электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 04, апрель 2016

УДК 004.77

Фреймворк SignalR в ASP.NET MVC

Стародубцев А.В., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю.Е., к.т.н., доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»
gapyu@bmstu.ru

Введение. С развитием интернет-приложений появилась необходимость в функционале, обеспечивающем операции в реальном времени – получение данных с сервера, отправку на него данных, рассылку данных с сервера на большое число клиентов, и всё это без перезагрузки страницы – без нажатия клавиши *refresh*. Одним из вариантов реализации такого функционала является библиотека SignalR. Этот фреймворк можно использовать в веб-формах, WPF и консольных приложениях, но наибольшее распространение он получил в приложениях на ASP.NET MVC.

Границы применимости. Использование SignalR библиотеки обусловлено необходимостью постоянного обновления данных в режиме реального времени. В случае, если данные обновляются редко и/или с определенной заданной (заранее известной) периодичностью, стоит рассмотреть возможность использования, к примеру, AJAX запросов и отказа от SignalR. Также важна и нагрузка на сервер, поэтому и инструмент следует подбирать в зависимости от доступных ресурсов.

Хабы и Постоянные Подключения. Библиотека предоставляет два варианта работы: через хабы (*hubs*) и через постоянные подключения (*persistent connections*). Постоянные подключения – коммуникации на низком уровне абстракции взаимодействия. Другой вариант, хабы, представляет собой некоторое обобщение первого, это более высокий уровень абстракции.

Хабы. Как было сказано ранее, хабы работают на более высоком уровне абстракции, нежели постоянные подключения. Есть две части хабов – клиентская и серверная; схематично это изображено на рис. 1.

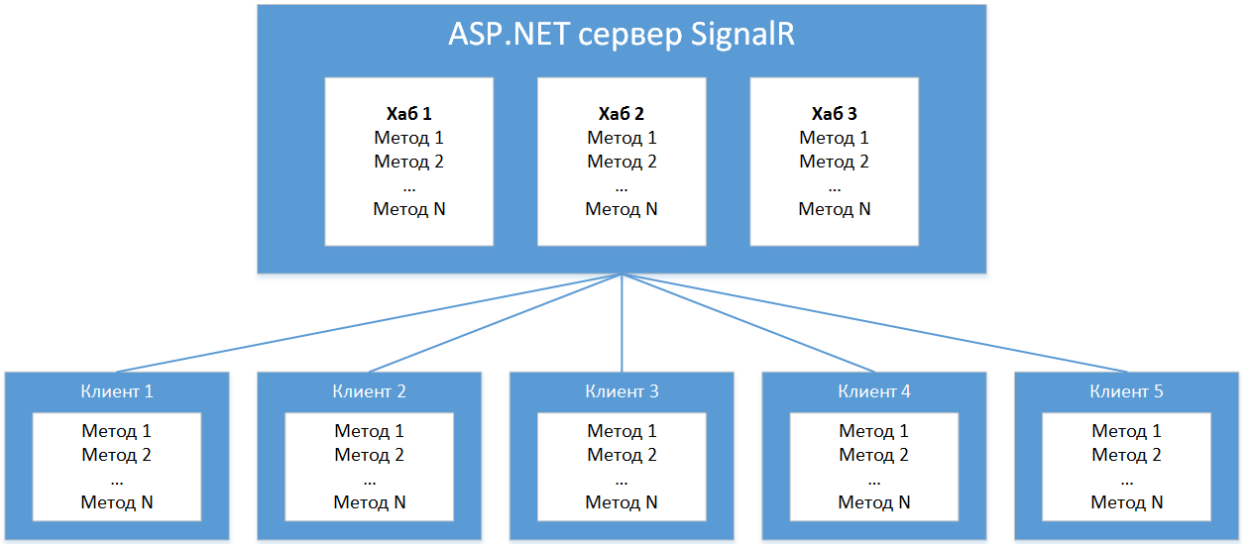

Рис. 1. Схема коммуникации клиента и сервера в SignalR

Клиентская обычно пишется с использованием JS, но есть реализации и под iOS, WP, Android, .NET. Серверная часть – ASP.NET (.NET и Mono).

