**«УТВЕРЖДАЮ»**

**Зав. кафедрой ИСиТ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Смелов**

**Экзаменационные вопросы дисциплины**

**«Программирование серверных кроссплатформенных приложений»**

**для студентов 3-го курса специальности ПОИТ**

**В билете 3 вопроса: 1 и 2 вопросы из списка (1-57), 3-й вопрос – демонстрация одной из лабораторных работ (1-16).**

**На экзамене студент обязан предоставить все выполненные (1-16) лабораторные работы. Студент, который не предоставит полный список выполненных лабораторных работ автоматически получает неудовлетворительную оценку.**

1. Протокол HTTP, основные свойства HTTP, структура запроса и ответа. Протокол HTTPS. Понятие web-приложения, структура и принципы работы web-приложения. Понятие асинхронности.
2. **HTTP:** основные свойства

* версии HTTP/1.1 – действующий (текстовый), HTTP/2 – черновой (не распространен, бинарный);
* два типа абонентов: клиент и сервер;
* два типа сообщений: request и response;
* от клиента к серверу – request;
* от сервера к клиенту – response;
* на один request всегда один response, иначе ошибка;
* одному response всегда один request, иначе ошибка;
* TCP-порты: 80, 443;
* для адресации используется URI или URN;
* поддерживается W3C, описан в нескольких RFC.

1. **Request**:

* метод;
* URI;
* версия протокола (HTTP/1.1);
* заголовки (пары: имя/заголовок);
* параметры (пары: имя/заголовок);
* расширение.

1. **Response:**

* версия протокола (HTTP/1.1);
* код состояния (1xx, 2xx, 3xx, 4xx, 5xx);
* пояснение к коду состояния;
* заголовки (пары: имя/заголовок);
* расширение.
* **1xx**: информационные сообщения;
* **2xx**: успешный ответ;
* **3xx**: переадресация;
* **4xx**: ошибка клиента;
* **5xx**: ошибка сервера.

клиент-серверное приложение, у которого клиент и сервер взаимодействуют по протоколу HTTP называется **web-приложением**.

**ПСКП:** когда говорят о разработке web-приложения, говорят о разработке **frontend** (клиента) и **backend** (сервера)



**узел Интернет:** устройство, имеющее IP-адрес и подключенное сети Интернет (обычно к сети Интернет-провайдера). Каждый узел характеризуется своей программно-аппаратной платформой – аппаратурой и операционной системой

**Асинхронные запросы:** запрос, при котором поток, выдавший http-запрос, не блокируется до поступления запроса; для обработки ответа применяется функция обратного вызова.

1. Протокол WebSockets, основные свойства, процедура установки соединения. WebSockets API.

**WebSocket** — протокол связи поверх [TCP](https://ru.wikipedia.org/wiki/TCP)-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

Протокол WebSocket (стандарт [RFC 6455](http://tools.ietf.org/html/rfc6455)) предназначен для решения любых задач и снятия ограничений обмена данными между браузером и сервером. Протокол WebSocket работает над TCP.

Это означает, что при соединении браузер отправляет по HTTP специальные заголовки, спрашивая: «поддерживает ли сервер WebSocket?».

Если сервер в ответных заголовках отвечает «да, поддерживаю», то дальше HTTP прекращается и общение идёт на специальном протоколе WebSocket, который уже не имеет с HTTP ничего общего.

##### Handshake

При подключении посредством вебсокетов происходит обмен заголовками наподобие заголовков HTTP, так называемый handshake или по-нашему «рукопожатие».  
Клиент отправляет заголовок подобного содержания:

GET /chat HTTP/1.1  
Host: server.example.com  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==  
Origin: [example.com](http://example.com/)  
Sec-WebSocket-Protocol: chat, superchat  
Sec-WebSocket-Version: 13

На что сервер должен ему ответить:

HTTP/1.1 101 Switching Protocols  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=  
Sec-WebSocket-Protocol: chat

Так написано в литературе (The WebSocket Protocol RFC 6455). Казалось бы что сложного: получил — ответил. Но здесь у меня возникли первые проблемы. Сервер получал заголовок от клиента, отвечал на него, но клиент не реагировал, вне зависимости от клиента (в данном случае — браузера). Пробовал все, на что хватило мозга, ничего не помогало. Подсказка была найдена [здесь](http://forum.ixbt.com/topic.cgi?id=26:41693). Смысл моей ошибки заключался в том, что браузер принимает заголовок с завершающей пустой строкой, а так как я ее не отправлял (ну не нашел в документации про это ни слова), то браузер продолжал ждать заголовок, и событие «вебсокет подключен (WebSocket.onopen)» в браузере не происходило. В итоге мой ответ выглядел следующим образом:

