуль nodemailer. Он позволяет отправлять почту через SMTP, sendmail, Amazon SES, поддерживает различные виды авторизации и кодировку UTF-8 и.

Пример отправки письма через SMTP:

```
var nodemailer = require("nodemailer");

// create reusable transport method (opens pool of SMTP connections)
var smtpTransport = nodemailer.createTransport("SMTP",{
        service: "Gmail",
        auth: {
                user: "gmail.user@gmail.com",
                pass: "userpass"
        }
});

// setup e-mail data with unicode symbols
var mailOptions = {
        from: "Fred Foo ✔ <foo@blurdybloop.com>",
        to: "bar@blurdybloop.com, baz@blurdybloop.com",
        subject: "Hello ✔", // Subject line
        text: "Hello world ✔", // plaintext body
        html: "<b>Hello world ✔</b>" // html body
}

// send mail with defined transport object
smtpTransport.sendMail(mailOptions, function(error, response){
        if(error){
                console.log(error);
        } else {
                console.log("Message sent: " + response.message);
        }

        // shut down the connection pool, no more messages
        //smtpTransport.close();
});
```

Для обработки входящих TCP запросов в стандартном модуле **http** есть функция **createServer**, которая создает сервер на заданном порту.

```
var http = require("http");

function onRequest(request, response) {
        response.writeHead(200, {"Content-Type": "text/plain"});
        response.write("Hello World");
        response.end();
}

http.createServer(onRequest).listen(8888);
console.log("Server has started.");
```

Все что нам нужно сделать - написать функцию-обработчик для входящих запросов **request** и использовать методы объекта **response** для формирования ответов.

Свойства request описаны в API http.IncomingMessage:

- method
- url
- headers

Свойства и методы response описаны в API http.ServerResponse:

- writeHead - для отправки статуса и заголовков
- statusCode - для установки статуса ответа
- setHeader - для установки заголовка
- write - для отправки данных
- end - для завершения запроса

Так как в GET запросах параметры передаются через url, для их обработки вы должны проанализировать эту строку. Удобно сделать это с помощью стандартного модуля **url** и его функции **parse.** Обратите внимание на то, что для автоматического преобразования GET параметров в объект с парами ключ-значение нужно вторым параметром этой фунции указать true.

```
var urlutils = require('url');

var params = urlutils.parse(
        'http://user:pass@host.com:8080/p/a/t/h?query=string#hash', true
);

console.dir(params);
```

С помощью этого модуля удобно проделать и обратную операцию: "собрать" из частей полный URL:

```
console.log(urlutils.format(params));
```

Если вы хотите задать search часть URL (ту, которая идет после знака ?) с помощью строки, используйте ключ search. Если вы хотите задать GET параметры с помощью объекта, используйте ключ query и убедитесь в том, что в объекте нет ключа search!

```
var urlutils = require('url');

// разбиваем URL на части
var params = urlutils.parse(
        'http://user:pass@host.com:8080/p/a/t/h?query=string#hash', true
);
console.dir(params);

// собираем URL с заменой параметров
delete params.search;
params.query = {key1: 'value1', key2: 'value2'};

console.dir(urlutils.format(params));
```

Все данные из POST запросов получаются асинхронно и вы сами решаете, когда и сколько данных обрабатывать. Для этого http.IncomingMessage реализует интерфейс Readable Stream, который является потоком входящих данных. Чтобы прочитать все данные из тела POST запроса, нужно добавить дополнительные команды:

```
function onRequest(request, response) {
	var postData = "";
	var pathname = url.parse(request.url).pathname;
	console.log("Request for " + pathname + " received.");

	request.setEncoding("utf8");

	request.addListener("data", function(chunk) {
		postData += chunk;
		console.log("POST data chunk '" + chunk + "'.");
	});

	request.addListener("end", function() {
		route(handle, pathname, response, postData);
	});
}
```

Для работы с HTML страницами на клиенте часто удобно использовать библиотеку jQuery с ее удобными селекторами выборки данных. В node.js у вас тоже есть такая возможность! npm модуль **cheerio** содержит аналогичные jQuery функции для работы с DOM на стороне сервера. Вы можете загрузить DOM из произвольной строки с помощью метода **load** и затем использовать привычные $-функции как в jQuery.

```
var request = require('request');
var cheerio = require('cheerio');

request('http://www.rbc.ru/', function (error, response, html) {
        if (!error && response.statusCode == 200) {
                var $ = cheerio.load(html);
                $('#js-rateContainer tr').each(function(i, element){
                        var cols = $(this).find('td');
                        console.dir(
                                cols.eq(0).text()
                                + " " + cols.eq(1).text()
                                + " " + cols.eq(2).text()
                        );
                });
        }
});
```

Если вам нужна эмуляция запуска JavaScript в рамках страницы (например, для выполнения скриптов для динамической подгрузки и отображения содержимого) - можно воспользоваться модулем **zombie**.

Если вам нужна полноценная эмуляция Google Chrome с возможностью рендеринга страниц и сохранения результатов рендеринга в графическом виде - можно воспользоваться модулем **phantomjs**.

Самоконтроль

✓ Модули работы с исходящими запросами
✓ Стандартный модуль для создания веб-сервера
✓ Модули для обработки входящих данных
✓ Отправка email

Домашнее задание

1) Создать программу для получения информации о последних новостей с выбранного вами сайта в структурированном виде.

2) Создать переводчик слов с английского на русский, который будет обрабатывать входящие GET запросы и возвращать ответы, полученные через API Яндекс.Переводчика.

Ссылка для получения ключа API Яндекс.Переводчика:

http://api.yandex.ru/key/form.xml?service=trnsl

Документация API Переводчика:

http://api.yandex.ru/translate/doc/dg/reference/translate.xml

Пример GET запроса к API:

https://translate.yandex.net/api/v1.5/tr.json/translate?key={сюда-подставить-ключ}&lang=ru-en