

Экзаменационные вопросы дисциплины

«Программирование серверных кроссплатформенных приложений»
для студентов 3-го курса специальности ПОИТ1ый вопрос в билете

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

HTTP является протоколом прикладного уровня.
Свойства - расширяемый, без состояния, прост.

структура

Стартовой строки, описывающей запрос, или статус (успех или сбой). Это всегда одна строка.

Произвольного набора HTTP заголовков, определяющих запрос или описывающих тело сообщения.

Пустой строки, указывающей, что вся мета информация отправлена.

Произвольного тела, содержащего пересылаемые с запросом данные (например, содержимое HTML-формы) или отправляемый в ответ документ. Наличие тела и его размер определяется стартовой строкой и заголовками HTTP.

Существует множество заголовков запроса. Их можно разделить на несколько групп:

- Основные заголовки (General headers), например, [Via \(en-US\)](#), относящиеся к сообщению в целом
- Заголовки запроса (Request headers), например, [User-Agent](#), [Accept-Type](#), уточняющие запрос (как, например, [Accept-Language](#)), придающие контекст (как [Referer](#)), или накладывающие ограничения на условия (like [If-None](#)).
- Заголовки сущности, например [Content-Length](#), относящиеся к телу сообщения. Как легко понять, они отсутствуют, если у запроса нет тела.

Протокол https - защищенный http (<https://habr.com/ru/post/258285/>)

Web-приложение - приложение, работающее на протоколе http. Структура - клиент-сервер.

Асинхронность - выполнение задачи в неблокирующем режиме. Отправка запроса на выполнение и получение ответа в будущем.







Операция, выполняющаяся в 2 этапа - первый это заявка, а второй - выполнение.

Сокет - ip+port

2. HTTP-аутентификация (Basic, Digest, Forms).

Аутентификация - проверка подлинности. В вебе - проверка подлинности пользователя путём сравнения введённого им пароля (для указанного [логина](#)) с паролем, сохранённым в [базе данных](#) пользовательских логинов;

Типы аутентификации

- **Basic** (смотреть [RFC 7617](#) , зашифрованные с помощью base64 учётные данные. Больше информации смотреть снизу.),
- **Bearer** (смотреть [RFC 6750](#) , bearer токены для доступа OAuth 2.0-защищённых ресурсов),
- **Digest** (смотреть [RFC 7616](#) , Firefox 93 и более поздние версии поддерживают шифрование SHA-256. Предыдущие версии поддерживают только хэширование MD5 (не рекомендуется).),
- **HOBA** (смотреть [RFC 7486](#) , Секция 3, HTTP Origin-Bound Authentication, digital-signature-based),
- **Mutual** (смотреть [draft-ietf-httpauth-mutual](#) ,),
- **AWS4-HMAC-SHA256** (смотреть [AWS документацию](#) ,).

Стандартизованные Basic и Digest

в Basic пароль открыт, в Digest зашифрован в md5

Forms - нестандартизован, передаётся в форме.

3. Протокол HTTPS. Протокол TLS. Сертификаты. Взаимодействие центра сертификации и владельца защищенного ресурса.

Центр сертификации выдаёт сертификат владельцу ресурса. В нём указывается информация о ресурсе, домены на которые он распространяется. Есть различные стоимости сертификатов, достигающие нескольких тысяч долларов. Информация о центрах сертификации встроена в браузер изначально.

(<https://habr.com/ru/post/258285/>)

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

Rf 6455 описывает дуплексный протокол websockets работающий over http.

The handshake from the client looks as follows:

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

The handshake from the server looks as follows:

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

В ответном хендшейке возвращается статус 101 - смена протоколов.

особенности - позволяет передавать данные без дополнительных запросов от клиента.

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

Websocket api позволяет открывать соединения, писать и получать сообщения и закрывает соединение.

5. Разработка простейшего HTTP-сервера в Node.js. Извлечение данных из HTTP-запроса, формирование данных HTTP-ответа. Пример. Тестирование с помощью браузера AJAX (XMLHttpRequest/Fetch).

6. Разработка HTTP-сервера в Node.js. Обработка GET, POST, PUT и DELETE-запросов. Генерация ответа с кодом 404. Пример. Тестирование с помощью POSTMAN.
7. Разработка HTTP-сервера в Node.js. Обработка запросов к статическим ресурсам: html, css, js, png, msword. Пример. Тестирование с помощью браузера.
8. Разработка HTTP-сервера в Node.js. Обработка query-параметров GET-запроса. Пример. Тестирование с помощью браузера.
9. Разработка HTTP-сервера в Node.js. Обработка uri-параметров GET-запроса. Пример. Тестирование с помощью браузера.
10. Разработка HTTP-сервера в Node.js. Обработка параметров POST-запроса. Пример. Тестирование с помощью браузера (<form>) и POSTMAN.
11. Разработка HTTP-сервера в Node.js. Обработка json-сообщения в POST-запросе. Пример. Тестирование с помощью POSTMAN.
12. Разработка HTTP-сервера в Node.js. Пересылка файла в POST-запросе (upload). Пример. Тестирование с помощью браузера.
13. Разработка HTTP-сервера в Node.js. Пересылка файла в ответе (download). Пример. Тестирование с помощью браузера.
14. Разработка HTTP-клиента в Node.js. Оправка GET запроса с query-параметрами. Пример. Тестирование с помощью с Node.js-сервера.
15. Разработка HTTP-клиента в Node.js. Оправка POST-запроса с параметрами в теле. Пример. Тестирование с помощью с Node.js-сервера.

