ASP.NET SignalR - это библиотека для ASP.NET разработчиков, которая упрощает добавление в приложения компонентов, работающих в реальном времени. Функциональность, работающая в реальном времени – это способность сервера отдать свежие данные подключенным клиентам немедленно, вместо того, чтобы ждать пока клиенты запросят эти данные.

* SignalR также раскрывает просторы для нового типа веб-приложений, где требуется быстрый обмен данными с сервером, а это игровые приложения в реальном времени.
* SignalR имеет простой API для вызовов удаленных процедур от сервера к клиенту (RPC server-to-client), которые вызывают Javascript функции в клиентских браузерах из кода .NET сервера.
* SignalR также имеет API ***для управления соединениями*** (например подключением или отключением) и ***группировкой соединений***.
* SignalR поддерживает функциональность "server push" (толкни сервер), когда код на сервере может отправить сообщения в браузеры, используя Remote Procedure Calls (RPC), это быстрее работает, чем обычная модель "запрос-ответ", существующая в вебе сегодня.

SignalR приложения могут масштабироваться на тысячах клиентах, используя Service Bus, SQL Server или Redis.

SignalR поставляется с открытым кодом, который доступен на GitHub.

### ****Real-time веб приложения****

Наверняка многие пользовались [Facebook](http://www.facebook.com), [Twitter](https://twitter.com/) или другими социальными сервисами и замечали, что когда приходят обновления новостей или статусов, страница обновляется в реальном времени и вы постоянно видите последнюю актуальную информацию. В [Google Docs](https://drive.google.com) можно совместно редактировать документы, и все обновления синхронизированы и появляются у вас сразу же, как только их сделает другой пользователь. Это примеры сервисов, которые можно назвать real-time веб-приложениями. Основной их отличительной способностью является то, что они “ломают” привычную нам всем модель работы в вебе – запрос-ответ, благодаря чему пользователи видят обновление данных сразу же, как только они появляются на сервере.

***Если не знать, насколько далеко вперед шагнули технологии за последние несколько лет, то можно предположить, что все подобные приложения реализованы при помощи периодического опрашивания сервера обычными Ajax-запросами***, т.н. polling. Выглядит похоже, но это не так, в чем можно легко убедиться, просмотрев логи запросов из браузера на сервер в Firebug, Fiddler или Chrome developer tools.

### ****Способы реализации****

Итак, какие же есть способы реализации подобной функциональности, их достоинства и недостатки.

|  |  |  |  |
| --- | --- | --- | --- |
| **техника** | **Описание** | **Преимущества** | **Недостатки** |
| Polling | Постоянный опрос сервера Ajax-запросами | + простота реализации + поддержка во всех современных браузерах | - задержка в результатах - при уменьшении задержки существенно увеличивается нагрузка на сервер |
| Long Polling | Ajax-запросы, идущие один за другим, но каждый запрос держится открытым в течение нескольких минут | + сниженная нагрузка на сервер по сравнению с обычным Polling + уменьшенный трафик + поддержка во всех современных браузерах | - больше одновременно открытых соединений, т.к. каждый запрос живет дольше |
| Server-Sent Events | Новый стандарт HTML5, работающий поверх HTTP. Позволяет создавать долгоживущее соединение с сервером, чтобы сервер мог отправлять данные на клиент | + нет необходимости постоянно пересоединяться с сервером + нет изменений на стороне сервера, поэтому работает на всех современных веб-серверах | - не поддерживается в IE ([даже в IE10](http://caniuse.com/#feat=eventsource)) - работает только в направлении сервер –> клиент (на сервер можно отправлять обычные Ajax запросы) |
| WebSockets | Новый протокол (ws:// и wss://), работающий поверх TCP на одном уровне с HTTP. Позволяет создавать двустороннее долгоживущее соединение с клиентом | + нет необходимости постоянно пересоединяться с сервером + работает в двустороннем режиме | - поддерживается не во всех веб-серверах (IIS8) - поддерживается не во всех браузерах ([в IE7-9, Android](http://caniuse.com/#feat=websockets)) |

**WebSocket** - является веб-технологией, обеспечивающей полнодуплексные каналы связи поверх одного TCP соединения. Протокол WebSocket никак не связан с протоколом HTTP. Единственное, что протокол HTTP используется для проверки того, что стороны поддерживают протокол WebSocket. Если WebSockets обеими сторонами подтверждается, то весь последующий процесс обмена данными проходит без использования HTTP. Естественно, если одна из сторон (обычно таковыми являются веб-браузеры) запрашивает данную информацию, то это значит, что она уже поддерживает WebSocket.

