**在 Asp.NET MVC 中使用 SignalR 实现推送功能**

**罗朝辉 (**[**http://blog.csdn.net/kesalin**](http://blog.csdn.net/kesalin)**)**

**CC许可，转载请注明出处**

**一，简介**

Signal 是微软支持的一个运行在 Dot NET 平台上的 html websocket 框架。它出现的主要目的是实现服务器主动推送（Push）消息到客户端页面，这样客户端就不必重新发送请求或使用轮询技术来获取消息。

可访问其官方网站：<https://github.com/SignalR/> 获取更多资讯。

**二，实现机制**

SignalR 的实现机制与 .NET WCF 或 Remoting 是相似的，都是使用远程代理来实现。在具体使用上，有两种不同目的的接口：[**PersistentConnection**](https://github.com/SignalR/SignalR/wiki/PersistentConnection) 和 [**Hubs**](https://github.com/SignalR/SignalR/wiki/Hubs)，其中 PersistentConnection 是实现了长时间的 Javascript 轮询（类似于 Comet），Hub 是用来解决实时信息交换问题，它是利用 Javascript 动态载入执行方法实现的。SignalR 将整个连接，信息交换过程封装得非常漂亮，客户端与服务器端全部使用 JSON 来交换数据。

下面就 Hubs 接口的使用来讲讲整个流程：

1，在服务器端定义对应的 hub class;

2，在客户端定义 hub class 所对应的 proxy 类；

3，在客户端与服务器端建立连接（connection）；

4，然后客户端就可以调用 proxy 对象的方法来调用服务器端的方法，也就是发送 request 给服务器端；

5，服务器端接收到 request 之后，可以针对某个/组客户端或所有客户端（广播）发送消息。

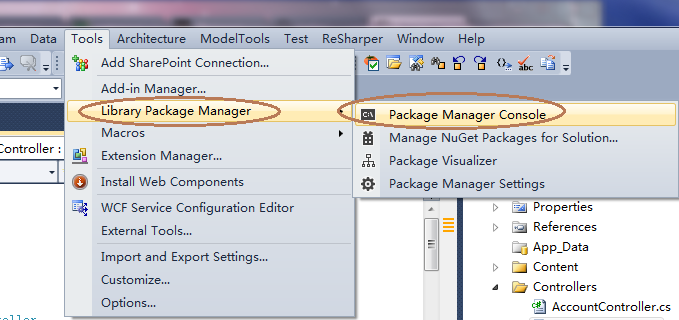
**三，Hub 示例教程**

**1，工具准备**

SignalR 运行在 .NET 4.5 平台上，所以需要安装 [.NET 4.5](http://www.microsoft.com/en-us/download/details.aspx?id=30653)。为了方便演示，本示例使用 ASP.NET MVC 在 Win 7 系统来实现。这需要安装 [ASP.NET MVC 3](http://go.microsoft.com/fwlink/?LinkID=208140) 或 [ASP.NET MVC 4](http://www.microsoft.com/en-us/download/details.aspx?id=30683)。

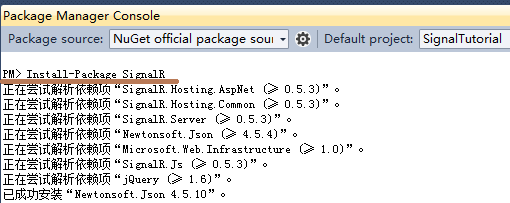
**2，建立工程**

打开 VS2010/VS2012 新建名为 SignalRTutorial 的 ASP.NET MVC 3 Web Application 工程，选择 Internet Application 模板， Razor 视图引擎以及勾选 Use HTMl 5 标签。



**3，安装 SignalR**

打开 NuGet 的 package manager console(Tools->Library package manager)，输入：install-package SignalR.Sample，回车安装。如图所示：



**4，实现 Hub 服务器端代码**

向工程中新建 SignalR 目录，在其中添加 ChatHub.cs 文件，内容如下：

namespace SignalTutorial.SignalR

{

[HubName("chat")]

public class Chat : Hub

{

public void Send(string clientName, string message)

{

//var toSelfinfo = "You had sent message " + message;

//Caller.addSomeMessage(clientName, toSelfinfo);

// Call the addMessage method on all clients

Clients.addSomeMessage(clientName, message);

