**Глава 7. ПРОГРАММНЫЙ ИНТЕРФЕЙС WEB SOCKETS API**

**7.1. Назначение Web Sockets API и принципы его работы**

Для построения эффективных web-приложений до появления HTML5 применялась технология AJAX (Asynchronous JavaScript and XML), которая основывалась на возможности доступного из JavaScript объекта XMLHTTPRequest асинхронно отправлять http-запросы серверу и обрабатывать его ответы. В качестве формата передачи данных в AJAX, как правило, используется XML и JSON. Основное достоинство AJAX-приложения заключается в его динамичности: применение AJAX позволяет сократить сетевой трафик и избежать лишних перезагрузок html-страниц.

Основной недостаток AJAX заключается в том, что он основывается на протоколе HTTP, который требует работы в режиме «запрос-ответ». Другими словами, получить порцию данных от сервера можно только сделав к нему запрос. В приложениях, основная задача которых реагировать на события, происходящие на сервере, клиентской части приложения приходится периодически выполнять запросы к серверу, чтобы «увидеть» эти события. Для того, чтобы «не пропустить» серверное событие и своевременно на него отреагировать, приходится увеличивать частоту запросов. Это приводит к загрузке сетевого трафика и нерациональному расходу ресурса сервера, который вынужден постоянно отвечать на присланные клиентом запросы.

Web Sockets API – новый программный интерфейс, основанный на протоколе полнодуплексной связи WebSocket. Протокол работает над TCP-соединением и предназначен для обмена сообщениями между браузером и web-сервером. Протокол WebSocket описан в RFC 6465 [19].

Для установки соединения (процедура websocket handshake) между клиентом и сервером WebSocket использует запрос и ответ в формате протокола HTTP. При этом протокол определяет две URL-схемы: ws – для незашифрованного соединения и wss – для зашифрованного.

Сразу после отправки сервером ответа, TCP-соединение остается открытым, клиент и сервер могут начинать двунаправленный обмен сообщениями по этому же TCP-соединению.

Для разработки серверной компоненты web-приложения, работающей по протоколу WebSocket, программисты могут применять программные интерфейсы, входящие в состав различных технологий разработки web-приложений. Поддержка WebSocket осуществляется технологиями корпорации Microsoft: IIS, ASP.NET и WCF. В состав технологий Java EE тоже входит пакет (javax.websocket), обеспечивающий взаимодействие клиента и Application Server по протоколу WebSocket.

**7.2. Применение Web Sockets API**

Для демонстрации возможностей программного интерфейса Web Sockets API рассмотрим небольшое web-приложение, включающее клиентскую и серверную компоненты применяющие для обмена данными протокол WebSocket.

**7.2.1. Клиент протокола WebSocket**

На рис. 7.1 представлена html-страница web-приложения, демонстрирующего применение Web Sockets API.



Рис. 7.1. Код html-страницы web-приложения, демонстрирующего работу Web Sockets API

Тело html-страницы (рис. 7.1) включает два элемента <section>. Первый элемент <section> отображает результат проверки работоспособности Web Sockets API в рамках текущего браузера. Второй содержит два элемента <input type=”button”> и один <textarea>. Браузером страница (рис. 7.1) будет отображаться примерно так, как это показано на рис. 7.2.

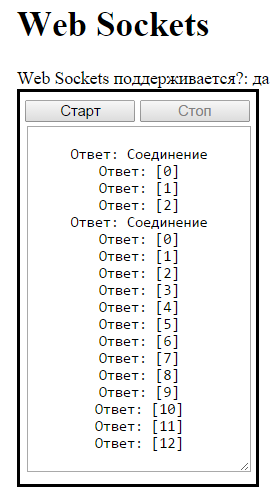


Рис. 7.2 Отображение браузером html-страницы,

представленной на рис. 7.1

Клавиша «Старт» (рис. 7.1, 7.2) предназначена для запуска процесса соединения и обмена информацией с сервером. Клавиша «Стоп» останавливает этот процесс. Результаты соединения и обмена данными отражаются с помощью элемента <textarea>

JavaScript-код приложения находится в файлах modernizr\_ws.js и WebSockets.js, загружаемых с помощью двух элементов <script>, расположенных в начале html-страницы.