**SignalR** – это библиотека для создания многопользовательских real-time ASP.NET (и не только) приложений. Она состоит из набора **серверных** и **клиентских** библиотек, и представляет собой абстракцию над целым набором транспортов. Оpen source, лежащий на GitHub: <https://github.com/SignalR>

SignalR был придуман и реализован двумя разработчиками Microsoft: [Damian Edwards](https://twitter.com/DamianEdwards) и [David Fowler](https://twitter.com/davidfowl).

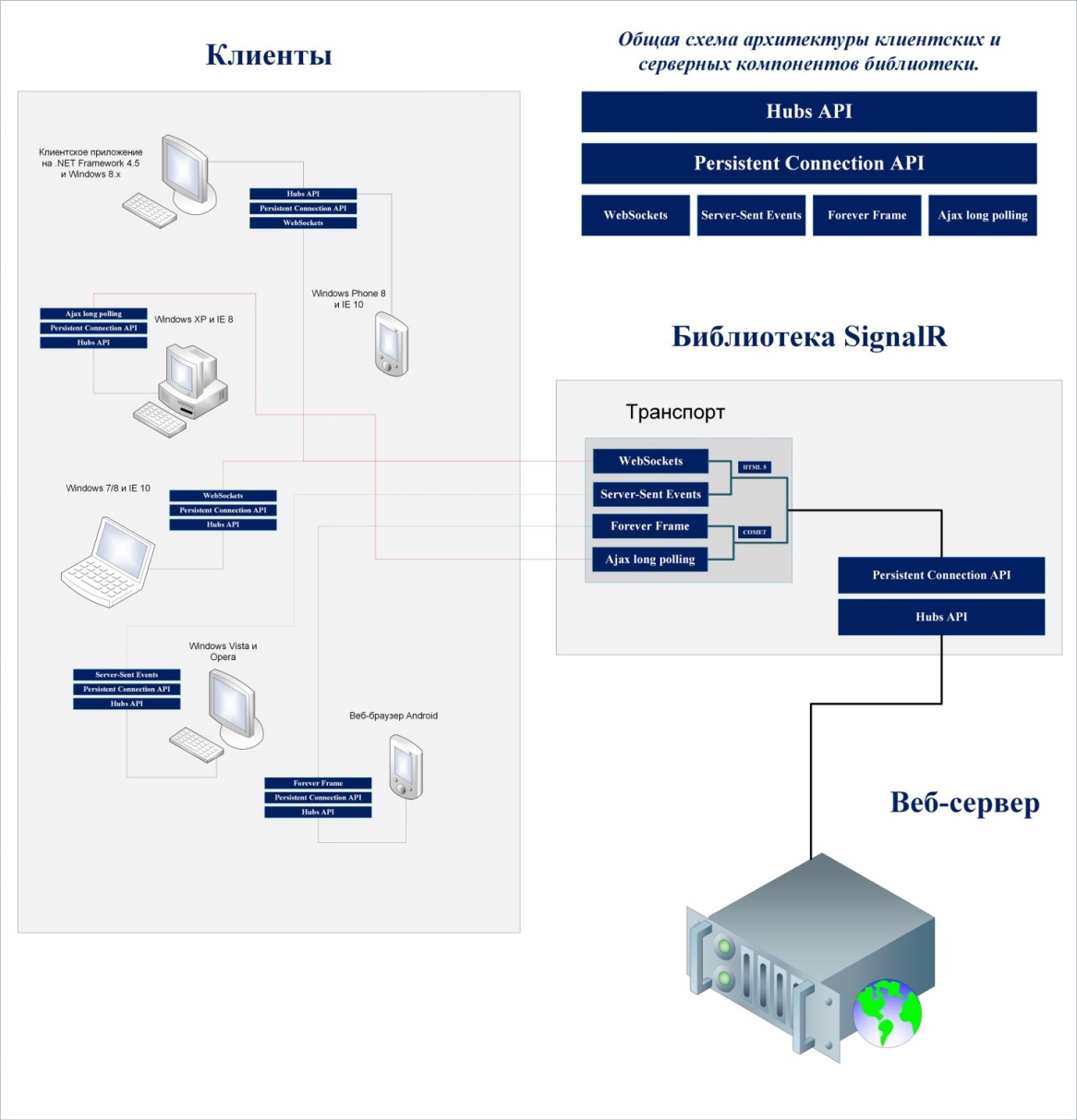
Если **WebSocket** недоступен, то SignalR использует технологию **Server Sent Events**, при ее поддержке сервером и клиентом.

