<http://www.cnblogs.com/landeanfen/p/5287064.html>

# [C#进阶系列——WebApi 身份认证解决方案：Basic基础认证](http://www.cnblogs.com/landeanfen/p/5287064.html)

**阅读目录**

* [一、为什么需要身份认证](http://www.cnblogs.com/landeanfen/p/5287064.html#_label0)
* [二、Basic基础认证的原理解析](http://www.cnblogs.com/landeanfen/p/5287064.html#_label1)
  + [1、常见的认证方式](http://www.cnblogs.com/landeanfen/p/5287064.html#_label1_0)
  + [2、Basic基础认证原理](http://www.cnblogs.com/landeanfen/p/5287064.html#_label1_1)
* [三、Basic基础认证的代码示例](http://www.cnblogs.com/landeanfen/p/5287064.html#_label2)
  + [1、登录过程](http://www.cnblogs.com/landeanfen/p/5287064.html#_label2_0)
  + [2、/Home/Index主界面](http://www.cnblogs.com/landeanfen/p/5287064.html#_label2_1)
  + [3、WebApiCORS验证部分（重点）](http://www.cnblogs.com/landeanfen/p/5287064.html#_label2_2)
* [四、优化](http://www.cnblogs.com/landeanfen/p/5287064.html#_label3)
  + [1、解决API的问题](http://www.cnblogs.com/landeanfen/p/5287064.html#_label3_0)
  + [2、解决ajax的问题](http://www.cnblogs.com/landeanfen/p/5287064.html#_label3_1)
  + [3、解决特殊不想使用验证的方法](http://www.cnblogs.com/landeanfen/p/5287064.html#_label3_2)
* [五、总结](http://www.cnblogs.com/landeanfen/p/5287064.html#_label4)

**正文**

前言：最近，讨论到数据库安全的问题，于是就引出了WebApi服务没有加任何验证的问题。也就是说，任何人只要知道了接口的url，都能够模拟http请求去访问我们的服务接口，从而去增删改查数据库，这后果想想都恐怖。经过一番折腾，总算是加上了接口的身份认证，在此记录下，也给需要做身份认证的园友们提供参考。

WebApi系列文章

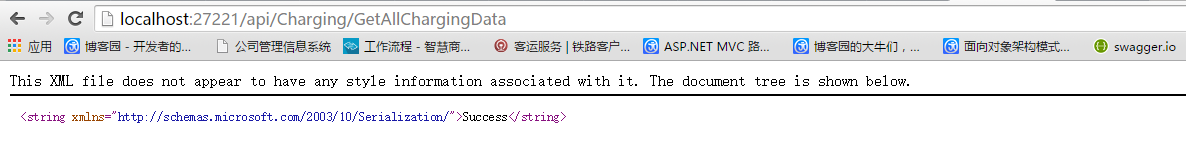
* [C#进阶系列——WebApi接口测试工具：WebApiTestClient](http://www.cnblogs.com/landeanfen/p/5210356.html)
* [C#进阶系列——WebApi 跨域问题解决方案：CORS](http://www.cnblogs.com/landeanfen/p/5177176.html)
* [C#进阶系列——WebApi身份认证解决方案：Basic基础认证](http://www.cnblogs.com/landeanfen/p/5287064.html)
* [C#进阶系列——WebApi接口传参不再困惑：传参详解](http://www.cnblogs.com/landeanfen/p/5337072.html)
* [C#进阶系列——WebApi接口返回值不困惑：返回值类型详解](http://www.cnblogs.com/landeanfen/p/5501487.html)
* C#进阶系列——WebApi异常处理解决方案
* C#进阶系列——WebApi区域Area使用小结