//Clients[Context.ConnectionId].addSomeMessage(clientName, data);

}

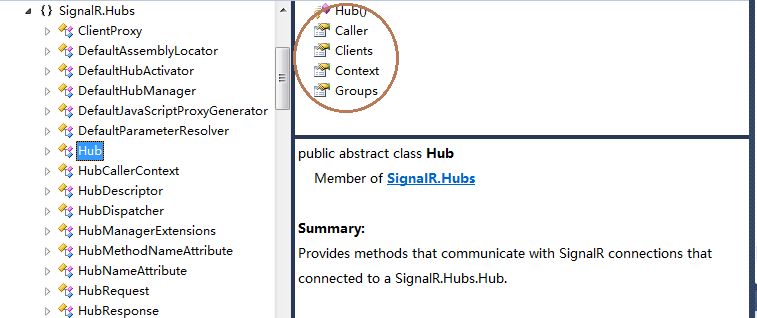
}

}

在上面的代码中：

1），HubName 这个特性是为了让客户端知道如何建立与服务器端对应服务的代理对象，如果没有设定该属性，则以服务器端的服务类名字作为 HubName 的缺省值；

2），Chat 继承自 Hub，从下面 Hub 的接口图可以看出：Hub 支持向发起请求者（Caller），所有客户端(Clients)，特定组（Group) 推送消息。



3），public void Send(string clientName, string message) 这个接口是被客户端通过代理对象调用的；

4），Clients 是 Hub 的属性，表示所有链接的客户端页面，它和 Caller 一样是 dynamic，因为要直接对应到 Javascript 对象；

5），Clients.addSomeMessage(clientName, message); 表示服务器端调用客户端的 addSomeMessage 方法，这是一个 Javascript 方法，从而给客户端推送消息。

6），总结：这里实现的服务很简单，就是当一个客户端调用 Send 方法向服务器发送 message 后，服务器端负责将该 message 广播给所有的客户端（也可以给特定组或特定客户端，见屏蔽代码），以实现聊天室的功能。

**5，实现 Hub 客户端代码**

**1），引用 SignalR Javascript**

为了简化引用 SignalR 脚本操作，我直接在 View/Shared/\_Layout.cshtml 中引入 SignalR 及其他脚本：

<head>

<meta charset="utf-8" />

<title>@ViewBag.Title</title>

<link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />

<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/jquery-1.6.4.js")" type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/jquery-ui-1.8.24.js")" type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/jquery.signalR-0.5.3.js")" type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/modernizr-1.7.min.js")" type="text/javascript"></script>

<script src="@Url.Content("~/signalr/hubs")" type="text/javascript"></script>

</head>

注意：signalR 依赖于 jquery，所以 signalR 必须放在 jquery 之后，而 hubs 又必须放在 signalR 之后。

然后在 body 部分加入 HubChat Tab：

<li>@Html.ActionLink("HubChat", "HubChat", "Home")</li>

**2），生成访问页面**

在 HomeController 中添加如下方法：

public ActionResult HubChat()

{

ViewBag.ClientName = "用户-" + Rnd.Next(10000, 99999);

return View();

}

这里由服务器根据随机数来设定客户端的名字，不够严谨，因为随机数生成的名字不是唯一的的，在这里仅为简化演示，实际应用中应该使用 GUID 。

然后生成对应的 View：HubChat.cshtml

@model dynamic

@{

ViewBag.Title = "title";

}

<script src="@Url.Content("~/Scripts/hubDemo.js")" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function () {

});

</script>

<h2>Hub Chat</h2>

<div>

<input type="text" id="Placeholder" value="@ViewBag.ClientName" hidden="true"/>

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

<input type="button" id="broadcast" value="广播" />

<br />

<br />

<h3>

消息记录： (你是：<span id="MyClientName">@ViewBag.ClientName</span>):

</h3>

<ul id="messages">

</ul>

</div>

在上面的页面代码中，我添加了名为 hubDemo.js 的脚本，这将在下面介绍；此外还有一个id 为 Placeholder 的隐藏 input 控件，这是为了向 Javascript 中传递客户端的名字。

**3），编写 Javascript**

向 Scripts 目录添加新的 Javescript 脚本：hubDemo.js。其内容如下：

$(function () {

var myClientName = $('#Placeholder').val();

