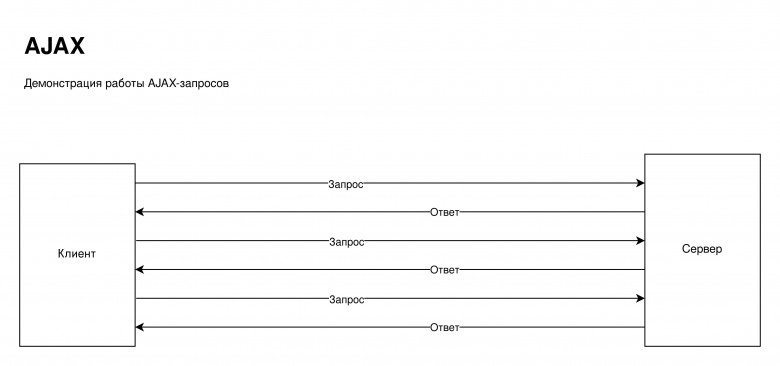
# **СТРВП ОТВЕТЫ ЛАБ1Б**

**AJAX** - асинхронные запросы с помощью JavaScript (Asynchonous JavaScript and XML). AJAX преследует все те же цели, что и HTTP, только делает это уже асинхронно. Если ранее нужно было для каждого запроса прописывать свой URL и перезагружать страницу, то теперь можно просто использовать AJAX и он сам будет отправлять нужные URL серверу и получать данные.



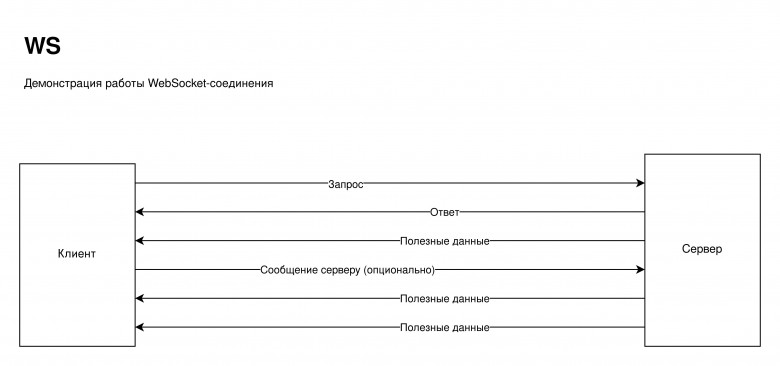
**Особенности AJAX**

* Все ещё обычный запрос, который не поддерживает соединение, после того, как отдает ответ на запрос.
* Все ещё заранее оговариваем действие, которое клиент хочет сделать в заголовке (HTTP Headers) - GET, POST, PUT, DELETE
* Мы отправляем заголовок что хотим сделать каждый раз, как обраемся к серверу
* Теперь мы делаем это асинхронно благодаря JavaScript

**WebSocket** - протокол (стандарт RFC 6455) для общения между клиентом и сервером, предоставляющий двухсторонне общение сверх протокола TCP.

Если что вебсокет это дуплекс.

Мы подключаем WS один раз, а затем сервер может отдавать нам ответы тогда, когда посчитает нужным. Он позволяет пересылать любые данные, на любой домен, безопасно и почти без лишнего сетевого трафика.



**Как работает:**

1. Обычный запрос TCP на сервер, ждем ответ (рукопожатие).
2. Обмен данными(один из клиентов отправил HTTP-запрос серверу и нужно отдать ответ не только одному клиенту, а целой сети, тогда сервер отдаст обычный ответ отправителю, а другим - пакеты по WebSocket-соединению с полезными данными).

**Установление соединения:**

Протокол WebSocket работает над TCP.

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.

**Запрос от браузера:**

|  |
| --- |
| GET /chat HTTP/1.1  Host: server.example.com  Upgrade: websocket  Connection: Upgrade  Origin: http://javascript.ru  Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==  Sec-WebSocket-Version: 13 |

**Описания заголовков:**

GET, Host

* Стандартные HTTP-заголовки из URL запроса

Upgrade, Connection

* Указывают, что браузер хочет перейти на websocket.

Origin

* Протокол, домен и порт, откуда отправлен запрос.

Sec-WebSocket-Key

* Случайный ключ, который генерируется браузером: 16 байт в кодировке Base64.

Sec-WebSocket-Version

* Версия протокола. Текущая версия: 13.

**HTTP-обработчики** (HTTP Handlers) используются для генерации ответа на HTTP-запросы. Они позволяют настраивать обработку запросов. HTTP-обработчик реализует интерфейс System.Web.IHttpHandler, который включает метод ProcessRequest(HttpContext context) для обработки запроса и свойство IsReusable, указывающее, может ли обработчик использоваться повторно.