Вызов методов сервера с клиента – достаточно простой процесс, а вот обратный – обращение сервера к клиенту – выполняется несколько сложнее: здесь уже используется механизм постоянных подключений.

В SignalR существует два типа библиотек:

1. Клиентские
2. Серверные

Клиентские, на JavaScript, .NET и других платформах, предназначены для методов клиента, эти методы могут вызываться с сервера, и для реализации механизма вызова серверных методов.

Серверные, ASP.NET библиотеки, предназначены для серверных методов. Они могут вызываться с клиентов. Также эти библиотеки могут использоваться для вызова некоторых методов, определённых на клиенте, с серверной стороны. Таким образом, хабы представляют собой не что иное, как реализацию удалённого вызова процедур или RPC.

В качестве простого примера работы с хабами, рассмотрим простой чат на SignalR между клиентами разных браузеров.

Для того, чтобы начать работу с хабами в SignalR 2.2.0, необходимо сперва добавить в проект OWIN (Open Web Interface for .NET) *startup* класс, в тело функции *Configuration* которого необходимо внести следующую строку:

```
app.MapSignalR();
```

Далее, создается класс хаба, который наследуется от базового *Hubs*.

```
public class chatHub : Hub
	{
		public void broadcastMessage(string text)
		{
			Clients.All.displayText(text);
		}
	}
```

Здесь, *broadcastMessage* – метод, вызываемый клиентом на сервере, *displayText* – метод, вызываемый сервером на клиенте. *All* указывает на то, что сообщение будет отправлено всем клиентам (о других возможностях будет рассказано позднее). Важно отметить, что метод *displayText* связывается уже непосредственно при выполнении.

Теперь, опишем файл разметки и, в нём же, скрипт.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
	<title>Tester</title>
</head>
<body>
	<div>
		<input type="text" id="msg" />
		<input type="button" id="broadcast" value="Broadcast" />
		<div id="messages"></div>
	</div>

	<script src="../Scripts/jquery-1.6.4.min.js"></script>
	<script src="../Scripts/jquery.signalR-2.2.0.js"></script>
	<script src="signalr/hubs"></script>


	<script type="text/javascript">
		$(document).ready(function ()
		{
			var broadcaster = $.connection.chatHub;