$answer = "HTTP/1.1 101 Switching Protocols\r\n"

."Upgrade: websocket\r\n"

."Connection: Upgrade\r\n"

."Sec-WebSocket-Accept: ".$hash."\r\n"

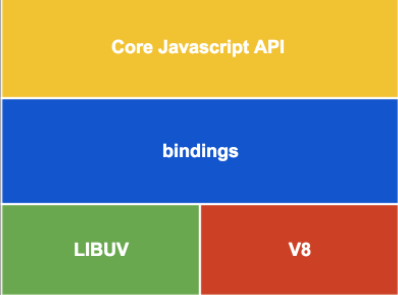
."Sec-WebSocket-Protocol: chat\r\n\r\n"

И клиент наконец его узрел.

1. Платформа Node.js, версии, назначение, основные свойства, структура, принципы работы, основные встроенные модули и их назначение, применение внешних модулей (пакетов). Web-приложение «Hello World». Пример.

**NODEJS:** программная платформа для разработки серверных web-приложений на языке JS/V8.

* основан на **Chrome V8**;
* **среда (контейнер) исполнения** приложений на JavaScript;
* поддерживает механизм **асинхронности**;
* ориентирован на **события**;
* **однопоточный** (код приложения исполняется только в одном потоке, один стек вызовов); обычно в серверах для каждого соединения создается свой поток, в Node.js все соединения обрабатываются в одном JS-потоке;
* **не блокирует** выполнение кода при вводе/выводе (в файловой системе до 4х одновременно);
* в состав Node.js входят инструменты: **npm** – пакетный менеджер; **gyp** - Python-генератор проектов; **gtest** – Google фреймворк для тестирования С++ приложений;
* использует библиотеки: **V8** – библиотека V8 Engine, **libuv** – библиотека для абстрагирования неблокирующих операций ввода/вывода; **http-parser** – легковесный парсер http-сообщений (написан на C и не выполняет никаких системных вызовов); **c-ares** -библиотека для работы с DNS; **OpenSSL** – библиотека для криптографии; **zlib** – сжатие и распаковка.

