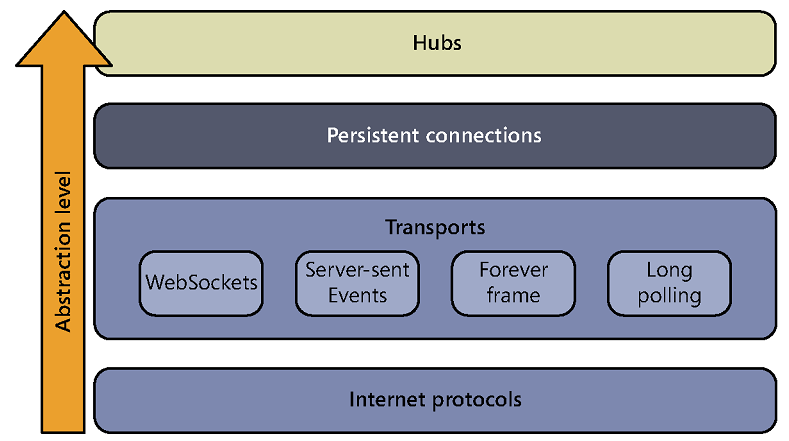
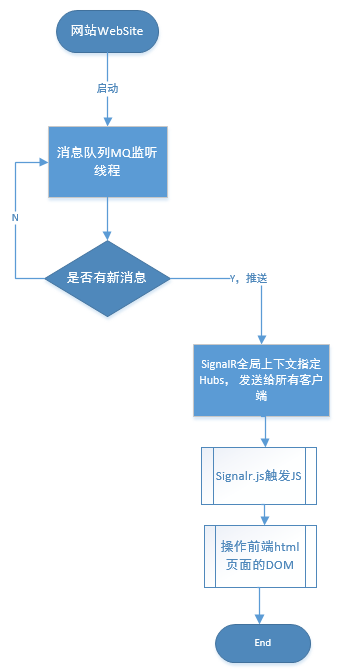
**[Asp.net SignalR 实现服务端消息推送到Web端](http://www.cnblogs.com/wintersun/p/4148223.html)**

          之前的文章介绍过[Asp.net SignalR](http://www.asp.net/signalr" \t "_blank),  ASP .NET SignalR是一个ASP .NET 下的类库，可以在ASP .NET 的Web项目中实现实时通信.  今天我们来实现服务端消息推送到Web端,   首先回顾一下它抽象层次图是这样的:

[](http://images.cnitblog.com/blog/15172/201412/061436257809546.png)

实际上 Asp.net SignalR 2 实现 服务端消息推送到Web端, 更加简单. 为了获取更好的可伸缩性, 我们引入消息队列, 看如下基本流程图:

[](http://images.cnitblog.com/blog/15172/201412/061436278897701.png)

消息队列MQ监听, 在Web site 服务端一收到消息,马上通过Signalr 推送广播到客户端.  创建ASP.NET MVC WEB APP,  从NuGet 安装SignalR 2.12

Install-Package Microsoft.AspNet.SignalR

具体实现代码,是这样的,我们增加一个空的Hub:

public class FeedHub : Hub

{

public void Init()

{

}

}

是简单的消息模型, 标题与正文属性:

[Serializable]

public class PushMessageModel

{

public int Id { get; set; }

public string MSG\_TITLE { get; set; }

public string MSG\_CONTENT { get; set; }

}

服务端推送具体类,记录日志, 创建消息队列实例,监听, 等待收取消息. 这里我们使用的是AcitveMQ的.net客户端. ActiveMQListenAdapter是一个封装过的对象.

public class MQHubsConfig

{

private static ILogger log = new Logger("MQHubsConfig");

/// <summary>

/// Registers the mq listen and hubs.

/// </summary>

public static void RegisterMQListenAndHubs()

{

var activemq = Megadotnet.MessageMQ.Adapter.ActiveMQListenAdapter<PushMessageModel>.Instance(MQConfig.MQIpAddress, MQConfig.QueueDestination);

activemq.MQListener += m =>

{

log.InfoFormat("从MQ收到消息{0}", m.MSG\_CONTENT);

GlobalHost.ConnectionManager.GetHubContext<FeedHub>().Clients.All.receive(m);

};

activemq.ReceviceListener<PushMessageModel>();

}

}

上面有一句关键代码GlobalHost.ConnectionManager.GetHubContext<FeedHub>().Clients.All.receive(m);  这里使用了GetHubContext方法后,直接来广播消息.