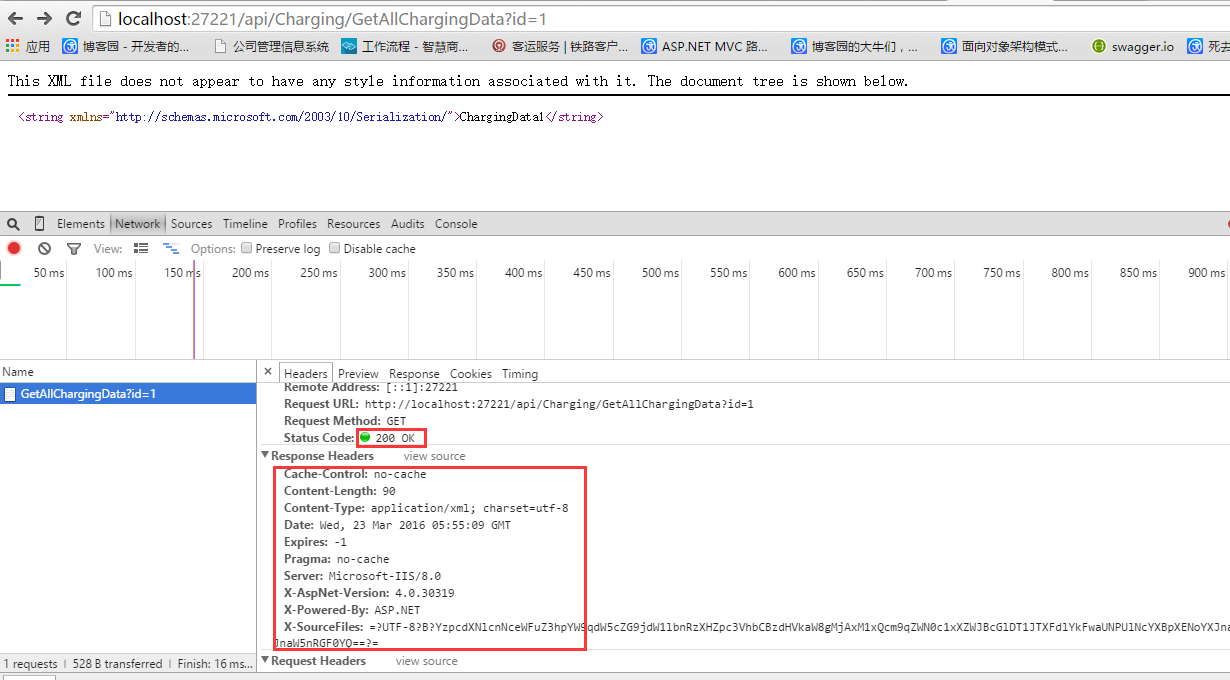
[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