Если технология **Server Sent Events** также недоступна, то используются скрытые фреймы - **Forever Frames**.

И если **Forever Frames** также недоступны, то используется **Long Polling**.

SignalR использует нижеперечисленные технологии, причём используется так называемый "резервный механизм" (от лучших к худшим, но более доступным). Порядок использования следующий.

1. Протокол WebSocket (если обе стороны поддерживают его).
2. События, посылаемые сервером (технология Server-Sent Events).
3. Использование скрытого фрейма (Forever Frame).
4. Использование длинного AJAX опроса  (Ajax long polling).



<http://www.msdr.ru/36/>

SignalR использует две модели взаимодействия сервера и клиента: Persistent Connection и хабы.

## Hubs(чаще всего)

Создадим новое приложение ASP.NET MVC 4

Установим через Manage NuGet -> Microsoft ASP.NET SignalR

Добавляем Item -> OWIN Startup class

[assembly: OwinStartup(typeof(WebAppSignalR.Startup))]

namespace WebAppSignalR

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

***app.MapSignalR();***//добавляем строку

}

}

}

в папку *Models* класс ChatUser

public class ChatUser

{

public string ConnectionId { get; set; }//!!задается средой и является строковым!!!

public string Name { get; set; }

}

Добавим в проект новую папку *Hubs*. В ней будут находиться хабы нашего приложения. В эту папку добавим новый класс *ChatHub.cs* с шаблона файла SignalR Hub Class (v2). ***Для поиска этого файла введем в строке поиска SignalR.***

В данном случае мы используем хабы. Для этого создаем свой хаб ChatHub, который наследуется от класса Hub.

public class ChatHub : Hub

{

static List<ChatUser> Users = new List<ChatUser>();

// Отправка сообщений

public void Send(string name, string message)

{

Clients.All.addMessage(name, message);

}

// Подключение нового пользователя

public void Connect(string userName)

{

var id = Context.ConnectionId;

if (!Users.Any(x => x.ConnectionId == id))

{

Users.Add(new MyUser { ConnectionId = id, Name = userName });

// Посылаем сообщение текущему пользователю

Clients.Caller.onConnected(id, userName, Users);

// Посылаем сообщение всем пользователям, кроме текущего

Clients.AllExcept(id).onNewUserConnected(id, userName);

}

}

// Отключение пользователя

public override System.Threading.Tasks.Task OnDisconnected(bool stopCalled)

{

var item = Users.FirstOrDefault(x => x.ConnectionId == Context.ConnectionId);

if (item != null)

{

Users.Remove(item);

var id = Context.ConnectionId;

Clients.All.onUserDisconnected(id, item.Name);

}

return base.OnDisconnected(stopCalled);

}

}

#### Формат вызова методов клиента

* Вызов метода на всех клиентах: **Clients.All.addMessage(name, message);**
* Вызов метода только на текущем клиенте, который обратился к серверу: **Clients.Caller.addMessage(name, message);**
* Вызов метода на всех клиентах, кроме того, который обратился к серверу: **Clients.Others.addMessage(name, message);**
* Вызов метода только у клиента с определенным id: **Clients.Client(Context.ConnectionId).addMessage(name, message);**
* Вызов метода на всех клиентах, кроме клиента с определенным id: **Clients.AllExcept(connectionId).addMessage(name, message);**
* Вызов метода на всех клиентах указанной группы: **Clients.Group(groupName).addMessage(name, message);**
* Вызов метода на всех клиентах указанной группы, за исключением клиента, у которого id - connectionId: **Clients.Group(groupName, connectionId).addMessage(name, message);**
* Вызов метода на всех клиентах указанной группы, за исключением обратившегося к серверу клиента: **Clients.OthersInGroup(groupName).addMessage(name, message);**

// разослать всем

Clients.All.addMessage(message);

// всем, кроме меня

Clients.Others.addMessage(message);

// всем, кроме определённого идентификатора соединения

Clients.AllExcept(Context.ConnectionId).addMessage(message);

// тому, кто прислал

Clients.Caller.addMessage(message);

// отослать всем в группе "foo"

Clients.Group("foo").addMessage(message);

// отослать всем из группы "foo", кроме того кто прислал

Clients.OthersInGroup("foo").addMessage(message);

// отослать всем из группы "foo", кроме определённого идентификатора соединения

Clients.Group("foo", Context.ConnectionId).addMessage(message);

// отослать клиенту с определённым идентификатором соединения

Clients.Client(Context.ConnectionId).addMessage(message);

Далее в выражении следует метод addMessage. Этот метод объявляется ***на стороне клиента в коде javascript***.