**Чем отличается от обычного хттп**

HTTP-обработчик – это конечная точка, которая обслуживает запрос к ресурсу. Фактически, HTTP-обработчик содержит код, который запускается для генерации результата для данного типа запросов.

HTTP – просто запрос, он не обрабатывает входящие HTTP-запросы и не генерирует ответы. HTTP-обработчик имеет логику обработки запросов.

Свойство IsReusable позволяет определить поведение обработчика при обращении к нему. Метод ProcessRequest является входной точкой для обработки запроса. В нем содержится логика по генерации ответа клиенту.

Если HTTP-обработчик выполняет какую-то длительную вычислительную операцию, то есть смысл реализовать асинхронный HTTP-обработчик. Идея асинхронных обработчиков состоит в том, то для выполнения длительной операции используется новый поток, а основном поток возвращается в пул потоков веб-серверов. После завершения выполнения длительной операции из пула потоков заново изымается основной поток, в рамках которого производится передача обработанной информации клиенту.

**Как устанавливается соединение по веб сокету**

Установка соединения по веб-сокету начинается с клиента, который отправляет HTTP-запрос с заголовком Upgrade на сервер для перехода на протокол WebSocket. Сервер, поддерживающий WebSocket, отвечает статусом 101 Switching Protocols, и соединение устанавливается.

**Рукопожатие через заголовки**

Процесс установления соединения WebSocket известен как открывающее рукопожатие и состоит из обмена запросами/ответами HTTP/1.1 между клиентом и сервером. Клиент всегда инициирует рукопожатие; он отправляет запрос GET на сервер, указывая, что он хочет обновить соединение с HTTP на WebSockets. Сервер должен вернуть код ответа протокола коммутации HTTP 101 для установки соединения WebSocket. Как только это произойдет, соединение WebSocket можно будет использовать для постоянной двунаправленной полнодуплексной связи между сервером и клиентом.

В таблице ниже описаны **заголовки**, **используемые** **во время открытия** **рукопожатия**.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 2.1: Открытие заголовков рукопожатия | | | |
| ЗАГОЛОВОК | ОБЯЗАТЕЛЬНОСТЬ | ОПИСАНИЕ |
| Host | Да | Имя хоста и, при необходимости, номер порта сервера, на который отправляется запрос. Если номер порта не указан, подразумевается значение по умолчанию (80 для ws или 433 для wss). |
| Connection | Да | Указывает, что клиент хочет согласовать изменение способа использования соединения. Значение должно быть Upgrade. Также возвращается сервером. |
| Upgrade | Да | Указывает, что клиент хочет обновить соединение до альтернативных средств связи. Значение должно быть websocket. Также возвращается сервером. |
| Sec-WebSocket-Version | Да | Единственное допустимое значение — 13. Любая другая версия, переданная в этом заголовке, недействительна. |
| Sec-WebSocket-Key | Да | Одноразовое случайное значение (nonce) в кодировке Base64, отправленное клиентом. Автоматически обрабатывается большинством библиотек WebSocket или с помощью класса WebSocket, предоставляемого в браузерах. Дополнительные сведения см. в разделах Sec-WebSocket-Key и Sec-WebSocket-Accept в этой главе. |
| Sec-WebSocket-Accept | Да | Хеш-значение SHA-1 в кодировке Base64, возвращаемое сервером как прямой ответ на Sec-WebSocket-Key. Указывает, что сервер готов инициировать соединение WebSocket. Дополнительные сведения см. в разделах Sec-WebSocket-Key и Sec-WebSocket-Accept в этой главе. |
| Sec-WebSocket-Protocol | Нет | Необязательное поле заголовка, содержащее список значений, указывающих, какие подпротоколы хочет использовать клиент, упорядоченные по предпочтениям. Серверу необходимо включить это поле вместе с одним из выбранных значений подпротокола (первым из списка, который он поддерживает) в ответ. Дополнительные сведения см. в разделе «Подпротоколы» далее в этой главе. |
| Sec-WebSocket-Extensions | Нет | Необязательное поле заголовка, первоначально отправляемое от клиента на сервер, а затем впоследствии отправляемое с сервера клиенту. Это помогает клиенту и серверу согласовать набор расширений уровня протокола, которые будут использоваться на протяжении всего соединения. Дополнительные сведения см. в разделе «Расширения» далее в этой главе. |
| Origin | Нет | Поле заголовка, отправляемое всеми браузерными клиентами (необязательно для небраузерных клиентов). Используется для защиты от несанкционированного использования сервера WebSocket из разных источников сценариями, использующими API WebSocket в веб-браузере. Соединение будет отклонено, если указанный Origin неприемлем для сервера. |

