электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Интерактивный веб-сервис WebSocket

 # 01, январь 2015

Федоренков Р. В.[1,*], Ничушкина Т. Н.[1]

УДК: 004.4

[1]Россия, МГТУ им. Н.Э. Баумана
[*]suffering123494@yandex.ru

Введение

В настоящее время особую популярность получили различные веб-сервисы. Трудно представить нашу жизнь без онлайн бронирования авиа или ж/д билетов, гостиниц в других городах или даже странах, заказа такси, покупки различных вещей через интернет магазины и многого другого. Основным критерием при построении таких веб-сервисов, например, интернет магазина, является актуальность информации. В таких сервисах необходим очень частый обмен информацией между клиентом, например сидящим перед своим монитором и выбирающем какую-либо продукцию и сервером, который хранит информацию обо всей продукции, представленной на сайте. Сервер должен вовремя обновлять информацию у клиента, чтобы не получилась так, что один и тот же продукт заказали 2 разных человека.

1. Способы построения интерактивного веб-сервиса.

Любой интерактивный веб-сервис образно можно представить следующим образом. Имеется сервер, клиент, и некая среда передачи — транспорт. Схематически это будет выглядеть как показано на рисунке 1.

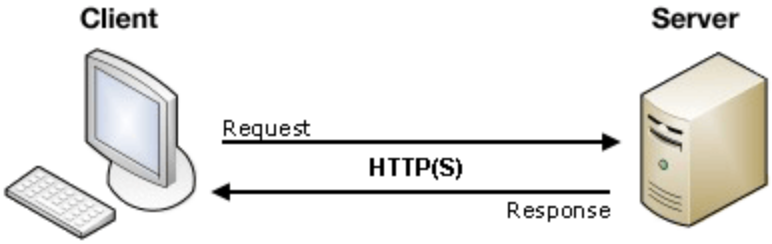

Рис. 1. Схематическое представление веб-сервиса

Серверная часть может быть реализована на многих языках программирования, например, таких как:

- Javascript(NodeJS)
- Python(Twisted, Tornado)
- Ruby(EventMachine)

Наиболее популярно в последнее время стало применение NodeJS, так как это средство позволяет использовать общую кодовую базу(Javascript) как на клиентской, так и на серверной части.

Теперь, чтобы веб-приложение работало так же быстро, как, например, десктопное приложение, необходимо устроить обмен между сервером и клиентом таким образом, чтобы он выполнялся посредством маленьких, максимально частых сообщений. В таком случае и минимизируется латентность, и можно обеспечить асинхронность.

Самый простой способ общения сервера с клиентом можно представить следующим образом (см. рисунок 2).

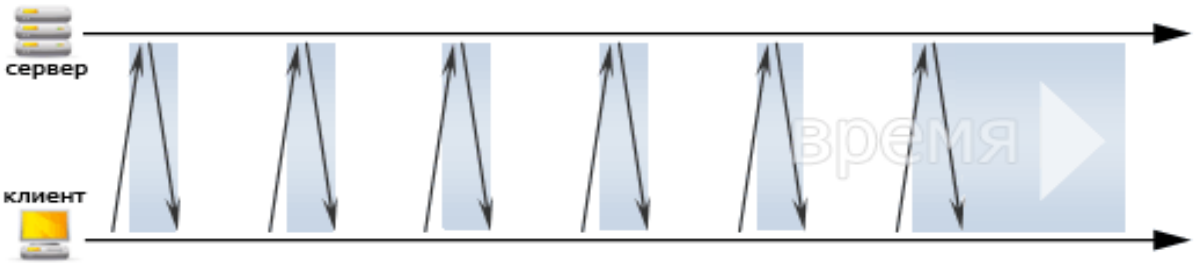

Рис. 2. Классический опрос

Такой метод связи называется классическим опросом (англ. *polling*). Клиент запросил — сервер ответил. Например: клиент запрашивает: «Дай мне последние новости за 5 минут». Сервер ему присылает список новостей либо говорит: «Извини, новостей нет, приходи потом».