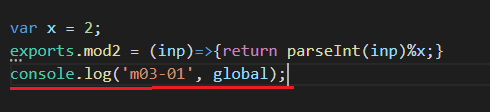


* первая версия: **2009 г**.;
* стабильные версии: с **2015 г., Node.js 4.0.0;**

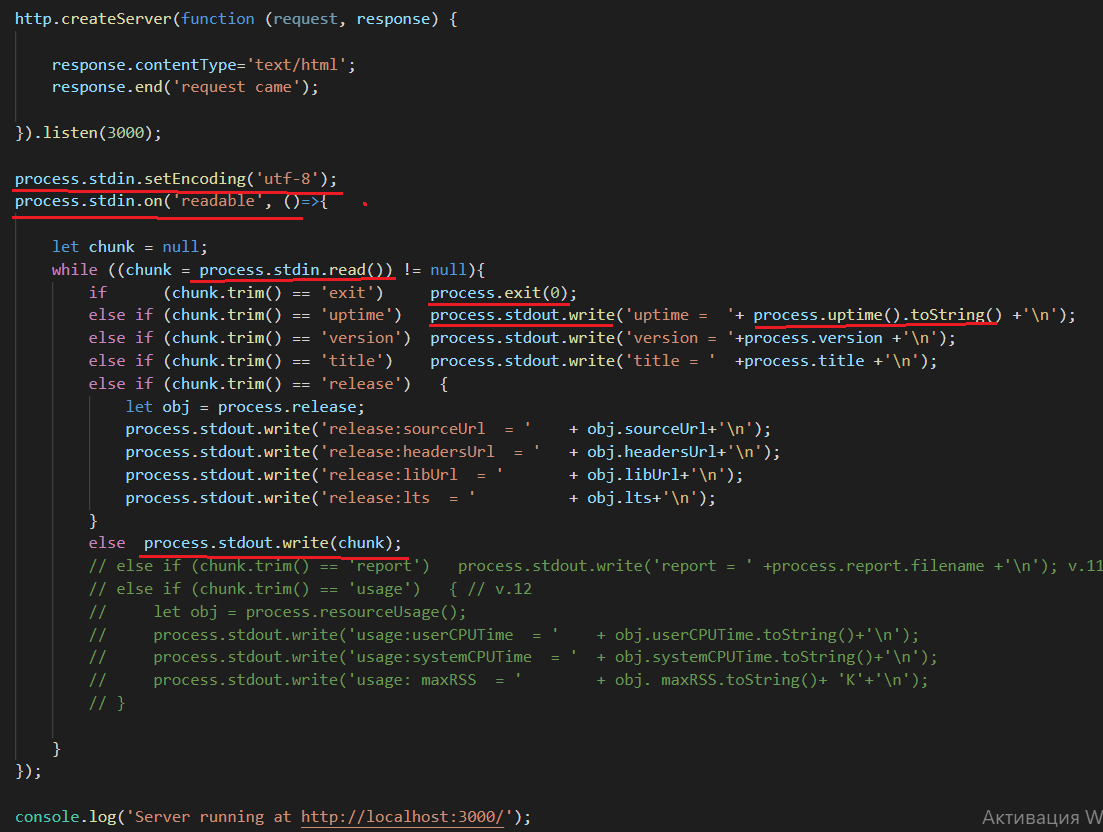


1. Глобальные объекты Node.js (global, process) и их применение. Системные (стандартные потоки) Node.js (stdin, stdout, stderr) и их применение. Модуль console: функции log, error, dir, time, timeEnd, trace. Примеры.

**global: *хранит var-данные на уровне модуля***.



**process:** информация о среде выполнения

****

function foo() {

function bar() {

console.trace();

}

bar();

}

foo();

В консоли следующий трэйс будет отображен так:

bar

foo

<anonymous>

1. Класс EventEmitter, назначение, применение. Пример.
2. Функции setTimeout, setInterval, nextTick, ref, unref, назначение, применение. Примеры.
3. Модули и пакеты Node.js, функция require, кэширование модуля, область видимости в пакете, экспорт объектов, функций, конструкторов. Применение require для работы с json-файлами. Параметризируемый модуль. Пример.
4. Пакетный менеджер NPM, глобальное хранилище, просмотр установленных пакетов, скачивание пакетов, назначение файла package.json, локальные хранилища пакетов, удаление пакетов, публикация пакета. Примеры.
5. Разработка простейшего HTTP-сервера в Node.js. Извлечение данных из HTTP-запроса, формирование данных HTTP-ответа. Пример. Тестирование с помощью POSTMAN.
6. Разработка простейшего HTTP-сервера в Node.js. Извлечение данных из HTTP-запроса, формирование данных HTTP-ответа. Пример. Тестирование с помощью браузера AJAX (XMLHTTPRequest/Fetch).
7. Разработка HTTP-сервера в Node.js. Обработка GET, POST, PUT и DELETE-запросов. Генерация ответа с кодом 404. Пример. Тестирование с помощью POSTMAN.
8. Разработка HTTP-сервера в Node.js. Обработка URI HTPP-запроса, маршрутизация запросов, генерация ответа с кодом 404. Пример. Тестирование с помощью POSTMAN.
9. Разработка HTTP-сервера в Node.js. Обработка запросов к статическим ресурсам: html, css, js, png, msword. Пример. Тестирование с помощью браузера.
10. Разработка HTTP-сервера в Node.js. Обработка query-параметров GET-запроса. Пример. Тестирование с помощью браузера.
11. Разработка HTTP-сервера в Node.js. Обработка uri-параметров GET-запроса. Пример. Тестирование с помощью браузера.
12. Разработка HTTP-сервера в Node.js. Обработка параметров POST-запроса. Пример. Тестирование с помощью браузера (<form>) и POSTMAN.
13. Разработка HTTP-сервера в Node.js. Обработка json-сообщения в POST-запросе. Пример. Тестирование с помощью POSTMAN.
14. Разработка HTTP-сервера в Node.js. Обработка xml-сообщения в POST-запросе. Пример. Тестирование с помощью POSTMAN.
15. Разработка HTTP-сервера в Node.js. Пересылка файла в POST-запросе (upload). Пример. Тестирование с помощью браузера.
16. Разработка HTTP-сервера в Node.js. Пересылка файла в ответе (download). Пример. Тестирование с помощью браузера.
17. Разработка HTTP-клиента в Node.js. Оправка GET запроса с query-параметрами. Пример. Тестирование с помощью с Node.js-сервера.
18. Разработка HTTP-клиента в Node.js. Оправка POST-запроса с параметрами в теле. Пример. Тестирование с помощью с Node.js-сервера.
19. Разработка HTTP-клиента в Node.js. Оправка POST-запроса с json-сообщением. Пример. Тестирование с помощью с Node.js-сервера.
20. Разработка HTTP-клиента в Node.js. Обработка json-ответа. Пример. Тестирование с помощью с Node.js-сервера.
21. Разработка HTTP-клиента в Node.js. Обработка xml-ответа. Пример. Тестирование с помощью с Node.js-сервера.
22. Разработка HTTP-клиента в Node.js. Пересылка файла на сервер в POST-запросе (upload). Пример. Тестирование с помощью с Node.js-сервера.
23. Разработка HTTP-клиента в Node.js. Обработка ответа с файлом (download). Пример. Тестирование с помощью с Node.js-сервера.
24. Разработка Websockets-приложения: Node.js-сервер, браузер-клиент. Пример.
25. Разработка широковещательного Websockets-приложения: Node.js-сервер, Node.js-клиент. Пример.
26. Разработка Websockets-приложения: Node.js-сервер с применением потока, Node.js-клиент. Пример.
27. Разработка Websockets-приложения: ping/pong-сообщения, Node.js-сервер, Node.js-клиент. Пример.
28. Разработка Websockets-приложения: обработка json-сообщений, Node.js-сервер, Node.js-клиент. Пример.
29. Разработка Websockets-приложения: отправка клиентом файла (upload), Node.js-сервер, Node.js-клиент. Пример.
30. Разработка Websockets-приложения: отправка сервером файла (download), Node.js-сервер, Node.js-клиент. Пример.
31. Разработка RPC-Websockets-сервера. Пример. Тестирование: Node.js-клиент.
32. Разработка RPC-Websockets-сервера: обработка уведомлений. Пример. Тестирование: Node.js-клиент.
33. Работа с файловой системой в Node.js: создание, копирование, проверка существования файла, запись, запись в конец, чтение, синхронные асинхронные операции. Пример.
34. Работа с файловой системой в Node.js: создание, удаление, переименование, запись, запись в конец, чтение, синхронные асинхронные операции. Пример.
35. Работа с файловой системой в Node.js: создание, слежение за файлом, запись, запись в конец, чтение, синхронные асинхронные операции. Пример.
36. Работа с файловой системой в Node.js: запись в файл потока октетов, чтение из файла потока октетов. Пример.
37. Работа с файловой системой в Node.js: запись в файл массива 32-битовых целочисленных данных, чтение из файла массива 32-битовых целочисленных данных. Пример.
38. Применение потокового чтение (Readable) и записи (Writable) файлов в Node.js. Пример.
39. Применение функции pipe для обработки данных (файла) запроса и записи в файл файловой системы. Пример.
40. Применение функции pipe для обработки данных (файла) файловой системы и записи в http-ответ. Пример.
41. Разработка клиент-серверного TCP-приложения: обмен текстовыми сообщениями. Пример.
42. Разработка клиент-серверного TCP-приложения: пересылка массива целочисленных данных. Пример.
43. Разработка клиент-серверного TCP-приложения: пересылка файла от клиента серверу. Пример.
44. Разработка клиент-серверного TCP-приложения: пересылка файла от сервера клиенту. Пример.
45. Разработка клиент-серверного TCP-приложения прослушивающего два порта, обмен текстовыми сообщениями. Пример.
46. Разработка клиент-серверного UDP-приложения: обмен текстовыми сообщениями. Пример.
47. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического SELECT-запроса.

1. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического INSERT-запроса. Пример.
2. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического UPDATE-запроса. Пример.
3. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического DELETE-запроса. Пример.
4. Разработка приложения, выполняющего запрос к SQL-базе данных: вызов удаленной процедуры. Пример.

1. Разработка приложения, выполняющего graphql-запрос к SQL-базе данных: query-запрос. Пример.
2. Разработка приложения, выполняющего graphql-запрос к SQL-базе данных: mutation-запрос. Пример.

Доцент каф. ИСиТ В.В. Смелов