УДК 004.5

Беседа Д.Г., Семенистый Н.В. Аноприенко А.Я.
Донецкий национальный технический университет, г. Донецк
кафедра компьютерной инженерии

ИССЛЕДОВАНИЕ ТЕХНОЛОГИИ ПОСТРОЕНИЯ ПРИЛОЖЕНИЙ РЕАЛЬНОГО ВРЕМЕНИ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА WEBSOCKET

Аннотация

Беседа Д.Г., Семенистый Н.В., Аноприенко А.Я. Исследование технологий построения приложений реального времени с использованием протокола websocket*. Выполнен анализ технологии WebSocket и ее внедрение в современные инструменты создания Web-приложений. Описана реализация данной технологии в платформах Microsoft ASP.NET и Node.js.*

Ключевые слова**:** *socket.io, signalR, Websocket, MVC, AJAX, polling, long polling, hub.*

Постановка проблемы: В связи с постоянным ростом количества пользователей сети Интернет и расширением использования таких технологий как AJAX, которые сильно нагружают серверы постоянным потоком своих запросов, значительно возрастает и нагрузка на серверное оборудование. В связи с этим необходимы новые подходы к построению приложений, работающих в режиме реального времени. Одним из таких решений являет внедрение протокола Websocket.

Целью данного доклада является анализ особенностей протокола Websocket, его технических характеристик, преимуществ и недостатков, а также – сравнение реализаций «оберток» для протокола на платформах .NET и Node.js.

Анализ существующих исследований и разработок. Проведен анализ технологии Websocket. Протокол является достаточно новым, но, несмотря на это, получил широкую популярность, включен в спецификацию HTML5.

Постановка задачи. Необходимо рассмотреть транспортные параметры протокола Websocket, структуру его сообщения, возможные проблемы при его использовании. Рассмотрев реализации на разных платформах, необходимо сформировать вывод по эффективности работы с протоколом, используя библиотеки данных платформ.

Анализ протокола Websocket. Являясь синхронным протоколом, **http** предназначен для решения бытовых задач. Время запроса занимает

незначительное время или вовсе не берется в расчет, но с ростом вычислительных мощностей и развитием Интернета нагрузка на сервера увеличивалась, что сделало данный протокол непригодным для решения многих современных задач.

Одним из решений этой проблемы стал протокол **Websocket**. Websocket – это протокол полнодуплексной двунаправленной связи поверх TCP-соединения, предназначенный для обмена данными между браузером и веб-сервером в режиме реального времени. Это стало доступно за счет того, что теперь нет понятия «клиент» и «сервер», а имеются два равноправных участника обмена данными.

Протокол мог бы стать успешным решением в проекте «Миллион пикселей для библиотеки».[1][2] При выбранной области она станет доступной для отображения только после перезагрузки страницы, а использование протокола Websocket позволило бы видеть изменения в структуре сетки изображений в момент их добавления. Структурная схема работы протокола WebSocket изображена на рисунке 1.

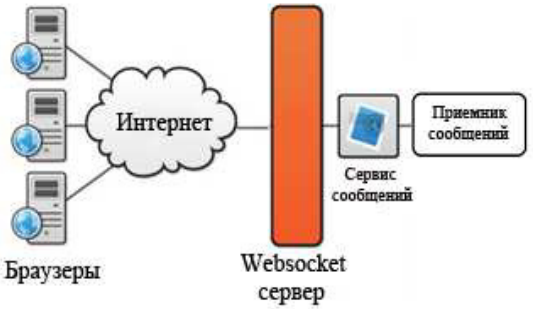

Рис. 1 – Структурная схема работы WebSocket.

На клиентской стороне работа с протоколом выполняется при помощи объекта Websocket.[3] Структура сообщений данного объекта отображена на рисунке 2.

На рисунке 3 приведен формат dataframe протокола Websocket.