			$.connection.hub.start().done(function () {
				$("#broadcast").click(function () {
					broadcaster.server.broadcastMessage($('#msg').val());
				});
			});
			broadcaster.client.displayText = function (text) {
				$('#messages').append('<li>' + text + '</li>');
				};
		})
	</script>
</body>
</html>
```

Для начала, необходимо подключить соответствующие библиотеки JavaScript JQuery, SignalR и библиотеку хабов. Создается объект *broadcaster*, который связывается с

хабом chatHub, он нужен, чтобы связываться с сервером и клиентами. Функции *displayText* и *broadcastMessage* – функции соответствующие одноименным в хабе *chatHub*, но уже реализованные на JavaScript. Первые метод – *displayText* – вызывается с сервера и построчно заполняет *div*-элемент сообщениями чата. Второй – *broadcastMessage* – вызов метода сервера, рассылающего сообщения на клиенты. Эта функция связывается с кнопкой. Результат выполнения кода представлен на рис. 2.

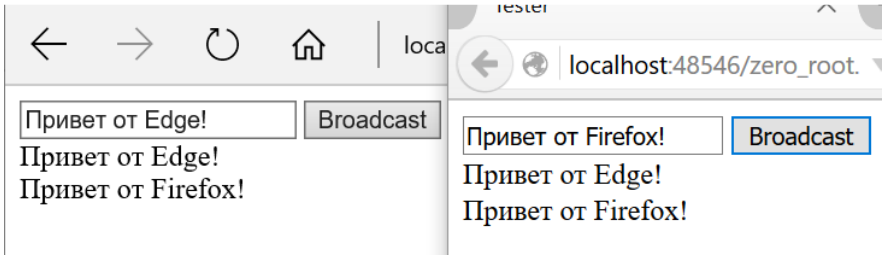

Рис. 2. Общение клиентов через хабы

В примере чат запущен с двух браузеров – Firefox и Edge, с каждого было отправлено соответствующее сообщение в чат.

Библиотека SignalR также предлагает различные варианты рассылки сообщений в случае, если нам необходимо обращаться к какому-то конкретному клиенту или группе клиентов. Для этого существуют механизмы групп и рассылки по конкретным клиентам.

Во-первых, можно отправлять сообщения всем пользователям. Во-вторых, можно отправлять сообщения всем пользователям, кроме того, который вызывает функцию рассылки. В-третьих, можно отправлять сообщение тому, кто вызывает функцию. Дополнительно существует механизм, позволяющий отослать сообщение какому-то конкретному пользователю сообщение (с использованием *ConnectionID*) или же всем, кроме него.

В группы можно добавлять клиентов – создаётся задача (*Task*), которая прописывается в хабе. Параметром для добавления в группу является строковая константа – имя группы; текущий клиент определяется параметром *ConnectionID*.

```
public Task Join(string groupName)
{
	return Groups.Add(Context.ConnectionId, groupName);
}
```

Для групп существует механизм рассылки по всем членам группы:

```
public void BroadcastMessage(Person person)
{
	Clients.Group(GroupName).displayText(Name, Message);
}
```

по всем членам группы, кроме конкретного пользователя:

```
public void BroadcastMessage(Person person)
```

```
{
	Clients.OthersInGroup(GroupName).displayText(Name, Message);
}
```

рассылка конкретному пользователю:

```
public void BroadcastMessage(Person person)
{
	Clients.Group(GroupName, Context.ConnectionId).displayText(Name, Message);
}
```

Для того, чтобы убрать пользователя из группы, необходимо в хабе создать соответствующую задачу:

```
public Task Leave(string groupName)
{
	return Groups.Remove(Context.ConnectionId, groupName);
}
```

Вызов методов производится стандартным образом, к примеру, вход в группу, посылка сообщения и выход из группы на JavaScript:

```
$("#broadcast").click(function () {
	broadcaster.server.join($('#groupName').val());
	broadcaster.server.broadcastMessage({ Name: $('#name').val(),
	Message: $('#message').val(), Group: $('#groupName').val() });
	broadcaster.server.leave($('#groupName').val());
});
```

Здесь *groupName*, *name* и *message* – соответствующие значения в input на html-странице.

Постоянные подключения

Постоянные подключения – это надежные полнодуплексные соединения с маленькой задержкой при передаче данных, с дополнительными возможностями по безопасности.

Клиент посылает запрос на установление связи. Сервер отвечает на этот запрос с информацией для установления соединения. Клиент использует эту информацию для установления соединения с лучшими параметрами. Клиент посылает запрос на подключение по налаженному соединению. Как только сервер принимает запрос на подключение, постоянное подключение считается установленным.

Порядок работы следующий: вначале устанавливается постоянное соединение с сервером, затем, посылаются *keep-alive* пакеты для поддержания соединения или же происходит обмен данными – запрос данных с сервера; посылка данных, отложенных для передачи на сервер. Или же, если соединение было разорвано из-за таймаута, происходит переподключение. В заключении, когда в постоянном подключении уже нет нужды, происходит разрыв соединения: клиент шлет команду отключения, а сервер, если получает эту команду, закрывает соединение, либо ждет таймаута. Пример работы изображен на рисунках 3 и 4.

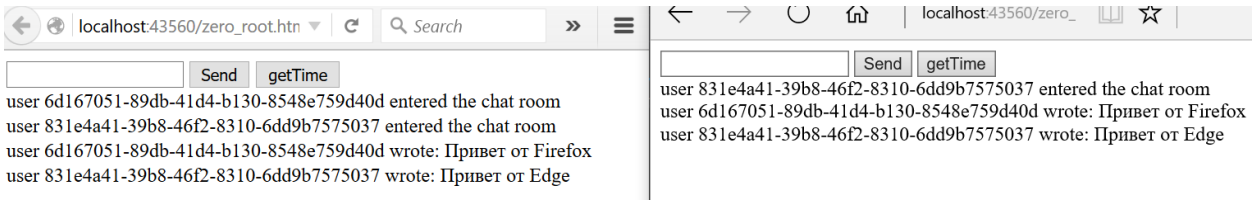

Рис. 3. Общение клиентов посредством постоянных подключений.

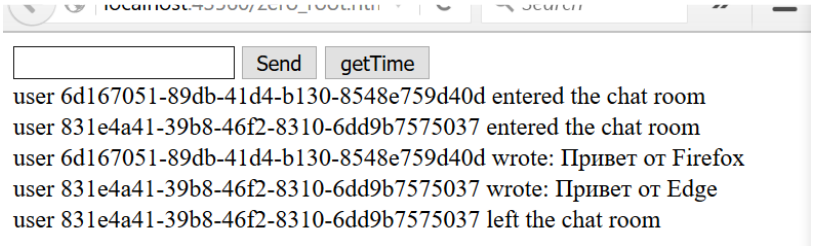

Рис. 4. Событие при отключении одного из клиентов

Схема работы с постоянными подключениями начинается с того же шага – создания OWIN *startup* класса, в тело функции *Configuration* которого необходимо внести следующие строки:

app.MapSignalR();

app.MapSignalR<chatPC>("/AlexPC");

Где "/AlexPC" задает имя для подключения.

Далее создается класс, наследующийся от базового *PersistentConnection*. В нем описываются события *OnConnected*, *OnReceived* и *OnDisconnected* – при подключении, при получении сообщения и при отключении соответственно – являющиеся типом *Task* – задачами. В них, схожим с хабами образом, определяется серверная логика, но, в отличие от хабов, автоматически передается и ID подключения – *connectionId*.

```
public class chatPC : PersistentConnection
{
	protected override Task OnConnected(IRequest request, string connectionId)
	{
	return Connection.Broadcast("user " + connectionId + " entered the chat room");
	}

	protected override Task OnReceived(IRequest request, string connectionId, string
	data)
	{
	return Connection.Broadcast("user " + connectionId + " wrote:\n" + data);
	}

	protected override Task OnDisconnected(IRequest request, string connectionId, bool
	stopCalled)
	{
	return Connection.Broadcast("user " + connectionId +" left the chat room");
	}
}
```

Создаем страницу разметки:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
	<title>Tester</title>
</head>
<body>
	<div>
		<input type="text" id="msg" />
		<input type="button" id="send" value="Send" />
		<input type="button" id="getTime" value="getTime" />
		<div id="messages"></div>
	</div>

	<script src="../Scripts/jquery-1.6.4.min.js"></script>
	<script src="../Scripts/jquery.signalR-2.2.0.js"></script>


	<script type="text/javascript">
		$(function () {

			var myConnection = $.connection("/AlexPC");
			myConnection.received(function (data) {
				$("#messages").append("<li>" + data + "</li>");
			});

			myConnection.start().done(function () {
				$('#send').click(function () {
					myConnection.send($('#msg').val());
					$('#msg').val('');
				});
			});
		});
	</script>
</body>
</html>
```

Здесь *myConnection* задает соединение по ключу, определенному в хабе. Методы *send* и *received* определяют соответствующую логику посылки и принятия сообщений.