# 一、为什么需要身份认证

在前言里面，我们说了，如果没有启用身份认证，那么任何匿名用户只要知道了我们服务的url，就能随意访问我们的服务接口，从而访问或修改数据库。

### 1、我们不加身份认证，匿名用户可以直接通过url随意访问接口：

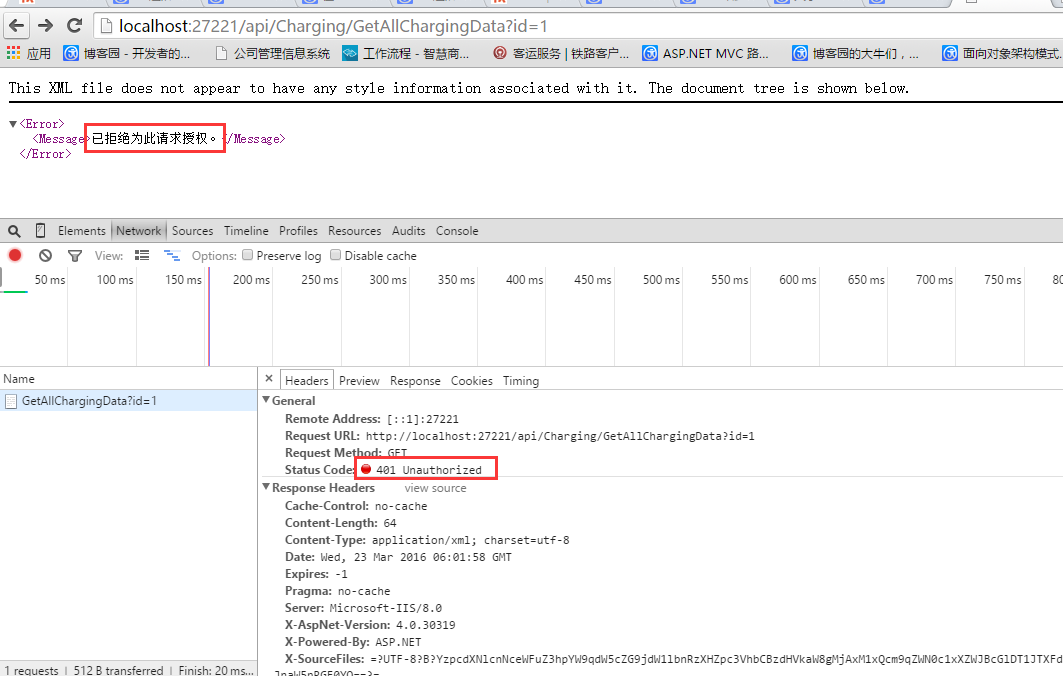




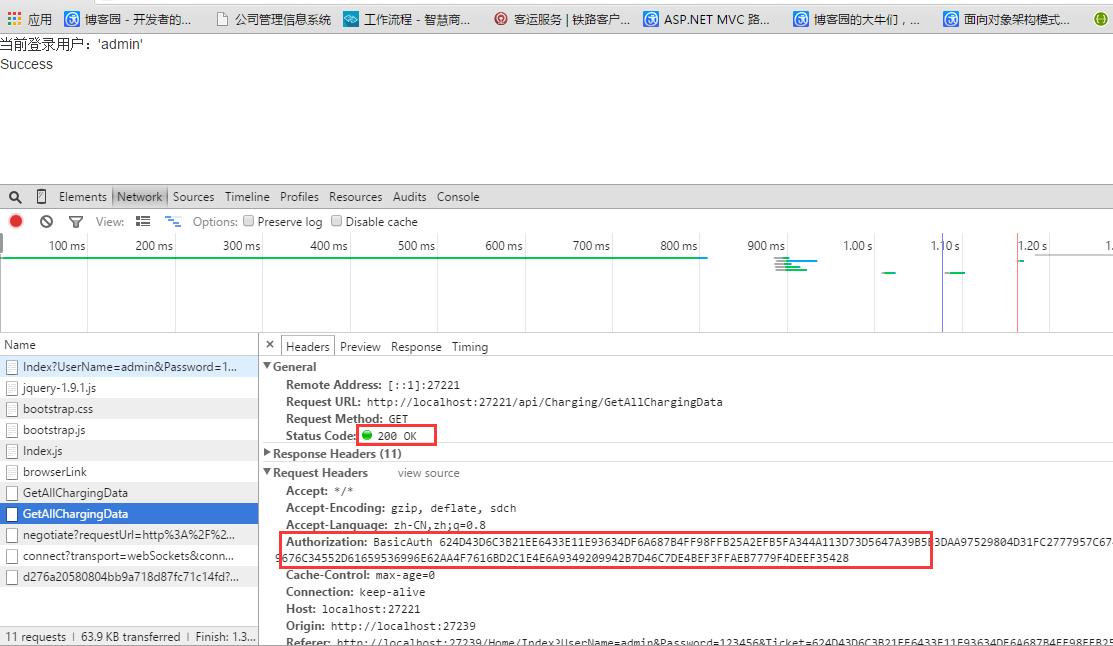
可以看到，匿名用户直接通过url就能访问我们的数据接口，最终会发生什么事，大家可以随意畅想。

### 2、增加了身份认证之后，只有带了我们访问票据的请求才能访问我们的接口。

例如我们直接通过url访问，会返回401



 如果是正常流程的请求，带了票据，就OK了。



可以看到，正常流程的请求，会在请求报文的头里面增加Authorization这一项，它的值就是我们的Ticket票据信息。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

# 二、Basic基础认证的原理解析

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 1、常见的认证方式

我们知道，asp.net的认证机制有很多种。对于WebApi也不例外，常见的认证方式有

* FORM身份验证
* 集成WINDOWS验证
* Basic基础认证
* Digest摘要认证

园子里很多关于WebApi认证的文章，各种认证方式都会涉及到，但感觉都不够细。这里也并不想去研究哪种验证方式适用哪种使用场景，因为博主还是觉得“贪多嚼不烂”，也可能是博主能力所限。对于认证机制，弄懂其中一种，其他的都能融会贯通。此篇就使用Basic基础认证来详细讲解下整个的过程。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 2、Basic基础认证原理

 我们知道，认证的目的在于安全，那么如何能保证安全呢？常用的手段自然是加密。Basic认证也不例外，主要原理就是加密用户信息，生成票据，每次请求的时候将票据带过来验证。这样说可能有点抽象，我们详细分解每个步骤：

1. 首先登陆的时候验证用户名、密码，如果登陆成功，则将用户名、密码按照一定的规则生成加密的票据信息Ticket，将票据信息返回到前端。
2. 如果登陆成功，前端会收到票据信息，然后跳转到主界面，并且将票据信息也带到主界面的ActionResult里面(例如跳转的url可以这样写：/Home/Index?Ticket=Ticket)
3. 在主界面的ActionResult里面通过参数得到票据信息Ticket，然后将Ticket信息保存到ViewBag里面传到前端。
4. 在主界面的前端，发送Ajax请求的时候将票据信息加入到请求的Head里面，将票据信息随着请求一起发送到服务端去。
5. 在WebApi服务里面定义一个类，继承AuthorizeAttribute类，然后重写父类的OnAuthorization方法，在OnAuthorization方法里面取到当前http请求的Head，从Head里面取到我们前端传过来的票据信息。解密票据信息，从解密的信息里面得到用户名和密码，然后验证用户名和密码是否正确。如果正确，表示验证通过，否则返回未验证的请求401。

