УДК 004.021

Е.В. Бычкова, А.О. Голубенко
Донецкий национальный технический университет, г. Донецк
кафедра программного обеспечения интеллектуальных систем

РЕАЛИЗАЦИЯ СИСТЕМЫ МГНОВЕННОГО ОБМЕНА СООБЩЕНИЯМИ В ВЕБ-ПРИЛОЖЕНИЯХ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА WEBSOCKET

Аннотация

Бычкова Е.В., Голубенко А.О. Реализация системы мгновенного обмена сообщениями в веб-приложениях с использованием протокола WebSocket. *Проанализированы методы построения веб-приложений реального времени, необходимые для организации систем мгновенного обмена сообщениями в веб-приложениях. Приведен пример реализации системы мгновенного обмена сообщениями с использованием* Node.js *и библиотеки Socket.io.*

Ключевые слова: *WebSocket, XMPP, Node.js, Socket.io, Redis, publish–subscribe, long-pooling, Comet, Javascript, PHP.*

Постановка проблемы. В настоящее время существует огромное множество веб-приложений. Для привлечения наибольшего количества пользователей разработчикам приходится искать новые конкурентные преимущества. Отправка уведомлений пользователю и фоновое обновление контента в реальном времени, мгновенный обмен сообщениями между пользователями – все это делается для удобства использования приложения и избавляет пользователя от лишних действий. Поэтому проблема разработки веб-приложений реального времени на сегодняшний день весьма актуальна.

Анализ литературы. Проведен анализ методов построения веб-приложений реального времени, необходимых для организации системы мгновенного обмена сообщениями между пользователями сайта. В статье описывается метод организации мгновенного обмена сообщениями при помощи платформы Node.js и библиотеки Socket.io.

Цель статьи – проанализировать возможность использования протокола WebSocket для организации системы мгновенного обмена сообщениями в веб-приложениях.

Для реализации мгновенного обмена сообщениями между пользователями сайта необходимо использовать сервер, способный удерживать постоянное или, когда это не представляется возможным, long-pooling соединение с большим количеством пользователей. Например, для этого можно использовать хорошо себя зарекомендовавшие XMPP-сервера,

comet-сервера, такие как Dklab Realplexor, или сервера, работающие с использованием протокола WebSocket.

Использование протокола XMPP хорошо подходит в том случае, когда разработчика устраивает стандартный набор функциональности XMPP-сервера и набор дополнительных расширений XEP (XMPP Extension Protocols). В случае если необходимо создавать довольно сложную систему мгновенного обмена сообщениями с функционалом, выходящим за рамки существующих XEP, возникает необходимость писать собственные расширения. Для этого приходится разбираться во внутренней структуре работы сервера, и на это может уйти много времени. Также недостатком протокола XMPP является избыточность передаваемой информации – около 70% межсерверного трафика составляют сообщения о присутствии, из которых 60% - лишние [1]. Еще одним большим недостатком XMPP-серверов является сложность интеграции с веб-приложениями. Они никак не взаимодействуют с серверами, на которых работают сайты (например, Apache или nginx). Поэтому для синхронизации xmpp-сессий и сессий веб-приложения приходится использовать дополнительные расширения и механизмы. Например, если необходимо после загрузки страницы сайта подключить пользователя к XMPP-серверу, можно использовать XMPP Prebind [2].

Comet-сервера тоже достаточно часто используются для реализации мгновенного обмена сообщениями, однако эту технологию можно назвать устаревшей. На ее место пришел WebSocket, который имеет много преимуществ по сравнению с Comet. Он обеспечивает ассинхронный обмен сообщениями между браузером и веб-сервером в режиме реального времени при помощи полнодуплексной связи поверх TCP-соединения. Клиенты, поддерживающие WebSocket, обычно работают быстрее и создают меньше запросов и, следовательно, потребляют меньше трафика [3]. Для реализации мгновенного обмена сообщениями через websocket отлично подходит платформа Node.js и библиотека Socket.io.

Node.js является серверной технологией, которая основана на разработанном компанией Google JavaScript-движке V8. Это прекрасно масштабируемая система, поддерживающая не программные потоки или отдельные процессы, а асинхронный ввод-вывод, управляемый событиями. Она идеально подходит для веб-приложений, которые не выполняют сложных вычислений, но к которым происходят частые обращения [4].