Следующий способ общения называется длинный опрос (англ. *long polling request*) (см. рисунок 3).

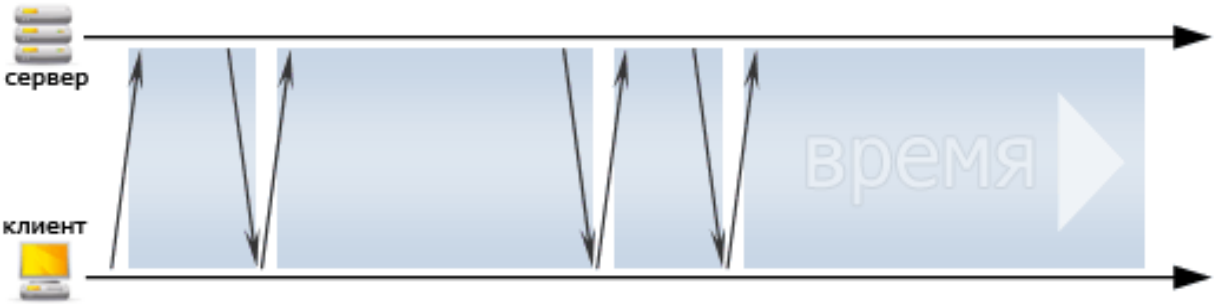

Рис. 3. Длинный опрос

В этом случае, сервер, вместо того, чтобы отвечать сразу: «Новостей нет, приходи потом», задерживается с ответом до наступления какого-либо события. На практике, соединение обычно переустанавливается раз в 20-30 секунд, чтобы избежать возможных проблем, например с HTTP-прокси, которые не любят висящие неактивные соединения. Если за время, отведенное на задержку ответа, приходят новые новости, то они отправляются клиенту. Иначе сервер закрывает соединение и клиент тут же отправляет новый ожидающий запрос.

Третий метод обмена информации между сервером и клиентом это потоковая передача(англ. *HTTP Live Streaming*). Сервер, вместо того чтобы дожидаться полной загрузки запрашиваемых данных, начинает отвечать на запрос частями, то есть, отправляет новые события порционно (см. рисунок 4). Такой тип передачи используется в потоковой мультимедии, когда можно начать воспроизведение видео не дожидаясь конца передачи.

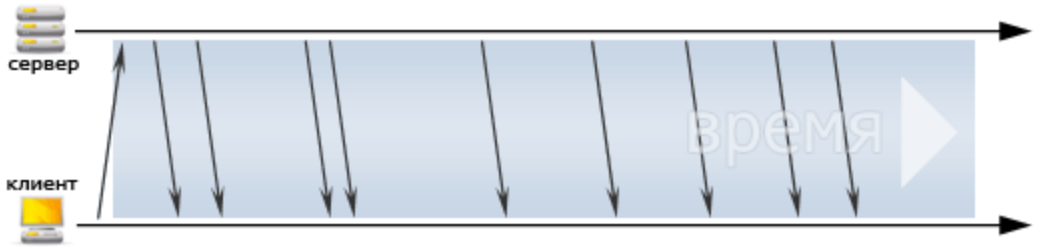

Рис. 4. HTTP Live Streaming

Во всех рассматриваемых методах общения клиента с сервером есть проблема в распределении ролей в протоколе: клиент - ведущий, а сервер всегда ведомый. Клиент спрашивает — сервер отвечает. Клиент может прислать на сервер какую-то информацию, но сервер по своей инициативе не присылает клиенту ничего. Он отправляет данные только как ответ на действие клиента.