 这个基本的原理。下面就按照这个原理来看看每一步的代码如何实现。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html" \l "_labelTop)

# 三、Basic基础认证的代码示例

首先说下我们的示例场景，上次介绍 [CORS](http://www.cnblogs.com/landeanfen/p/5177176.html) 的时候我们在一个解决方案里面放了两个项目Web和WebApiCORS，我们这次还是以这个为例来说明。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 1、登录过程

### 1.1、Web前端

[复制代码](javascript:void(0);)

<body>

<div style="text-align:center;">

<div>用户名：<input type="text" id="txt\_username" /></div>

<div>密 码：<input type="password" id="txt\_password" /></div>

<div><input type="button" value="登录" id="btn\_login" class="btn-default" /></div>

</div>

</body>

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

$(function () {

$("#btn\_login").click(function () {

$.ajax({

type: "get",

url: "http://localhost:27221/api/User/Login",

data: { strUser: $("#txt\_username").val(), strPwd: $("#txt\_password").val() },

success: function (data, status) {

if (status == "success") {

if (!data.bRes){

alert("登录失败");

return;

}

alert("登录成功");  
　　　　　　　　　　　 //登录成功之后将用户名和用户票据带到主界面

window.location = "/Home/Index?UserName=" + data.UserName + "&Ticket=" + data.Ticket;

}

},

error: function (e) {

},

complete: function () {

}

});

});

});

[复制代码](javascript:void(0);)

### 1.2、登录的API接口

[复制代码](javascript:void(0);)

　　public class UserController : ApiController

{

/// <summary>

/// 用户登录

/// </summary>

/// <param name="strUser"></param>

/// <param name="strPwd"></param>

/// <returns></returns>

[HttpGet]

public object Login(string strUser, string strPwd)

{

if (!ValidateUser(strUser, strPwd))

{

return new { bRes = false };

}

FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(0, strUser, DateTime.Now,

DateTime.Now.AddHours(1), true, string.Format("{0}&{1}", strUser, strPwd),

FormsAuthentication.FormsCookiePath);

//返回登录结果、用户信息、用户验证票据信息

var oUser = new UserInfo { bRes = true, UserName = strUser, Password = strPwd, Ticket = FormsAuthentication.Encrypt(ticket) };

//将身份信息保存在session中，验证当前请求是否是有效请求

HttpContext.Current.Session[strUser] = oUser;

return oUser;

}

//校验用户名密码（正式环境中应该是数据库校验）

private bool ValidateUser(string strUser, string strPwd)

{

if (strUser == "admin" && strPwd == "123456")

{

return true;

}

else

{

return false;

}

}

}

public class UserInfo

{

public bool bRes { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

public string Ticket { get; set; }

}

[复制代码](javascript:void(0);)

这里有一点需要注意的是，因为WebApi默认是没有开启Session的，所以需要我们作一下配置，手动去启用session。如何开启WebApi里面的Session，请参考：[http://www.cnblogs.com/tinya/p/4563641.html](http://www.cnblogs.com/tinya/p/4563641.html" \t "_blank)

正如上面的原理部分说的，登录如果失败，则直接返回；如果成功，则将生成的票据Ticket带到前端，传到主界面/Home/Index，下面，我们就来看看主界面Home/Index。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 2、/Home/Index主界面

[复制代码](javascript:void(0);)

　　public class HomeController : Controller

{

// GET: Home

public ActionResult Index(string UserName, string Ticket)

{

ViewBag.UserName = UserName;

ViewBag.Ticket = Ticket;

return View();

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Index</title>

<script src="~/Content/jquery-1.9.1.js"></script>

<link href="~/Content/bootstrap/css/bootstrap.css" rel="stylesheet" />

<script src="~/Content/bootstrap/js/bootstrap.js"></script>

<script src="~/Scripts/Home/Index.js"></script>

<script type="text/javascript">

//打开页面的时候保存票据信息

var UserName = '@ViewBag.UserName';

var Ticket = '@ViewBag.Ticket';