// Proxy created on the fly

var chat = $.connection.chat;

// Declare a function on the chat hub so the server can invoke it

chat.addSomeMessage = function (clientName, message) {

writeEvent('<b>' + clientName + '</b> 对大家说: ' + message, 'event-message');

};

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

// Call the chat method on the server

chat.send(myClientName, $('#msg').val())

.done(function () {

console.log('Sent message success!');

})

.fail(function (e) {

console.warn(e);

});

});

// Start the connection

$.connection.hub.start();

//A function to write events to the page

function writeEvent(eventLog, logClass) {

var now = new Date();

var nowStr = now.getHours() + ':' + now.getMinutes() + ':' + now.getSeconds();

$('#messages').prepend('<li class="' + logClass + '"><b>' + nowStr + '</b> ' + eventLog + '.</li>');

}

});

上面代码有详细的注释，下面再讲讲关键之处：

1，首先获取客户端页面的名字；

2，然后通过 $.connection.chat 建立对应服务器端 Hub 类的代理对象 chat；

3，定义客户端的 Javascript 方法 addSomeMessage ，服务器通过 dynamic 方式调用客户端的该方法以实现推送功能。在这里每当收到服务器推送来的消息，就在客户端页面的 messages 列表表头插入该消息。

4，当点击广播按钮时，客户端通过代理对象调用服务器端的 send 方法以实现向服务器发送消息。

5，通过 $.connection.hub.start(); 语句打开链接。

**6），编译运行 Hub 示例**

在多个浏览器窗口打开页面，效果如下：



**四，Persistent Connection 示例教程**

**1，实现服务器端代码**

**1），编写服务器 PersistentConnection 代码**

向工程中 SignalR 目录中添加 PersistentConnection.cs 文件，内容如下：

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using SignalR;

namespace SignalTutorial.SignalR

{

public class MyConnection : PersistentConnection

{

protected override Task OnConnectedAsync(IRequest request, string connectionId)

{

return Connection.Broadcast("Connection " + connectionId + " connected");

}

protected override Task OnReconnectedAsync(IRequest request, IEnumerable<string> groups, string clientId)

{

return Connection.Broadcast("Client " + clientId + " re-connected");

}

protected override Task OnReceivedAsync(IRequest request, string connectionId, string data)

{

var info = data + ". ConnectionId is [" + connectionId + "]";

// return Connection.Send(connectionId, info);

// Broadcast data to all clients

return Connection.Broadcast(info);

}

protected override Task OnDisconnectAsync(string connectionId)

{

return Connection.Broadcast("Connection " + connectionId + " disconncted");

}

protected override Task OnErrorAsync(Exception error)

{

return Connection.Broadcast("Error ocurred " + error);

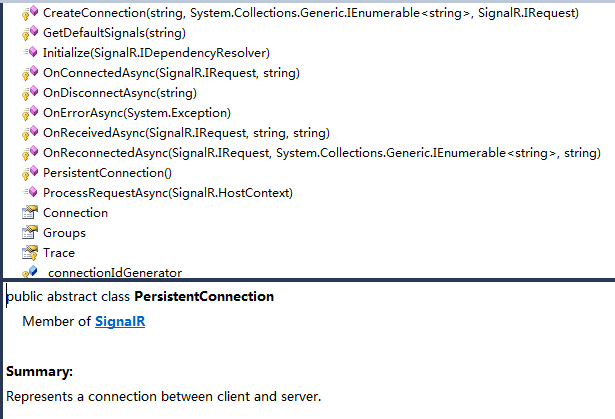
}

}

}

在上面的代码中：

1，MyConnection 继承自 PersistentConnection，这样我们就能在客户端连接，重连接，断开连接，发送消息以及连接出错的情况下进行相关的处理。从下面的 PersistentConnection 接口中可以看到，PersistentConnection 同样支持组进行推送。



2，推送消息由 PersistentConnection 的属性 Connection 来提供，它继承自 IConnection 接口，该接口提供两个函数来实现对特定客户端的推送和广播功能。

System.Threading.Tasks.Task Send(string signal, object value)  
System.Threading.Tasks.Task Broadcast(object value)

**2），配置访问路由**

为了支持客户端访问，需要在路由表中进行配置。打开 Global.asax.cs ，修改 Application\_Start() 函数如下：

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

RouteTable.Routes.MapConnection<MyConnection>("echo", "echo/{\*operation}");

RegisterGlobalFilters(GlobalFilters.Filters);

RegisterRoutes(RouteTable.Routes);

// Make connections wait 50s maximum for any response. After

// 50s are up, trigger a timeout command and make the client reconnect.