Следующая проблема с которой мы сталкиваемся при построении интерактивных веб-приложений - необходимость при пересылки каких-либо данных пересылать вместе с полезной нагрузкой еще и заголовки. Размер заголовков варьируется от нескольких сотен байт до нескольких килобайт. Это не является проблемой, когда речь идет о передаче странички размеров 100 килобайт. Но когда нам нужно передать клиенту около 20 килобайт информации (например небольшое текстовое сообщение в чате), то заголовки оказывают существенное влияние.

2. Протокол WebSocket

И тут нам на помощь приходит протокол **WebSocket** (стандарт RFC 6455). **WebSocket** — протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

- Реализующее *дуплексный* способ связи устройство может в любой момент времени и передавать, и принимать информацию. Передача и прием ведутся устройством *одновременно* по двум физически разделённым каналам связи (по отдельным проводникам, на двух различных частотах и др. за исключением разделения во времени — поочередной передачи). Пример дуплексной связи — разговор двух человек (корреспондентов) по городскому телефону: каждый из говорящих в один момент времени может и говорить, и слушать своего корреспондента. Дуплексный способ связи иногда называют *полнодуплексным* (от англ. *full-duplex*); это синонимы.

Помимо дуплексной, выделяют также *полудуплексную* и *симплексную* связь.

- Реализующее *полудуплексный* (англ. *half-duplex*) способ связи устройство в один момент времени может либо передавать, либо принимать информацию. Пример полудуплексной связи — разговор по рации: каждый из корреспондентов в один момент времени либо говорит, либо слушает. Для обозначения конца передачи и перехода в режим приема корреспондент произносит слово «прием» (англ. «*over*»).
- Реализующее симплексный способ связи устройство в один момент времени информация передается только в одном направлении. Такая связь используется в радиовещании, поскольку нет необходимости передавать какие-либо данные обратно на передающую станцию.

Установка соединения

В процессе установления соединения клиент передает веб-серверу обычный HTTP-запрос, на который сервер отвечает либо положительно, что означает, что он поддерживает websocket, либо отрицательно - что свидетельствует о невозможности перейти на websocket. Этот запрос является командой HTTP GET, сконфигурированной как запрос переключения (upgrade request):

Пример *HTTP request header*:

GET /chat HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: example.com
Origin: http://example.com

Если сервер понимает Websocket Protocol, то он отвечает следующим образом:

HTTP/1.1 101 Web Socket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

WebSocket-Origin: http://example.com

WebSocket-Location: ws://example.com/chat

Код состояния 101 в заголовке свидетельствует об успехе. Теперь клиент и сервер могут обмениваться сообщениями по протоколу WebSocket.

Как только одна сторона хочет передать другой какую-то информацию, она отправляет дата-фрейм следующего вида:

0x00, <строка в кодировке UTF-8>, 0xFF

То есть, это просто строка текста, в начало которой помещен нулевой байт 0x00, а в конец — 0xFF. И больше никаких заголовков и метаданных!

Кроме того, с помощью WebSockets так же можно передавать и бинарные данные. Для них используется другой дата-фрейм следующего вида:

0x80, <длина - один или несколько байт>, <тело сообщения>

Сравнительный анализ с ранее используемыми методами

При использовании протокола Websocket в отличие от классического AJAX-запроса, где для передачи каждого сообщения необходимо было передавать несколько килобайт заголовков, служебная информация занимает всего 2 дополнительных байта(один в начале сообщения и один в конце). Разница будет особенно заметна, если выполнять частый обмен небольшими блоками данных.

Рассмотри пример. Предположим, что мы хотим отправить, например, пустое сообщение своему другу. Рассмотрим 3 случая: 1000 запросов в секунду, 10000 запросов в секунду и 100000 запросов в секунду.

При использовании обычного запроса: для каждого сообщения нам необходимо еще передавать около 871 байта заголовков.

