УДК 004.75

SOCKJS ДЛЯ АСИНХРОННОГО ОБМЕНА ДАННЫМИ В РЕАЛЬНОМ ВРЕМЕНИ

М.А. БАЛАБАШ
(*Представлено: канд. физ.-мат. наук, доц. А.Ф. ОСЬКИН*)

В данной статье рассматривается библиотека с открытым исходным sockjs. Библиотека предоставляет согласованный кросс-браузерный Javascript API, который создает полнодуплексный междоменный канал связи с малой задержкой между браузером и веб-сервером для обмена данными в реальном времени.

Введение. Интернет создавался главным образом на основе так называемой парадигмы запросов и ответов, реализованной с помощью протокола HTTP. Она заключается в том, что пока клиент не отправит запрос на открытие следующей страницы, она не загрузится. Примерно с 2005 г. с появлением технологии AJAX работа в Интернете стала более динамичной. При этом обмен данными по протоколу HTTP все равно инициировался клиентом, и это требовало от пользователя выполнения определенных действий или периодического опроса сервера для загрузки обновленной информации.

Технологии, позволяющие серверу отправлять клиенту новые данные в момент их появления, существуют довольно давно. Они известны как Push или Comet. Один из распространенных способов создания иллюзии соединения, инициируемого сервером, – это так называемый метод длинного опроса. Его суть в том, что клиент создает подключение к HTTP- серверу и оно не закрывается до отправки ответа. Если на сервере действительно есть обновленные данные, он отправляет ответ (в других технологиях для этого используются Flash, multipart-запросы XHR и особые HTML-файлы). Технология длинного опроса, как и другие подобные приемы, прекрасно работает. Вы пользуетесь ими каждый день, например, в чате Gmail.

Однако у этого метода есть один недостаток: он использует протокол HTTP, что плохо подходит для приложений с низким временем реакции. В частности, я имею в виду финансовые приложения с непрерывным потоком данных (биржи) и другие сетевые приложения в реальном времени.

Основной раздел. WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

WebSocket разработан для реализации в веб-браузерах и веб-серверах, но он может быть использован для любого клиентского или серверного приложения. Протокол WebSocket — это независимый протокол, основанный на протоколе TCP. Он делает возможным более тесное взаимодействие между браузером и веб-сайтом, способствуя распространению интерактивного содержимого и созданию приложений реального времени.

Под капотом SockJS всегда пытается использовать нативные WebSockets. Если это не удается, он может использовать различные транспортные протоколы для конкретного браузера и представлять их в виде абстракций, подобных WebSocket.

Основными сферами применения технологии являются:

- Быстрый обмен данными для онлайн игр, чатов и пр.;
- Приложения с интенсивным обменом данными, требовательные к скорости обмена и каналу;
- Push уведомления;
- IoT-приложения;
- Удалённый доступ.

Преимущества данного подхода:

1. Высокую скорость и эффективность передачи обеспечивает малый размер передаваемых данных, который иногда даже будет помещаться в один TCP-пакет — здесь, конечно, же все зависит от вашей бизнес-логики.

2. В отличие от HTTP веб-сокеты не имеют ограничений на время жизни в неактивном состоянии. Это значит, что больше не надо периодически обновлять соединение, т.к. его не вправе закрывать всякие прокси-серверы. Значит, соединение может висеть в неактивном виде и не требовать ресурсов. Конечно, можно возразить, что на сервере будут забиваться TCP-сокеты. Для этого достаточно использовать хороший мультиплексор, и сервер легко потянет до миллиона открытых соединений.

3. Как известно в HTTP предусмотрено ограничение на число одновременных открытых сессий к одному серверу. Из-за этого если у вас много различных асинхронных блоков на странице, то вам приходилось делать не только серверный, но и клиентский мультиплексор. К счастью, это ограничение не распространяется на веб-сокеты. Открываете столько, сколько вам нужно. А сколько использовать — одно (и через него все мультиплексировать) или же наоборот — на каждый блок свое соединение — решать вам. Исходите из удобства разработки, нагрузки на сервер и клиент.

4. Возможность разрабатывать кросс-доменные приложения. AJAX имеет ряд ограничений на создание кросс-доменных запросов. WebSockets не имеет таких ограничений.

Sockjs подходит для использования на клиенте и сервере. Для работы с технологией WebSockets на сервере существует множество реализаций socks на других языках программирования (node.js, python, erlang, java, scala, rust, go, ruby).

Протоколом проводной связи (установления соединения и обмена данными между клиентом и сервером) для технологии WebSocket является RFC6455. Последние версии Chrome и Chrome для Android полностью соответствуют спецификации RFC6455, в том числе и в части отправки двоичных сообщений. Этот протокол также поддерживают браузеры Firefox 11+ и Internet Explorer 10+.

Работа с библиотекой не представляет большой сложности.

Пример создания сервера на node.js:

```
const http = require('http');
const sockjs = require('sockjs');
const echo = sockjs.createServer({ prefix:'/echo' });

echo.on('connection', function(conn) {
  conn.on('data', function(message) {
    conn.write(message);
  });
  conn.on('close', function() {});
});

const server = http.createServer();
echo.attach(server);
server.listen(9999, '0.0.0.0');
```

На клиенте, после загрузки библиотеки, необходимо создать экземпляр SockJS объекта с которым мы будем работать. Доступные параметры:

- server — строка URL для фактического подключения к данным;
- transports — позволяет задать реализации транспорта, которые будут использованы если WebSockets не доступен;
- sessionId — идентификатор сеанса для клиента и сервера;
- timeout — минимальный таймаут в миллисекундах для использования в реализациях транспорта.

После создания объекта нам доступны следующие события для обработки:

- onopen — при установке соединения с сервером;
- onmessage — при получении сообщения от сервера;
- onclose — после закрытия соединения с сервером.

Дополнительно можно отметить, что соединение может устанавливаться между клиентами (которые поддерживают WebSockets) без участия сервера.

Заключение. WebSockets — технология, которая позволяет создавать интерактивное соединение между клиентом и сервером для обмена сообщениями в режиме реального времени. Веб-сокеты, в отличие от HTTP, позволяют работать с двунаправленным потоком данных, что делает эту технологию совершенно уникальной.

ЛИТЕРАТУРА

1. Wikipedia [Электронный ресурс] WebSocket. – Режим доступа: https://en.wikipedia.org/wiki/WebSocket. – Дата доступа: 27.09.19.
2. IETF Documents [Электронный ресурс] The WebSocket Protocol. – Режим доступа: https://tools.ietf.org/html/rfc6455. – Дата доступа: 27.09.19.
3. Github [Электронный ресурс] sockjs-client. – Режим доступа: https://github.com/sockjs/sockjs-client. – Дата доступа: 27.09.19.