需要在MVCApplication下加载:

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

MQHubsConfig.RegisterMQListenAndHubs();

}

}

同时需要增加一个Starup.cs, 用于Owin

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

namespace RealTimeApp

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

// Any connection or hub wire up and configuration should go here

app.MapSignalR();

}

}

}

接下来是客户端App.js:

function App() {

var init = function () {

Feed();

$.connection.hub.logging = true;

$.connection.hub.start()

.done(function() {

console.log("Connected!");

$(document).trigger("Connected");

})

.fail(function() { console.log("Could not Connect!"); });

};

init();

};

Feed.js 具体与SignalR.js通信, 创建名为receive的function, 与服务端对应

function Feed() {

var chat = undefined;

var init = function () {

// Reference the auto-generated proxy for the hub.

chat = $.connection.feedHub;

// Create a function that the hub can call back to display messages.

chat.client.receive = function (item) {

var selector = "ul.feed-list li[data-id=" + item.Id + "]";

if (!($(selector).length > 0)) {

$("ul.feed-list").prepend($(".feed-template").render(item));

$("ul.feed-list li:gt(3)").remove();

}

$.messager.show({

title: 'Tips',

msg: item.MSG\_CONTENT,

showType: 'show'

});

};

// Start the connection.

$.connection.hub.start().done(function () {

chat.server.init();

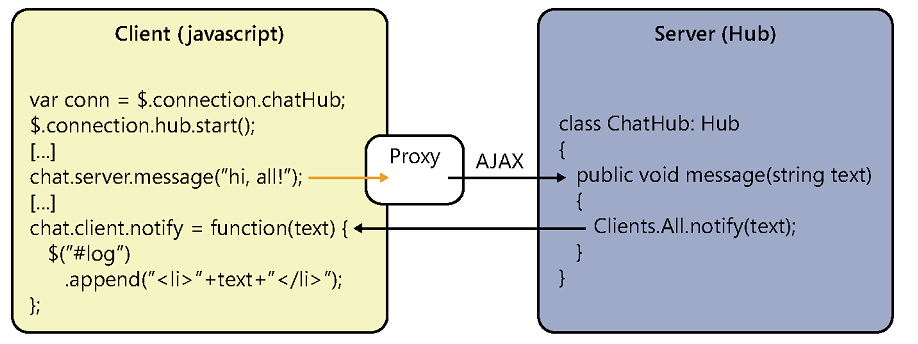
});

};

init();

};

        上面的javascript代码与服务端有通信, 具体看如下图:

[](http://images.cnitblog.com/blog/15172/201412/061436304364257.png)

        在Index.cshtml,  我们需要引用SignalR客户端JS, 放置hubs, 这里我们使用了jsrender,  easyui.js 来呈现推送的消息.

@model dynamic

@section Scripts {

<link href="/Content/themes/default/window.css" rel="stylesheet" />

<link href="~/Content/themes/default/progressbar.css" rel="stylesheet" />

<link href="~/Content/themes/default/linkbutton.css" rel="stylesheet" />

<script src="~/Scripts/jquery.signalR-2.1.2.min.js"></script>

<!--Reference the autogenerated SignalR hub script. -->

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

<script src="~/Scripts/jsrender.js"></script>

<script src="~/Scripts/jquery.easyui.min-1.4.1.js"></script>

@Scripts.Render("~/Scripts/project.js")

<script type="text/javascript">

$(document).ready(function () {

var app = new App();

});

</script>

}

<div class="row-fluid">

<div class="span8">

<div class="widget">

<div class="widget-header">

<h2>Feed</h2>

</div>

<div class="widget-content">

<ul class="span12 feed-list"></ul>

</div>

</div>

</div>

</div>

<script class="chat-template" type="text/x-jquery-tmpl">

<li>

<p>{{>Message}}</p>

</li>

</script>

<script class="feed-template" type="text/x-jquery-tmpl">

<li data-id="{{>Id}}">

<div class="row-fluid">

<div class="span8">

<h3>{{>MSG\_CONTENT}}</h3>

</div>

</div>

</li>

</script>

上代码服务端引用js的Script.Render, 需要在BundleConfig.cs中加入以下代码:

bundles.Add(new ScriptBundle("~/Scripts/project.js")

.IncludeDirectory("~/Scripts/Project", "\*.js", false));

同时我们构建一个WebAPI来发送需要推送的消息, 片断代码:

/// <summary>

/// SendMessage

/// </summary>

/// <param name="messagemodel">The messagemodel.</param>

/// <returns></returns>

[HttpPost]

public IHttpActionResult SendMessage(PushMessageModel messagemodel)

{

return SendToServer(messagemodel);

}

/// <summary>

/// Sends to server.