- Случай A: 1,000 запросов в секунду: Пропускная способность сети: (871 x 1000) = 871000 байт = 6968000 бит в секунду (6.6 Mbps)
- Случай B: 10,000 запросов в секунду:Пропускная способность сети: (871 x 10000) = 8710000 байт = 69680000 бит в секунду (66 Mbps)
- Случай C: 100,000 запросов в секунду: Пропускная способность сети: (871 x 100000) = 87100000 байт = 696800000 бит в секунду (665 Mbps)

При использование протокола websocket: нам также необходимо передавать заголовки, но их размер равен 2 байтам(один байт вначале и один байт в конце сообщения), вместо 871!

- Случай A: 1000 запросов в секунду: Пропускная способность сети: (2 x 1000) = 2000 байт = 16000 бит в секунду (6.6 Mbps)
- Случай B: 10000 запросов в секунду:Пропускная способность сети: (2 x 10000) = 20000 байт = 160000 бит в секунду (66 Mbps)
- Случай C: 100000 запросов в секунду: Пропускная способность сети: (2 x 100000) = 87,100,000 байт = 696,800,000 бит в секунду (665 Mbps)

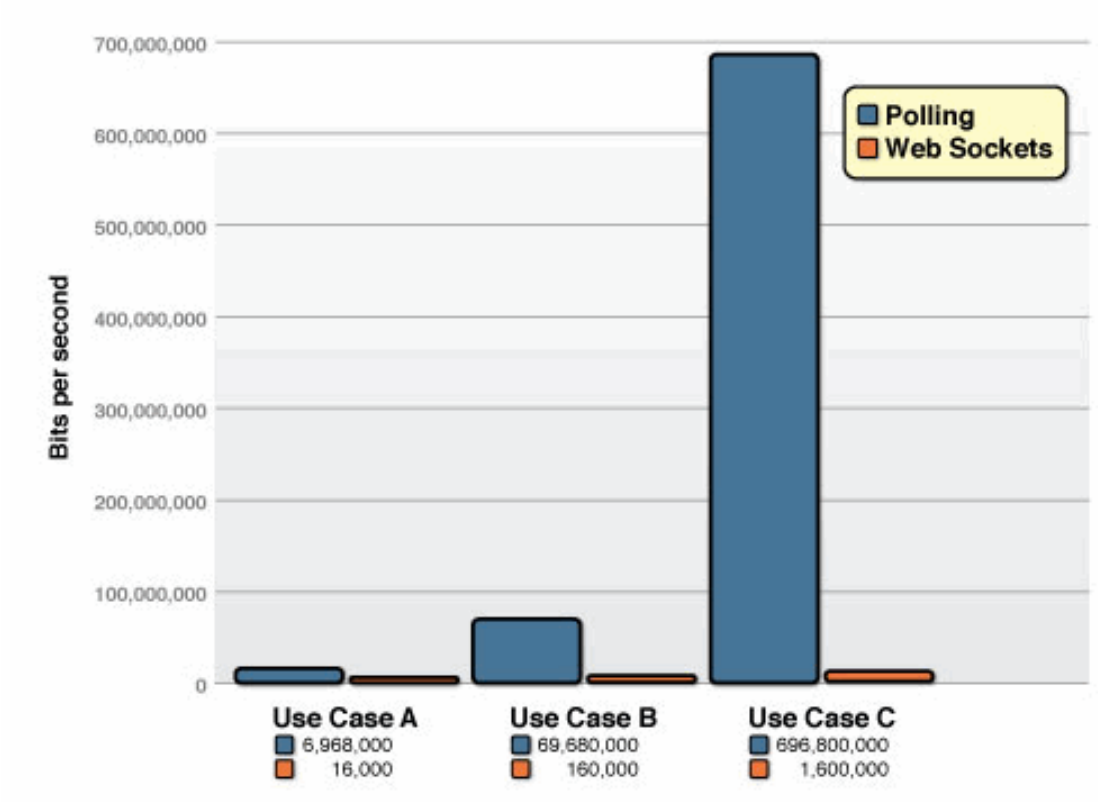

Рис. 6. Сравнение необходимой пропускной способности сети при передачи пустого сообщения при помощи обычного http соединения и при помощи вебсокетов.