**Методы веб сокета**

API WebSocket поддерживает четыре типа событий:

* open
* message
* error
* close

API WebSocket поддерживает два метода: send() и close().

|  |  |
| --- | --- |
| СВОЙСТВО | ОПИСАНИЕ |
| onopen | Вызывается, когда свойство readyState соединения WebSocket изменяется на 1; это означает, что соединение открыто и готово отправлять и получать данные. |
| onmessage | Вызывается при получении сообщения от сервера. |
| onerror | Вызывается при возникновении события ошибки, влияющего на соединение WebSocket. |
| onclose | Вызывается с событием закрытия, когда свойство readyState соединения WebSocket изменяется на 3; это означает, что соединение закрыто. |

**Потоки**

В C# потоки (threads) используются для выполнения параллельных задач, что позволяет создавать более отзывчивые и эффективные приложения.

Нередко программа выполняет такие операции, которые могут занять продолжительное время, например, обращение к сетевым ресурсам, чтение-запись файлов, обращение к базе данных и т.д. Такие операции могут серьезно нагрузить приложение. В синхронном приложении при выполнении продолжительных операций в основном потоке этот поток просто бы блокировался на время выполнения операции. И чтобы продолжительные операции не блокировали общую работу приложения, в C# можно задействовать асинхронность.

**Асинхронность** позволяет вынести отдельные задачи из основного потока в специальные асинхронные методы и при этом более экономно использовать **потоки**. Асинхронные методы выполняются в отдельных потоках. Однако при выполнении продолжительной операции поток асинхронного метода возвратится в пул потоков и будет использоваться для других задач. А когда продолжительная операция завершит свое выполнение, для асинхронного метода опять выделяется поток из пула потоков, и асинхронный метод продолжает свою работу.

Асинхронный; если одно или несколько вычислений может выполняться отдельно от основного потока программы.

В асинхронных методах для остановки метода на некоторое время можно применять метод Task.Delay(). В качестве параметра он принимает количество миллисекунд в виде значения int, либо объект TimeSpan, который задает время задержки. Причем метод Task.Delay сам по себе представляет асинхронную операцию, поэтому к нему применяется оператор await.

**Что такое протокол Web Sockets API?**

В этой главе вы познакомитесь с интерфейсом прикладного программирования (API) WebSocket, который расширяет возможности протокола WebSocket для веб-приложений. API WebSocket обеспечивает управляемую событиями связь через постоянное соединение. Это позволяет создавать веб-приложения, которые действительно работают в режиме реального времени и менее ресурсоемки как на клиенте, так и на сервере по сравнению с методами HTTP.

API WebSocket представляет собой технологию, позволяющую открыть постоянный двусторонний полнодуплексный канал связи между веб-клиентом и веб-сервером. Интерфейс WebSocket позволяет асинхронно отправлять сообщения на сервер и получать ответы на основе событий без необходимости опроса обновлений.

**Какие дополнительные возможности предоставляет Web Sockets API?**

1. **Поддержка текстовых и бинарных данных**:
   * Web Sockets API позволяет отправлять как текстовые сообщения (строки), так и бинарные данные (например, Blob или ArrayBuffer).
2. **Поддержка расширений**:
   * Протокол WebSocket поддерживает расширения, которые могут добавлять дополнительные функции, такие как сжатие данных.
3. **Серверная поддержка**:
   * WebSocket API позволяет устанавливать двусторонние соединения, что означает, что сервер может отправлять сообщения клиенту в любое время.
4. **Подключение через защищенные протоколы**:
   * WebSocket поддерживает защищенные соединения через wss://, что обеспечивает шифрование данных.
5. **Поддержка различных событий**:
   * Возможность обрабатывать события, такие как потеря соединения и повторное подключение, что повышает надежность приложения.
6. **Параллельное соединение**:
   * Можно установить несколько соединений по WebSocket для работы с разными ресурсами одновременно.

**Коды закрытия**

* 1000 – норма;
* 1001 – процесс сервера убит или браузер перешел на другую страницу;
* 1002 – ошибка протокола;
* 1003 – получены данные, которые не может принять.

**PING / PONG**

В протокол встроена проверка связи при помощи управляющих фреймов типа PING и PONG.

Тот, кто хочет проверить соединение, отправляет фрейм PING с произвольным телом. Его получатель должен в разумное время ответить фреймом PONG с тем же телом.