Данные минимально обозначены двумя байтами: 0x00 вначале и 0xFF в конце. Это позволяет быстрее обмениваться данными участникам. Соединение достаточно установить один раз и, следовательно, экономить время и трафик на его установление при необходимости. Таким образом, отсутствует ограничение на время жизни в неактивном состоянии. Соединение может оставаться без траты ресурсов. Единственным недостатком в данной ситуации является то, что на сервере могут забиваться TCP-сокеты.

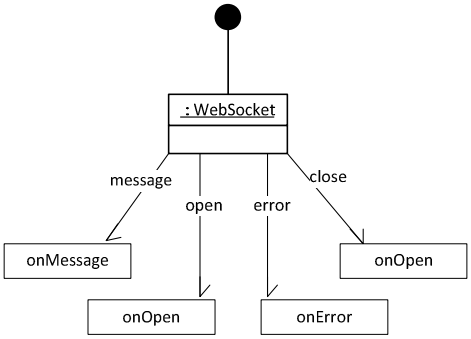

Рис. 2 – Асинхронные сообщения объекта Websocket.

Рис. 3 – Формат dataframe протокола Websocket.

- Маска предназначена для защиты от так называемых атак «отравленного кэша». Маска представляет собой случайное 32-битное значение, которое варьируется от пакета к пакету.
- Опкод – задает интерпретацию данных фрейма.
- Длина тела сообщения. (7/7+16/7+64 бит)
- Тело фрейма.

Еще одним преимуществом Websocket является отсутствие ограничения на количество как активных, так и пассивных соединений (кроме памяти самих серверов). Протокол зарекомендовал себя с хорошей позиции по множеству параметров, таких как безопасность, универсальность, скорость, кросс-доменность. Он был реализован во всех современных браузерах, что делает возможным его использование в проектах, основой которых есть интерактивное взаимодействие с пользователем, потому что протокол в некотором смысле реализует паттерн publish/subscribe, то есть, будучи подписанным на обновления какого-либо публикатора, вам больше не нужно отсылать запросы о получении информации, а ее распространением будет заниматься только сторона, обладающая этой информацией.

Протокол Websocket позволяет экономить существенное количество передаваемой информации за счет того, что он имеет крайне малые заголовки в сравнении с обычными HTTP-запросами. Это продемонстрировано на рисунке 4. Очевидно, что технология Polling, которая базируется на асинхронных запросах, посылает на порядок больше сообщений в канал, чем идет по протоколу Websocket.

Обертки более высокого уровня были реализованы во множестве языков и на множестве платформ. В качестве примера рассмотрим библиотеки SignalR платформы .Net и Socket.io Node.js.

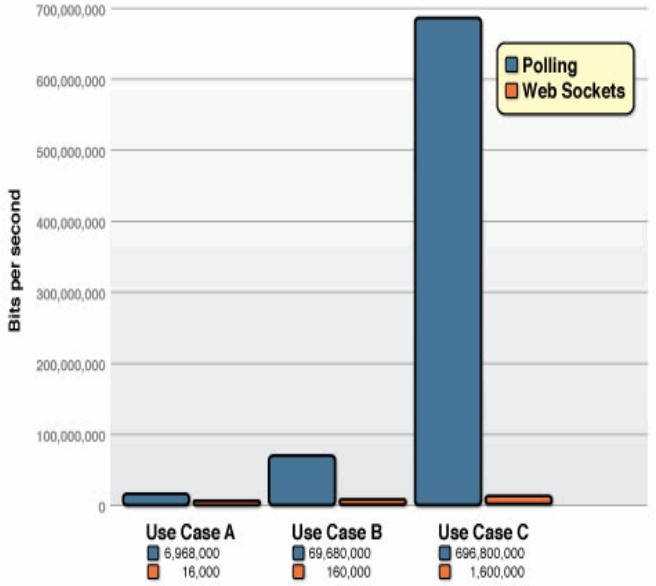

Рисунок 4. Сравнительная характеристика количества использованных битов для передачи одинаковой информации технологий Polling и Websocket