16. Разработка HTTP-клиента в Node.js. Оправка POST-запроса с json-сообщением. Пример. Тестирование с помощью с Node.js-сервера.
17. Разработка HTTP-клиента в Node.js. Обработка json-ответа. Пример. Тестирование с помощью с Node.js-сервера.
18. Разработка HTTP-клиента в Node.js. Пересылка файла на сервер в POST-запросе (upload). Пример. Тестирование с помощью с Node.js-сервера.
19. Разработка HTTP-клиента в Node.js. Обработка ответа с файлом (download). Пример. Тестирование с помощью с Node.js-сервера.
20. Разработка Websockets-приложения: Node.js-сервер, браузер-клиент. Пример.
21. Разработка Websockets-приложения: обработка json-сообщений, Node.js-сервер, Node.js-клиент. Пример.
22. Разработка RPC-Websockets-сервера. Пример. Тестирование: Node.js-клиент.
23. Применение функции pipe для обработки данных (файла) файловой системы и записи в http-ответ. Пример.
24. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического SELECT-запроса (лабораторная работа).
25. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического INSERT-запроса. Пример (лабораторная работа).

26. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического UPDATE-запроса. Пример (лабораторная работа).
27. Разработка приложения, выполняющего запрос к SQL-базе данных: выполнение динамического DELETE-запроса. Пример.

2ый вопрос в билете

28. Применение СУБД Redis. Основные принципы работы. Пример (лабораторная работа).
29. Применение пакета Sequelize. Основные принципы работы. Пример (лабораторная работа).
30. Пакет Express. Основные принципы работы. Middleware-код. Пример.
31. Пакет Express. Основные принципы работы. Маршрутизация. Пример.
32. Пакет Express. Основные принципы работы. Статические файлы. Пример.
33. Пакет Express. Основные принципы работы. Обработка query-параметров GET-запроса. Пример (POSTMAN).
34. Пакет Express. Основные принципы работы. Обработка uri-параметров запроса. Пример (POSTMAN).
35. Пакет Express. Основные принципы работы. Обработка body-параметров POST-запроса. Пример (POSTMAN).
36. Пакет Express. Основные принципы работы. Обработка json-данных POST-запроса. Пример (POSTMAN).

37. Пакет Express. Основные принципы работы. Обработка xml-данных POST-запроса. Пример (POSTMAN).
38. Пакет Express. Основные принципы работы. download/attachment файлы GET-запроса. Пример (браузер).
39. Пакет Express. Основные принципы работы. upload файла в POST-запросе. Пример (браузер).
40. Пакет Express. Основные принципы работы. Обработка Cookie. Signed cookie. Пример (POSTMAN).
41. Пакет Express. Основные принципы работы. Применение объекта Session для сохранения состояния. Пример (POSTMAN).
42. Пакет Express. Основные принципы работы. Переадресация. Пример (POSTMAN).
43. Пакет Express. Основные принципы работы. Выполнение shell-команд (spawn, pipe). Пример.
44. Пакет Express. Основные принципы работы. Запуск процесса операционной системы (exec), работа со стандартными потоками ввода/вывода. Пример.
45. Пакет Express. Основные принципы работы. Выполнение js-скриптов в отдельном процессе (fork, send, worker). Пример.
46. Протокол WebDav. Разработка приложения с применением WebDav. Пример (лабораторная работа).
47. Протокол JSON-RPC. Разработка клиент-серверное приложение использующее протокол JSON-RPC.

48. Разработка клиент-серверного приложения с применением технологии WebAssembly на стороне браузера. Пример (WasmFiddle-компиляция).

49. Разработка клиент-серверного приложения с применением технологии WebAssembly на стороне сервера Node.js. Пример (WasmFiddle-компиляция).

50. Long pool-сервер, принцип работы. Пример (Telegram bot, лабораторная работа).

3ый вопрос в билете (лабораторные работы)

51. Лабораторная 17 (REDIS).

52. Лабораторная 18 (SEQUELIZE).

53. Лабораторная 20 (HBS).

54. Лабораторная 21 (Basic, Digest, Forms).

55. Лабораторная 22 (HTTPS).

56. Лабораторная 23 (CRYPTO).

57. Лабораторная 24 (WEBDAV).

58. Лабораторная 25 (JSONRPC).

59. Лабораторная 26 (WASM).

60. Лабораторная 27 (TLGBOT).

Доцент каф. ИСиТ

В.В. Смелов