Файл modernizr\_ws.js содержит код библиотеки Modernizr (см. 1.8), с помощью которой осуществляется проверка работоспособности Web Sockets API в текущем браузере.

Файл WebSockets.js содержит, клиентский JavaScript-код, обеспечивающий обмен данными с серверной компонентой приложения по протоколу WebSocket.

При загрузке html-страницы выполняется анонимная функция, заданная в качестве обработчика события load встроенного объекта window (рис. 7.3).



Рис. 7.3. Фрагмент файла WebSocket.js: анонимная функция, выполняющаяся при загрузке html-страницы

Функция (рис.7.3) помощью метода websockets объекта Modenizr проверяет работоспособность Web Sockets API в браузере, загружающим эту страницу. После проверки с помощью функции WriteMessage выводится соответствующие сообщение. Далее функция устанавливает значения глобальных переменных и свойства disable, регулирующее доступность клавиш «Старт» и «Стоп».

При нажатие клавиши «Старт» вызывается функция exe\_start (рис. 7.4). Функция проверяет наличие открытого соединения с сервером и, если его нет, то создает его. Обратите внимание: в адресной строке (параметр конструктора WebSocket), указывающей местоположение сервера, используется URL-схема ws.

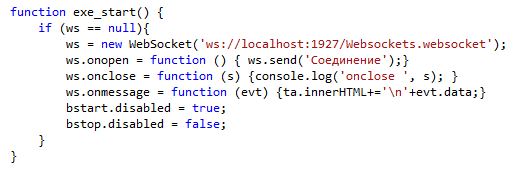


Рис. 7.4. Фрагмент файла WebSocket.js: функция exe\_start

Кроме того, в функции задаются три обработчика событий соединения: onopen (соединение открыто), onclose (соединение закрыто), onmessage (от сервера поступило сообщение) и устанавливаются новые значения свойства disable для клавиш.

Если соединение с сервером успешно (событие onopen) установлено, то серверу с помощью метода send отправляется текстовое сообщение «Соединение». При получении сообщения от сервера (onmessage), текст сообщения отображается на html-странице с помощью элемента <textarea>. При закрытии соединения (onclose) выводится соответствующее сообщение к консоль браузера.

Закрытие соединения осуществляется функцией exe\_stop (рис. 7.5), которая вызывается при нажатии клавиши «Стоп». Собственно закрытие осуществляется методом close, принимающем два параметра: пользовательский код (допускается в пределах от 3000 до 4999) и произвольное текстовое сообщение, характеризующее причину закрытия.

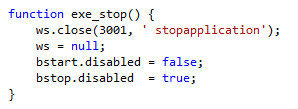


Рис. 7.5. Фрагмент файла WebSocket.js:

функция exe\_stop

Таким образом, после установки соединения с сервером, клиентская часть приложения готова к обработке сообщений от сервера. При этом работа html-страницы не блокируется и пользователь может продолжать работу с отображаемой браузером страницей независимо от установленного соединения. При поступлении сообщения от сервера, вызывается обработчик события onmessage, который может выполнить необходимые действия в рамках клиентской части. Другими словами, с помощью протокола WebSocket появляется возможность обрабатывать серверные события (сообщение извещает клиента о событии) на стороне клиента, не выполняя запросов к серверу, как это бы потребовалось при применении HTTP.

**7.2.2. Сервер протокола WebSocket**

Серверная часть web-приложения, может быть реализована с помощью любой технологии разработки web-приложений, поддерживающей протокол WebSocket.

На рис. 7.6 представлен серверный код web-приложения, реализованный на языке C# в рамках технологии ASP.NET 5. Код некоторых методов свернут: он будет поясняться позже.

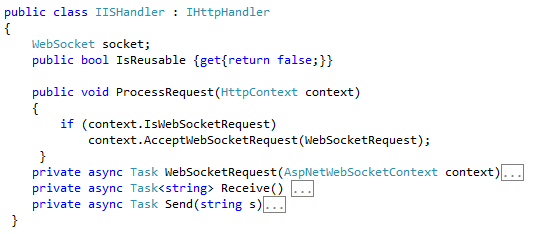


Рис. 7.6. Серверный код web-приложения, взаимодействующего с клиентом по протоколу WebSocket

Программный код, отображенный на рис. 7.6, представляет собой http-обработчик ASP.NET. Способ вызова http-обработчика описывается в конфигурационном файле web-приложения (рис. 7.7) элементом handlers.