В методе Connect мы сначала получаем id текущего пользователя, который и обратился к методу Connect, через объект Context.ConnectionId. **Этот id задается средой и хранит строковое значение (не числовое).** Затем вызываем методы на клиенте через объект Clients.

В Index.cshtml добавим код

@{

ViewBag.Title = "Chat";

}

<h2>Chat-room</h2>

<div class="main">

<div id="loginBlock">

Введите логин:<br />

<input id="txtUserName" type="text" />

<input id="btnLogin" type="button" value="Войти" />

</div>

<div id="chatBody" hidden>

<div id="header"></div>

<div id="inputForm">

<input type="text" id="message" />

<input type="button" id="sendmessage" value="Отправить" />

</div>

<div id="chatroom"></div>

<div id="chatusers">

<p><b>Все пользователи</b></p>

</div>

</div>

<input id="hdId" type="hidden" />

<input id="username" type="hidden" />

</div>

<script src="~/Scripts/jquery-2.2.2.min.js"></script>

<script src="~/Scripts/jquery.signalR-2.2.0.min.js"></script>

<!--Ссылка на автоматически сгенерированный скрипт хаба SignalR -->

<script src="~/signalr/hubs"></script>

<script>

$(function () {

$('#chatBody').hide();

$('#loginBlock').show();

// Ссылка на автоматически-сгенерированный прокси хаба

var chat = $.connection.chatHub;

// Объявление функции, которая хаб вызывает при получении сообщений

chat.client.addMessage = function (name, message) {

// Добавление сообщений на веб-страницу

$('#chatroom').prepend('<p><b>' + htmlEncode(name)

+ '</b>: ' + htmlEncode(message) + '</p>');

};

// Функция, вызываемая при подключении нового пользователя

chat.client.onConnected = function (id, userName, allUsers) {

$('#loginBlock').hide();

$('#chatBody').show();

// установка в скрытых полях имени и id текущего пользователя

$('#hdId').val(id);

$('#username').val(userName);

$('#header').html('<h3>Добро пожаловать, ' + userName + '</h3>');

// Добавление всех пользователей

for (i = 0; i < allUsers.length; i++) {

AddUser(allUsers[i].ConnectionId, allUsers[i].Name);

}

}

// Добавляем нового пользователя

chat.client.onNewUserConnected = function (id, name) {

AddUser(id, name);

}

// Удаляем пользователя

chat.client.onUserDisconnected = function (id, userName) {

$('#' + id).remove();

}

// Открываем соединение

//$.connection.hub.start({ transport: 'longPolling' })

$.connection.hub.start({ transport: 'longPolling' }).done(function () {

$('#sendmessage').click(function () {

// Вызываем у хаба метод Send

chat.server.send($('#username').val(), $('#message').val());

$('#message').val('');

});

// обработка логина

$("#btnLogin").click(function () {

var name = $("#txtUserName").val();

if (name.length > 0) {

chat.server.connect(name);

}

else {

alert("Введите имя");

}

});

});

});

// Кодирование тегов

function htmlEncode(value) {

var encodedValue = $('<div />').text(value).html();

return encodedValue;

}

//Добавление нового пользователя

function AddUser(id, name) {

var userId = $('#hdId').val();

if (userId != id) {

$("#chatusers").append('<p id="' + id + '"><b>' + name + '</b></p>');

}

}

</script>

SignalR сгенерирует для нас хаб. Его можно посмотреть в браузере localhost:3819/signalr/hubs

|  |  |
| --- | --- |
| var chat = $.connection.chatHub;  // Объявление функции, которуя хаб вызывает при получении сообщений  chat.client.addMessage = function (name, message) {  // Добавление сообщений на веб-страницу  $('#chatroom').prepend('<p><b>' + htmlEncode(name)  + '</b>: ' + htmlEncode(message) + '</p>');  }; | public class ChatHub : Hub  {  static List<MyUser> Users = new List<MyUser>();  // Отправка сообщений  public void Send(string name, string message)  {  Clients.All.addMessage(name, message);  } |