Эта функциональность встроена в браузерную реализацию, так что браузер ответит на PING сервера, но управлять ей из JavaScript нельзя.

Иначе говоря, сервер всегда знает, жив ли посетитель или у него проблема с сетью.

**WSS**

Соединение WebSocket можно открывать как WS:// или как WSS://. Протокол WSS представляет собой WebSocket над HTTPS.

Кроме большей безопасности, у WSS есть важное преимущество перед обычным WS – большая вероятность соединения.

Дело в том, что HTTPS шифрует трафик от клиента к серверу, а HTTP – нет.

Если между клиентом и сервером есть прокси, то в случае с HTTP все WebSocket-заголовки и данные передаются через него. Прокси имеет к ним доступ, ведь они никак не шифруются, и может расценить происходящее как нарушение протокола HTTP, обрезать заголовки или оборвать передачу.

А в случае с WSS весь трафик сразу кодируется и через прокси проходит уже в закодированном виде. Поэтому заголовки гарантированно пройдут, и общая вероятность соединения через WSS выше, чем через WS.

|  |
| --- |
| ﻿<!DOCTYPE html>  <html>  <head>  <meta http-equiv="content-type" content="text/html; charset=utf-8" />  <title>WebSockets</title>  <script src="https://cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.min.js"></script>  </head>  <body>  <header> <h1>Web Sockets</h1> </header>  <section style="width: 800px; float:left">  <div id="wsproperties">  Web Sockets поддерживается?:  <span id="support"></span><br />  </div>  </section>  <br />  <section style="width: 215px;border: solid;">  <div style="margin: 5px">  <input id="bstart" type="button" value="Пуск"  onclick="exe\_start()" style="width: 100px" />  <input id="bstop" type="button" value="Стоп"  onclick="exe\_stop()" style="width: 100px" />  <textarea id="ta" rows="20" cols="25"  style="text-align: center;" readonly>  </textarea>  </div>  </section>  <div style="margin:25px;">  <button id="bclear">Очистить</button>  </div>  <script>  var ta; // окошко с сообщениями  var ws = null; // руль всего  var bstart; // кнопка пуск  var bstop; // кнопка стоп  var bclear = document.getElementById('bclear'); // кнопка чтобы нахуй всё снести  window.onload = function () {  if (Modernizr.websockets) {  WriteMessage("support", "да ");  ta = document.getElementById('ta');  bstart = document.getElementById("bstart");  bstop = document.getElementById("bstop");  bstart.disabled = false;  bstop.disabled = true;  }  }  function WriteMessage(idspan, txt) {  var span = document.getElementById(idspan);  span.innerHTML = txt;  }  function exe\_start() {  if (ws == null) {  // Создаем WS-объект, с помощьюю него будем рулить потоками  // (отправка, принятие запросов)  ws = new WebSocket('ws://localhost:5070/ws'); // открытие соединения с указанием протокола ws  // есть 4 колбэка:  // 1 при получении данных  // 3 при изменении состояния соединений  // onerror – в коде нет, но хай буде есть  /\*  Не забываем, что мы можем отправить сообщение серверу  только если соединение открыто  Поместим все общение с сервером внутрь ивента onopen,  именно он срабатывает, когда соединение открыто  \*/  ws.onopen = function () { ws.send('Соединение'); }  // разрыв соединения  ws.onclose = function (s) { console.log('onclose ', s); }  // Вешаем слушатель на принятие сообщений  ws.onmessage = function (evt) { ta.innerHTML += '+\n' + evt.data; }  bstart.disabled = true;  bstop.disabled = false;  }  }  function exe\_stop() {  ws.close(3001, 'stopapplication'); // закрытие соединеняия ws = null;  bstart.disabled = false;  bstop.disabled = true;  }  bclear.addEventListener('click', function () {  ta = document.getElementById('ta');  ta.innerHTML = "";  })  </script>  </body>  </html> |

<https://akawah.ru/linux/wshb.html>

<https://nodejsdev.ru/guides/webdraftt/websocket/#websocket_1>

<https://medium-com.translate.goog/bina-nusantara-it-division/implementing-websocket-client-and-server-on-asp-net-core-6-0-c-4fbda11dbceb?_x_tr_sl=auto&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_hist=true>

<https://www.internet-technologies.ru/articles/vvedenie-vveb-sokety.html>

<https://habr.com/ru/sandbox/171066/>

<https://habr.com/ru/articles/14246/>

https://intuit.ru/studies/courses/4455/712/lecture/21310?page=2