Socket.io. Реализации данной библиотеки выполнена на платформе Node.js, которая набирает все больше и больше популярности в обществе веб-разработчиков. Среди числа ее преимуществ:

- написание серверного кода на языке веб-разработки JavaScript, то есть возможность писать серверный и клиентский код на одном языке;

- простота API. Node.js вернул в некоторой мере процедурный взгляд на программирование веб-сервера после активного перехода большинства веб-программистов на программирование в парадигме Model View Controller, в то же время для того, чтобы запустить веб-сервер необходимо минимальное количество инструкций, конфигурирующих его;

- асинхронность, наиболее существенный агрумент. Движок Node.js представляет собой event loop. Таким образом, он реализует паттерн Event Driven Design (EDD). Большинство серверных операций многих веб-проектов представляют собой операции ввода-вывода. Node.js не создает отдельный поток на каждый запрос, а ставит их в очередь, но обрабатывает после того, как они были приняты асинхронно, не ожидая окончания обработки предыдущего. Это является недостатком при условии, что на сервере будут

выполняться затратные вычисления, но в то же время дает огромное преимущество, если таковых не будет, что и встречается в большинстве случаев.

Именно асинхронность является той особенностью, которая заставляет использовать эту платформу на проектах, которые подразумевают обильный обмен данными между сервером и клиентом. Проекты, использующие протокол веб-сокетов, преимущственно создаются на платформе Node.js с использованием библиотеки Socket.io.

Socket.io предоставляет много преимуществ перед стандартной реализацией протокола:

- Несколько сокетов внутри одного соединения (благодаря пространству имен);
- Определение потери связи благодаря периодическим сообщениям о состоянии соединения (heartbeats);
- Поддержка восстановления соединения с буферизацией (особенно ценно для плохих сетей).

Рассмотрим конфигурацию объекта socket. Один из главных настраиваемых атрибутов – возможные транспорты. В библиотеке реализована поддержка следующих протоколов:

- WebSocket;
- Adobe® Flash® Socket;
- AJAX long polling;
- AJAX multipart streaming;
- Forever Iframe;
- JSONP Polling.

Можно указывать массив поддерживаемых транспортов. При этом flash socket не будет активирован в браузерах, которые полностью поддерживают технологию Websocket.

Конфигурируется целый ряд интервалов и таймаутов, как время закрытия соединения, таймаут и интервал heartbeat, устанавливающие время сообщений о состоянии клиентов, длительность процесса polling.

Следующим важным аспектом являются клиентские настройки. Среди них: поле, свидетельствующее о том, является ли клиент браузером, для определения необходимости хостинга статических ресурсов, включен ли клиентский кэш, время актуальности ресурсов на клиенте, минификация файла socket.io.js, gzip архивирование скрипта.

В настройках также можно установить уровень логгирования:

- 0 – ошибка;
- 1 – предупреждение;
- 2 – информирование;
- 3 – дебаг.

Среди важнейших конфигурируемых настроек авторизация, которая по умолчанию отключена.

Socket.IO имеет поддержку 2 типов авторизации: применяемая глобально, которая активируется при инициализации /рукопожатии или авторизация для пространства имен.

Когда клиент хочет установить постоянное соединение с сервером Socket.IO, начинается процесс рукопожатия. Рукопожатие инициируется XHR запросом или JSONP (для кросс-доменных запросов).

В момент соединения с сервером начинается сбор данных из запроса, которые могут понадобиться позже. Это делается по двум причинам:

1. Пользователи могут хотеть авторизовать клиентов на основе информации из заголовков или по IP-адрес.
2. Не все транспорты отправляют заголовки, когда пытаются установить соединение с сервером, поэтому рукопожатие хранится внутренне, так что пользователи могут повторно использовать эти данные, когда клиент подключен. Например, можно считывать идентификатор сессии из заголовков cookie.