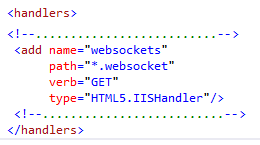


Рис. 7.7. Фрагмент конфигурационного файла

web-приложения

В соответствии с рис. 7.7. обработчик IISHandler (атрибут type) может быть вызван с помощью GET-запроса (атрибут verb) протокола HTTP и при этом, расширение имени вызываемого серверного ресурса должно быть websocket (атрибут path). Сопоставив с рис. 7.4 не сложно заметить, что в адресной строке имя ресурса имеет такое же расширение.

Интерфейс IHttpHandler, который реализует обработчик, предусматривает одно свойство IsReusable и один метод ProcessRequest.

Свойство IsReusable имеет булевый тип, предназначено только для чтения и устанавливается в значение false, если при каждом вызове обработчика следует создавать новый экземпляр обработчика.

При вызове обработчика, осуществляется вызов метода ProcessRequest, которому в качестве параметра передается объект, описывающий контекст текущего http-соединения. С помощью свойства IsWebSocketRequest этого объекта можно определить является ли пришедший запрос, запросом на соединение по протоколу WebSocket. Если это так, устанавливается соединение и управление предается методу WebSocketRequest (рис. 7.8).

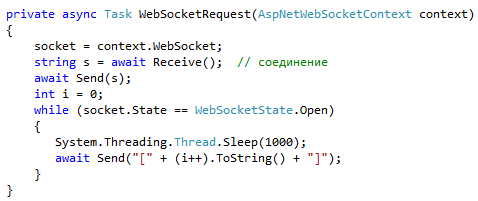


Рис. 7.8. Фрагмент http-обработчика (рис.7.6): метод WebSocketRequest

В качестве параметра метод WebSocketRequest принимает объект, описывающий контекст открытого соединения по протоколу WebSocket. Из свойства WebSocket этого объекта извлекается ссылка на объект операционной системы (называемый сокетом TCP), предназначенный для управления процессом обмена данными по протоколу TCP.

В процессе своей работы метод WebSocketRequest вызывает функции Receive (рис. 7.9) и Send (7.10), предназначенные соответственно для приема и оправки сообщений клиенткой части приложения.

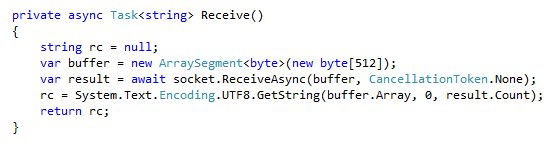
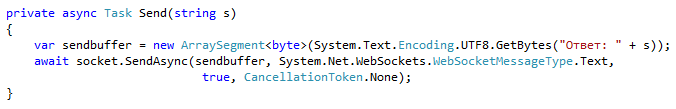


Рис. 7.8. Фрагмент http-обработчика (рис.7.6): метод Receive

Собственно получение данных по протоколу WebSocket в функции Receive (рис. 7.8), осуществляется с помощью метода ReceiveAsync, принимающего два параметра: буфер для получаемых данных и маркер отмены (используется для управления асинхронными операциями). После выполнения ReceiveAsync буфер будет содержать массив байтов в кодировке UTF-8 , который преобразуется к строчному виду и возвращается с помощью оператора return к точке вызова функции.

Рис. 7.10. Фрагмент http-обработчика (рис.7.6): метод Send

Функция Send принимает один параметр: строку, предназначенную для пересылки клиенту. Полученная строка преобразовывается в байтовый массив и с помощью метода SendAsync оправляются клиенту.

Обратим еще раз внимание на рис. 7.8 демонстрирующий основной цикл отправки сервером сообщений клиенту. Отправка сообщений осуществляется до тех пор, пока клиент не разорвет соединение (нажмет клавишу «Стоп»). В теле цикла, вызывается функция Sleep, осуществляющая приостановку потока выполнения программы на одну секунду перед каждой оправкой сообщения. Функция Sleep имитирует нагрузку. В реальном приложении, вместо нее должен быть программный код, доставляющий данные для отправки их клиенту.