GlobalHost.Configuration.ConnectionTimeout = TimeSpan.FromSeconds(50);

//DisconnectTimeout

//HeartBeatInterval

//KeepAlive

}

在上面的代码中，我将 echo 及其子路径的访问映射到 MyConnection 上，并设置连接超时时间为 50 s。在这里还可以设置其他的一些参数，如断连超时时间，心跳间隔等。

**2，实现客户端代码**

**1），生成访问页面**

在前面三 Hub 示例教程的基础上，我们向该工程加入使用 Persistent Connection 的演示。和前面一样，向 \_Layout.cshtml 中加入 PersistentChat Tab：

<li>@Html.ActionLink("PersistentChat", "PersistentChat", "Home")</li>

然后在 HomeController 中添加如下方法：

public ActionResult PersistentChat()

{

ViewBag.ClientName = "用户-" + Rnd.Next(10000, 99999);

return View();

}

这里由服务器根据随机数来设定客户端的名字，不够严谨，因为随机数生成的名字不是唯一的的，在这里仅为简化演示，实际应用中应该使用 GUID 。

然后生成对应的 页面： PersistentChat.cshtml：

@model dynamic

@{

ViewBag.Title = "title";

}

<script src="@Url.Content("~/Scripts/persistent.js")" type="text/javascript"></script>

<h2>Persistent Chat</h2>

<div>

<input type="text" id="Placeholder" value="@ViewBag.ClientName" hidden="true"/>

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

<input type="button" id="broadcast" value="广播" />

<br />

<br />

<h3>

消息记录： (你是：<span id="MyClientName">@ViewBag.ClientName</span>):

</h3>

<ul id="messages">

</ul>

</div>

在上面的页面代码中，我添加了名为 persistent.js 的脚本，这将在下面介绍；此外还有一个id 为 Placeholder 的隐藏 input 控件，这是为了向 Javascript 中传递客户端的名字。

**2），编写 Javascript**

向 Scripts 目录添加新的 Javescript 脚本：persistent.js。其内容如下：

$(function () {

var myClientName = $('#Placeholder').val();

var connection = $.connection('/echo');

connection.received(function (data) {

var msg = new String(data);

var index = msg.indexOf("#");

var clientName = msg.substring(0, index);

var content = msg.substring(index + 1);

if (clientName == null || clientName == "") {

writeEvent('<b>' + "系统消息" + '</b>: ' + content, 'event-message');

}

else {

writeEvent('<b>' + clientName + '</b> 对大家说: ' + content, 'event-message');

}

});

connection.start();

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

var msg = myClientName + "#" + $('#msg').val();

connection.send(msg);

});

//A function to write events to the page

function writeEvent(eventLog, logClass) {

var now = new Date();

var nowStr = now.getHours() + ':' + now.getMinutes() + ':' + now.getSeconds();

$('#messages').prepend('<li class="' + logClass + '"><b>' + nowStr + '</b> ' + eventLog + '.</li>');

}

});

上面的代码基本与前面的 Hub 实现相同，在此就不再一一讲述。有两点值得说明：

1，创建连接时，指定路径为 "/echo"，该路径在服务器端的路由映射表被映射为 MyConnection，因而这个连接就被指向前面提供的 MyConnection。

2，将 clientName 信息放入 message 中，并用 # 将 clientName 和消息内容连接成一个 msg。

**3，编译运行 Persistent 示例**



五，引用

SignalR：

<https://github.com/SignalR/>

利用SignalR實現遠端程式遙控功能：

<http://blog.darkthread.net/post-2012-07-10-signalr-remote-controller.aspx>

一个很酷的同步操作表格的示例（使用 jTable ）：

<http://www.codeproject.com/Articles/315938/Real-time-Asynchronous-Web-Pages-using-jTable-Sign>

组通知示例：

<http://www.codeproject.com/Articles/404662/SignalR-Group-Notifications>