</script>

</head>

<body>

<div>当前登录用户：'@ViewBag.UserName'</div>

<div id="div\_test">

</div>

</body>

</html>

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

$(function () {

$.ajax({

type: "get",

url: "http://localhost:27221/api/Charging/GetAllChargingData",

data: {},

beforeSend: function (XHR) {

//发送ajax请求之前向http的head里面加入验证信息

XHR.setRequestHeader('Authorization', 'BasicAuth ' + Ticket);

},

success: function (data, status) {

if (status == "success") {

$("#div\_test").html(data);

}

},

error: function (e) {

$("#div\_test").html("Error");

},

complete: function () {

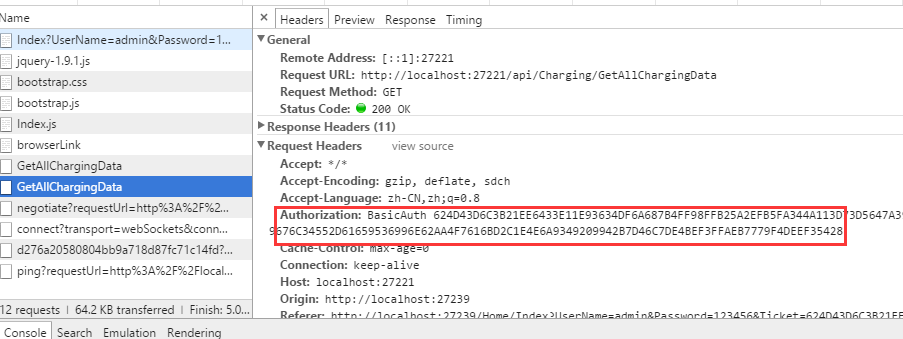
}

});

});

[复制代码](javascript:void(0);)

这里需要说明的是，我们在发送ajax请求之前，通过 XHR.setRequestHeader('Authorization', 'BasicAuth ' + Ticket); 这一句向请求的报文头里面增加票据信息。就是因为这里加了这一句，所以才有我们下图中的红线部分：



[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 3、WebApiCORS验证部分（重点）

我们看到，上面的/Home/Index页面里面发送了ajax请求去访问服务的 http://localhost:27221/api/Charging/GetAllChargingData 这个接口，那么我们在WebApi里面怎么去验证这个请求和合法的请求呢？接下来我们重点看看验证的这个过程。

### 3.1、在WebApiCORS项目里面自定义一个类RequestAuthorizeAttribute，去继承我们的AuthorizeAttribute这个类。然后重写OnAuthorization方法，在这个方法里面取到请求头的Ticket信息，然后校验用户名密码是否合理。

[复制代码](javascript:void(0);)

　　 /// <summary>

/// 自定义此特性用于接口的身份验证

/// </summary>

public class RequestAuthorizeAttribute : AuthorizeAttribute

{

//重写基类的验证方式，加入我们自定义的Ticket验证

public override void OnAuthorization(System.Web.Http.Controllers.HttpActionContext actionContext)

{

//从http请求的头里面获取身份验证信息，验证是否是请求发起方的ticket

var authorization = actionContext.Request.Headers.Authorization;

if ((authorization != null) && (authorization.Parameter != null))

{

//解密用户ticket,并校验用户名密码是否匹配

var encryptTicket = authorization.Parameter;

if (ValidateTicket(encryptTicket))

{

base.IsAuthorized(actionContext);

}

else

{

HandleUnauthorizedRequest(actionContext);

}

}

//如果取不到身份验证信息，并且不允许匿名访问，则返回未验证401

else

{

var attributes = actionContext.ActionDescriptor.GetCustomAttributes<AllowAnonymousAttribute>().OfType<AllowAnonymousAttribute>();

bool isAnonymous = attributes.Any(a => a is AllowAnonymousAttribute);

if (isAnonymous) base.OnAuthorization(actionContext);

else HandleUnauthorizedRequest(actionContext);

}

}

//校验用户名密码（正式环境中应该是数据库校验）

private bool ValidateTicket(string encryptTicket)

{

//解密Ticket

var strTicket = FormsAuthentication.Decrypt(encryptTicket).UserData;

//从Ticket里面获取用户名和密码

var index = strTicket.IndexOf("&");

string strUser = strTicket.Substring(0, index);

string strPwd = strTicket.Substring(index + 1);

if (strUser == "admin" && strPwd == "123456")

{

return true;

}

else

{

return false;

}

}

}

[复制代码](javascript:void(0);)