Также, как и в хабах, в постоянных подключениях существует механизм группировки. GroupManager позволяет добавлять членов в группу по *connectionId*, удалять членов из группы, осуществлять рассылку членам группы. Идентификаторам группы служит символьная строка. Группа существует с момента добавления в неё первого члена и до тех пор, пока в ней кто-то находится. Добавление членов в группу происходит путем добавления следующих сток в задачу OnConnected:

```
protected override Task OnConnected(IRequest request, string connectionId)
{
string groupName = request.QueryString["roomName"];
if (!string.IsNullOrWhiteSpace(groupName))
this.Groups.Add(connectionId, groupName);
return base.OnConnected(request, connectionId);
}
```

Удаление клиентов из группы происходит путем добавления следующих строк в задачу OnDisconnected:

```
protected override Task OnDisconnected(IRequest request, string connectionId, bool
stopCalled)
{
string groupName = request.QueryString["roomName"];
if (!string.IsNullOrWhiteSpace(groupName))
this.Groups.Remove(connectionId, groupName);
return base.OnDisconnected(request, connectionId, stopCalled);
}
```

Отправка сообщений группе осуществляется путём добавления следующих строк в задачу OnReceived:

```
protected override Task OnReceived(IRequest request, string connectionId, string data)
{
string groupName = request.QueryString["roomName"];
return this.Groups.Send(groupName, data, connectionId);
}
```

При этом, здесь *connectionId* указывает, каким клиентам сообщение не отправляется. К примеру, в данном случае, сообщение не выдается отправителю из-за клиентской логики на JavaScript.

Помимо перечисленных событий (задач) и свойств, у постоянных подключений существуют и другие:

- *URL* – используется только клиентами на JavaScript для настройки соединения
- *ConnectionToken* – *ConnectionId* с приписанным идентификатором пользователя
- *KeepAliveTimeout* – значение, использующееся для настройки времени проверки активности соединения
- *DisconnectTimeout* – время до разрыва соединения
- *TransportConnectTimeout* – время, которое должен выждать клиент, до разрыва соединения или попытки соединения другим способом

Есть также и другие параметры, для более прецизионной настройки постоянных подключений. К тому же, для более гибкой работы с подключениями есть дополнительные методы работы с такими событиями как переподключение, смена состояния соединения, замедление соединения и прочими.

Клиенты

Помимо клиентов на JavaScript (веб браузеров), примеры которых демонстрировались выше, есть варианты клиентов на платформе .NET 4.0 и 4.5, Windows RT, Windows Phone 8, Silverlight 5, iOS и Android.

Заключение

Библиотека SignalR постоянно обновляется и модифицируется, так что примеры кода актуальны на момент написания статьи. У этой технологии есть и свои проблемы: так, например, сообщалось, что в хабах не отслеживалось отключение клиента по таймауту.

Но, несмотря на такие проблемы, библиотека позволяет работать с интернет-приложениями действительно в реальном времени и на различных клиентах.

Список литературы

[1] Keyvan N., Darren W. Pro ASP.NET SignalR – Real-Time Communication in .NET with SignalR 2.1. NY: Apress, 2014. 360 p.

[2] SignalR Official Web site. Available at: http://signalr.net/, accessed 15.12.2015.

[3] Хабрахабр, коллективный блог. Режим доступа: http://habrahabr.ru/company/dnevnik_ru/blog/167307/, дата обращения 15.12.2015.

[4] Хабрахабр, коллективный блог. Режим доступа: http://habrahabr.ru/post/135604/ (дата обращения 15.12.2015).

[5] Metanit – сайт о программировании. Режим доступа: http://metanit.com/sharp/mvc5/16.1.php (дата обращения 15.12.2015).

[6] ASP.NET Official Web site. Available at: http://www.asp.net/signalr/overview/getting-started/tutorial-getting-started-with-signalr-and-mvc, accessed 15.12.2015.