/// </summary>

/// <param name="messagemodel">The messagemodel.</param>

/// <returns></returns>

private IHttpActionResult SendToServer(PushMessageModel messagemodel)

{

if (ModelState.IsValid)

{

if (messageRepository.SendMessage(messagemodel))

{

log.Debug("发送成功！");

return Ok();

}

else

{

log.ErrorFormat("发送失败！{0}", messagemodel);

return Content(HttpStatusCode.ExpectationFailed, new Exception("send message error"));

}

}

else

{

log.ErrorFormat("参数验证失败！{0}", messagemodel);

return Content(HttpStatusCode.BadRequest, ModelState);

}

}

发送消息到ActiveMQ的关键代码:

public class MessageRepository:IMessageRepository

{

private static ILogger log = new Logger("MessageRepository");

/// <summary>

/// 发送消息

/// </summary>

/// <param name="messagemodel"></param>

/// <returns></returns>

public bool SendMessage(PushMessageModel messagemodel)

{

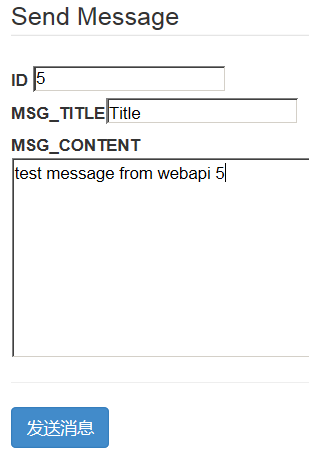
var activemq = new ActiveMQAdapter<PushMessageModel>(MQConfig.MQIpAddress, MQConfig.QueueDestination);

return activemq.SendMessage<PushMessageModel>(messagemodel)>0;

}

}

如果您需要运行DEMO程序,需要构建基于ActiveMQ的消息队列,   运行效果是这样的, 我们在一个静态html中, 发送一个ajax到webapi服务端,  发送后

[](http://images.cnitblog.com/blog/15172/201412/061436342642854.png)

另一个website网站收到后, 列表更新, 并在右下角弹出框

[](http://images.cnitblog.com/blog/15172/201412/061436362016996.png)

IE的控制台输出:

HTML1300: Navigation occurred.   
File: Index   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: Client subscribed to hub 'feedhub'.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: Negotiating with '/signalr/negotiate?clientProtocol=1.4&connectionData=%5B%7B%22name%22%3A%22feedhub%22%7D%5D'.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: This browser doesn't support SSE.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: Binding to iframe's load event.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: Iframe transport started.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: foreverFrame transport selected. Initiating start request.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: The start request succeeded. Transitioning to the connected state.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: Now monitoring keep alive with a warning timeout of 13333.333333333332 and a connection lost timeout of 20000.   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: Invoking feedhub.Init   
Connected!   
[11:05:25 GMT+0800 (China Standard Time)] SignalR: Invoked feedhub.Init   
[11:07:12 GMT+0800 (China Standard Time)] SignalR: Triggering client hub event 'receive' on hub 'FeedHub'.   
[11:07:18 GMT+0800 (China Standard Time)] SignalR: Triggering client hub event 'receive' on hub 'FeedHub'.   
[11:07:32 GMT+0800 (China Standard Time)] SignalR: Triggering client hub event 'receive' on hub 'FeedHub'.   
[11:07:51 GMT+0800 (China Standard Time)] SignalR: Triggering client hub event 'receive' on hub 'FeedHub'.   
**[11:09:25 GMT+0800 (China Standard Time)] SignalR: Triggering client hub event 'receive' on hub 'FeedHub'**.

上面粗体是 最后我们发的第5条信息控制台的输出.