Socket.io - это JavaScript-библиотека, предназначенная для реализации обмена данными между клиентом и сервером в режиме реального времени. Данная библиотека состоит из двух частей: серверной, работающей на Node.js, и клиентской, работающей в браузере. Оба компонента имеют похожее API. Socket.io может использовать разные транспортные механизмы: WebSocket, Adobe Flash Socket, AJAX long polling, AJAX multipart streaming, Forever Iframe, JSONP Polling [5]. Одним из главных преимуществ данной библиотеки является то, что разработчику не нужно задумываться над тем, какой из

транспортных механизмов необходимо использовать. Например, технологию WebSocket не поддерживают старые браузеры, которыми до сих пор пользуется большое количество людей. Библиотека сама определит, поддерживается ли данная технология браузером, и при необходимости переключится на ту, которая имеет меньший приоритет. При желании разработчик может сам устанавливать приоритет использования транспортных механизмов.

Реализовать передачу сообщений с использованием Socket.io можно двумя способами. Если при передаче сообщения на стороне сервера не нужно производить действия, потребляющие большое количество ресурсов, например, необходимо просто принять сообщение, сохранить в базу и отправить адресату, то можно ограничиться только использованием Node.js.

Схема работы системы мгновенного обмена сообщениями в данном случае будет следующей.

1. При запуске нового сервера Node.js он занимает определенный порт и подключает Socket.io, который включается в режим ожидания новых соединений.

2. Клиент при загрузке страницы приложения подключается к Socket.io, создает слушателя события приема сообщения с сервера (назовем такое событие «new message») и ждет срабатывания этого события. В тот же момент при подключении клиента, Socket.io создает событие, например «message from client», и ожидает отправки сообщений от клиента.

3. При отправке пользователем сообщения, в котором указывается событие «message from client», данное событие срабатывает в Socket.io, библиотека производит сохранение сообщения в базу и отправляет сообщение адресату с указанием события «new message».

Общая схема работы приложения в данном случае представлена на рис. 1.

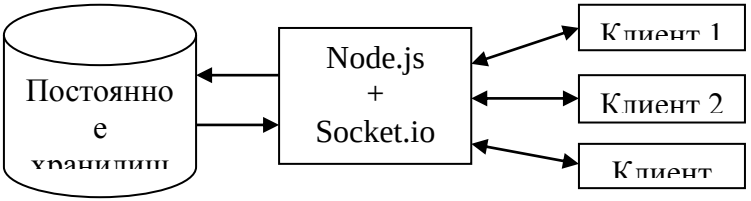

Рисунок 1 – Схема работы системы мгновенного обмена сообщениями с использованием Node.js и Socket.io для приема и отправки сообщений

В случае, когда при передаче сообщений на стороне сервера должны выполняться операции, требующие большого количества ресурсов, например, в теле сообщения могут быть переданы изображения или другие файлы, которые необходимо обработать и сохранить на сервере, лучше разделить отправку и прием новых сообщений. Прием сообщений от клиента и их обработку лучше поручить основному серверу, а передачу сообщений клиенту

оставить Socket.io. В данном случае возникает необходимость организации связи в реальном времени между основным сервером и Node.js. Для этого можно использовать утилиту Pub/Sub сетевого журналируемого хранилища данных Redis. В данном случае, если необходимо, основной сервер и Node.js можно разместить на отдельных физических серверах. Схема работы системы мгновенного обмена сообщениями в данном случае будет следующей.

1. При запуске нового сервера Node.js занимает определенный порт и подключает Socket.io. С использованием библиотеки «redis» Node.js подписывается на заранее известный канал Redis Pub/sub в ожидании сообщения от основного сервера.

2. Клиент, как и в предыдущем случае, при загрузке страницы приложения подключается к Socket.io, создает слушателя события приема сообщения с сервера («new message») и ждет срабатывания этого события.

3. При отправке на основной сервер нового сообщения выполняются операции, требующие большого количества ресурсов, и через Pub/sub сервер отправляет сообщения Node.js .

4. Node.js получает сообщение и с помощью библиотеки Socket.io отправляет его тому клиенту, которому оно адресовано.

Общая схема работы приложения представлена на рис. 2.