Как видно из диаграммы (см. рисунок 6), веб-сокеты почти в 1000 раз меньше нагружают сеть, чем классическое соединение.

Заключение

Можно сделать следующие выводы: по сравнению с ранее используемыми решениями WebSocket имеет ряд преимуществ:

- Websocket позволяют реализвовать полнодуплексную модель связи для веб-приложений. Таким образом, теперь инициатором может выступать не только клиент, но и сервер.
- Websocket имеют низкие требования к сетевым ресурсам и максимальный КПД передачи данных. Служебная информация (англ. overhead) у WebSocket добавляет ровно 2 байта на 1 сообщение. Один байт в начале, второй байт в конце. Никаких лишних данных. В стриминге заголовка нет. Казалось бы проблема больше не должна нам мешать. Но когда встречается кэширующий прокси-сервер или, например, некий антивирус, который любит собрать в себя как можно больше данных и только потом передать клиенту, то мы столкнемся со сложностями. Проблема застревания решается при помощи технологии двухкилобайтного вантуза, когда пакет данных увеличивается каким-либо образом до двух килобайт. Как правило, для

большинства серверов этого хватает, чтобы эффективно пробить засор и доставить данные клиенту. Но, если мы начнем перекидываться по сети такими двухкилобайтными «кирпичами», то об эффективности лучше не упоминать.

- Время жизни канала. Websocket в отличие от HTTP не имеют ограничений на время жизни в неактивном состоянии. Это значит, что его не вправе закрывать всякие прокси(англ. Proxy) и не надо больше периодически переустанавливать соединение. Значит, соединение может висеть в неактивном состоянии и не требовать ресурсов.
- Использование протокола Websocket позволяет построить комплексные веб-приложения. В HTTP предусмотрено ограничение на число одновременно открытых соединений к одному серверу. Поэтому существовала необходимость реализовывать не только серверный, но и клиентский мультиплексор, если на странице располагалось много асинхронных блоков. Это ограничение не распространяется на веб-сокеты. Можно открывать столько соединений, сколько Вам необходимо. И уже исходя из нагрузки на сервер и клиент, а также удобства разработки решать сколько использовать — одно соединение на все блоки или же наоборот — на каждый блок свое.
- И последнее: WebSocket предлагает очень простое и красивое API.

Список литературы

1. Кантор И. Современный учебник JavaScript. Ч.3.-Разное, Разд.-AJAX, Гл.- WebSocket // Javascript.ru: электронный ресурс, 2012. Режим доступа: http://learn.javascript.ru/websockets (Дата обращения: 10.01.2015)
2. Эспозито Дино. Исследуем потенциал WebSockets. // MSDN Magazine: электронный журнал, 2012. № 5. Режим доступа: https://msdn.microsoft.com/ru-ru/magazine/hh975342.aspx (Дата обращения: 10.01.2015)
3. Филиппов М. В., Белоус В. В., Бургина А. М. Метод автоматического обнаружения ошибок верстки веб-страниц. // Инженерный вестник: электронный научно-технический журнал, 2014. №5. Режим доступа: http://engbul.bmstu.ru/doc/711132.html (Дата обращения: 10.01.2015)
4. *Lubbers P., Greco F.* HTML5 Web Sockets: A Quantum Leap in Scalability for the Web. / *Peter Lubbers, Frank Greco. //* WebSocket.org, 2011. Режим доступа: http://www.websocket.org/quantum.html (Дата обращения: 10.01.2015)
5. Fette I., Melnikov A. The WebSocket Protocol. // The Internet Engineering Task Force (IETF), December 2011. P. 70. Режим доступа: https://tools.ietf.org/html/rfc6455 (Дата обращения: 15.01.2015)
6. WebSockets – полноценный асинхронный веб. // Habrahabr, Декабрь 2009. Режим доступа: http://habrahabr.ru/post/79038/ (Дата обращения: 10.01.2015)