Объект handshakeData содержит следующую информацию:

```
{
  headers: req.headers       // <Object> заголовки запроса
 , time: (new Date) +"       // <String> дата соединения
 , address: socket.address()  // <Object> сокет удаленного объекта
 , xdomain: !!headers.origin  // <Boolean> кросс-домен?
 , secure: socket.secure      // <Boolean> https соединение
 , issued: +date              // <Number> время, когда произошло рукопожатие
 , url: request.url           // <String> входной путь запроса
 , query: data.query          // <Object> результат обработки GET параметров
}
```

Итак, глобальная авторизация включается установкой конфигурационного метода configure. На клиентской части авторизация обрабатывается в двух callback-методах, connect и error, обрабатывающие случаи успеха и ошибки. Авторизация пространства имен дает гибкость для авторизации, что позволяет давать различные права пользователям.

Важной частью данного паттерна публикации и подписки является реализация возможности публиковать сообщения, предназначенные для разных групп пользователей. Данные группы называются комнатами. Зайти и выйти из комнаты можно с помощью методов join и leave.

SignalR. SignalR – библиотека плафторемы ASP.NET для реализации режима real-time веб-приложений. Данная технология является абстракцией транспортных каналов. При ее использовании в первую очередь выбирается тип наиболее оптимального соединения:

- WebSockets;
- Server-sent Event – стандарт HTML5, имеющий более широкое распространение чем сокеты, однако здесь инициатором событий является сервер

- Long Polling – последовательность AJAX-запросов, которые являются активными в течении нескольких минут. Имеет широкую поддержку, однако нагружает сервер при большом количестве соединений.
- Polling – последовательность коротких AJAX-запросов. Поддерживается во всех браузерах, но создает большую нагрузку на сервер.

В данной библиотеке есть 2 основных класса: PersistantConnection и Hub.

PersistantConnection является простым соединением для посылки одиночных, групповых и широковещательных сообщений. Этот класс дает разработчику доступ к низкому уровню канала сети.

Hub – это надстройка над PersistantConnection. Он позволяет напрямую работать клиенту и серверу друг с другом. Таким образом клиент может вызывать методы сервера так же просто, как он вызывает свои локальные. Также этот класс упрощает работу, если приложение должно посылать сообщения разных типов.

В большинстве случаев используется Hub. Для работы с ним необходимо создать класс, который наследует свойства базового класса после чего можно будет посылать сообщения.

Выводы. На сегодняшний день лучшим решением для написания приложений в режиме реального времени является технология Websocket. Она позволяет создавать устойчивое соединение между клиентом и сервером, уменьшая нагрузку на сервер по сравнению с остальными методами. Важность этой технологии подчеркивается также ее внедрением в наиболее популярные инструменты программирования как ASP.NET и JavaScript.

Перспективы дальнейших исследований. Websocket является технологией нового поколения, является одним из элементов реформации Web в составе HTML5. Вполне вероятно, что в ближайшее время появятся другие технологии, которые будут справляться с задачей обеспечения real-time соединения между компьютерами еще лучше.

Список литературы

1. Беседа Д.Г., Семенистый Н.В., Аноприенко А.Я. Миллион пикселей для библиотеки // Материалы III всеукраинской научно-технической конференции «Информационные управляющие системы и компьютерный мониторинг (ИУС и КМ 2012)» – 17-18 апреля 2012 г., Донецк, ДонНТУ, 2012. С. 358-362.

2. Беседа Д.Г., Семенистый Н.В., Аноприенко А.Я. Миллион пикселей для библиотеки // Материалы VIII международной научно-технической конференции «Информатика и компьютерные технологии – 2012» – 18-19 сентября 2012 г., Донецк, ДонНТУ, 2012.

3. Wang V., Salim F., Moskovits P. The Definitive Guide to HTML5 WebSocket. New York, Apress, 2013. С. 17