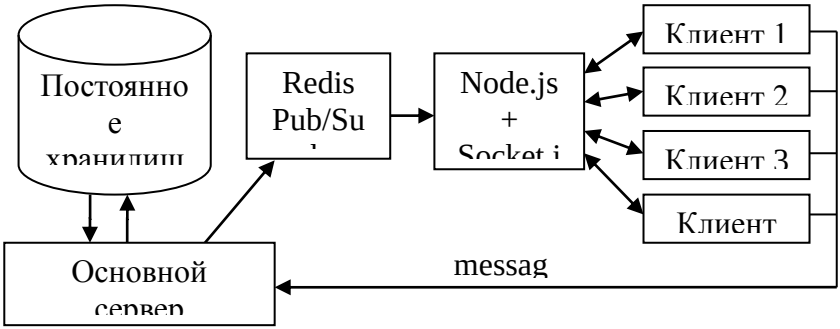

Рисунок 2 – Схема работы системы мгновенного обмена сообщениями с использованием Node.js и Socket.io только для отправки сообщений

В большинстве PHP-приложений авторизация происходит через сессии. Если необходимо реализовать возможность обмена сообщениями только между авторизованными пользователями, то при подключении клиента к Socket.io в событии «authorization» необходимо определить, авторизован пользователь в системе, или нет. Для этого необходимо, чтобы Node.js получил доступ к сессии пользователя. Лучше всего создать единое хранилище данных сессий для php и Node.js . Желательно, чтобы скорость доступа к данным в хранилище была достаточно высока, поэтому для этих целей идеально подойдет упомянутый ранее Redis. Для настройки работы с сессиями в PHP нужно использовать метод session_set_save_handler.

Если основное приложение построено полностью на AJAX (Full-AJAX приложение), может возникнуть проблема с разрывом соединения между клиентом и Node.js в случае, если сессия в основном приложении была завершена. То есть клиент продолжает использовать сайт в качестве неавторизованного пользователя, но при этом имеет соединение с Node.js как авторизованный пользователь. Эту проблему можно решить с помощью Redis Pub/Sub. Если основное приложение построено на с использованием шаблона MVC, можно перед каждым вызовом действия контроллера проверять, авторизован пользователь или нет и в случае, если сессия была завершена, публиковать в канал Pub/Sub, к которому подписан Node.js сообщение о том, что соединение с данным пользователем необходимо разорвать. То же самое можно сделать при чтении информации о сессии, если работа с сессиями была настроена с помощью функции session_set_save_handler.

Выводы. Проанализированы методы построения веб-приложений реального времени, необходимые для организации системы мгновенного обмена сообщениями. Одним из наилучших методов организации веб-приложений реального времени является протокол WebSocket. Недостатком использования Websocket является отсутствие поддержки данного протокола старыми браузерами, которые до сих пор применяет большое количество пользователей. От данного недостатка можно избавиться, используя для поддержки старых браузеров более старые транспортные механизмы, например Adobe Flash Socket или AJAX long polling. В статье предложены два метода построения системы мгновенного обмена сообщениями с использованием платформы Node.js и библиотеки Socket.io. В первом методе вся работа по обмену сообщениями между пользователями возложена на Node.js, во втором методе Node.js используется только для отправки сообщений; принимает и обрабатывает сообщения основной сервер.

Список литературы

1. Matthias Wimmer. XMPP Standards. [Электронный ресурс]. - Режим доступа: http://www.ibm.com/developerworks/ru/library/wa-reverseajax2/ - [Standards-JIG] proto-JEP: Smart Presence Distribution.
2. Moffitt J. Professional XMPP Programming with JavaScript and jQuery / J. Moffitt. – Wrox Press, 2010 – 480 p.
3. Матье Карбу. Reverse Ajax. [Электронный ресурс]. - Режим доступа: http://www.ibm.com/developerworks/ru/library/wa-reverseajax2/ - Часть 2. WebSockets.
4. Пауэрс Ш. Изучаем Node.js / Ш. Пауэрс. – СПб: Питер, 2014. – 400 с.: ил. – (Серия «Бестселлеры O'Reilly»).
5. Node.js in Action / [M. Cantelon, M. Harter, T.J. Holowaychuk, N. Rajlich]. – New York: Manning Publications Co., 2014 – 417 p.