### 3.2、在具体的Api接口增加我们上面自定义类的特性

[复制代码](javascript:void(0);)

[RequestAuthorize]

public class ChargingController : ApiController

{

/// <summary>

/// 得到所有数据

/// </summary>

/// <returns>返回数据</returns>

[HttpGet]

public string GetAllChargingData()

{

return "Success";

}

/// <summary>

/// 得到当前Id的所有数据

/// </summary>

/// <param name="id">参数Id</param>

/// <returns>返回数据</returns>

[HttpGet]

public string GetAllChargingData(string id)

{

return "ChargingData" + id;

}

}

[复制代码](javascript:void(0);)

增加了特性标注之后，每次请求这个API里面的接口之前，程序会先进入到我们override过的 OnAuthorization() 方法里面，验证通过之后，才会进到相应的方法里面去执行，否则返回401。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

# 四、优化

 通过上面的几步，基本就能达到我们想要的身份认证的效果，但是总是感觉不太方便，主要不太方便的点有以下几个。

1. 每次新建一个API，对应的接口上面都要标注 [RequestAuthorize] 这个一个东西，感觉好麻烦。
2. 每次发送ajax请求，都要在beforeSend事件里面加 XHR.setRequestHeader('Authorization', 'BasicAuth ' + Ticket); 这个，感觉也麻烦。
3. 如果有些WebApi服务的某些方法，我们不想使用这个验证，让它可以匿名用户验证（比如我们的登录方法Login）。该怎么处理呢。

关于以上两点，我们优化下

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 1、解决API的问题

在API里面加一个公共的父类，在父类上面标注 [RequestAuthorize] 即可。

[复制代码](javascript:void(0);)

namespace WebApiCORS.Controllers

{

[RequestAuthorize]

[EnableCors(origins: "\*", headers: "\*", methods: "\*")]

public class BaseApiController : ApiController

{

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

namespace WebApiCORS.Controllers

{

public class ChargingController : BaseApiController

{

/// <summary>

/// 得到所有数据

/// </summary>

/// <returns>返回数据</returns>

[HttpGet]

public string GetAllChargingData()

{

return "Success";

}

/// <summary>

/// 得到当前Id的所有数据

/// </summary>

/// <param name="id">参数Id</param>

/// <returns>返回数据</returns>

[HttpGet]

public string GetAllChargingData(string id)

{

return "ChargingData" + id;

}

　　}

}

[复制代码](javascript:void(0);)

 注意：我们登录的请求是不需要验证的，因为登录的时候还没有产生票据，所以登录的API不能够继承 BaseApiController

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 2、解决ajax的问题

还记得我们在 [JS组件系列——封装自己的JS组件，你也可以](http://www.cnblogs.com/landeanfen/p/5124542.html) 这篇里面介绍的增加ajax的error事件的公共处理方法吗？我们是否也可以通过同样的机制去增加这个呢。新建一个文件Jquery\_ajax\_extention.js

[复制代码](javascript:void(0);)

(function ($) {

//1.得到$.ajax的对象

var \_ajax = $.ajax;

$.ajax = function (options) {

//2.每次调用发送ajax请求的时候定义默认的error处理方法

var fn = {

error: function (XMLHttpRequest, textStatus, errorThrown) {

toastr.error(XMLHttpRequest.responseText, '错误消息', { closeButton: true, timeOut: 0, positionClass: 'toast-top-full-width' });

},

success: function (data, textStatus) { },

beforeSend: function (XHR) { },

complete: function (XHR, TS) { }

}

//3.扩展原生的$.ajax方法，返回最新的参数

var \_options = $.extend({}, {

error: function (XMLHttpRequest, textStatus, errorThrown) {

fn.error(XMLHttpRequest, textStatus, errorThrown);

},

success: function (data, textStatus) {

fn.success(data, textStatus);

},

beforeSend: function (XHR) {

XHR.setRequestHeader('Authorization', 'BasicAuth ' + Ticket);

fn.beforeSend(XHR);

},

complete: function (XHR, TS) {

fn.complete(XHR, TS);

}

}, options);

//4.将最新的参数传回ajax对象

\_ajax(\_options);

};

})(jQuery);

[复制代码](javascript:void(0);)

引用这个js后再发送ajax不必在每个请求的beforeSend里面写了。

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

## 3、解决特殊不想使用验证的方法

如果我们某些方法不想使用验证，使得它可以让匿名用户访问，我们可以在方法的上面加特性标注 [AllowAnonymous] ，申明该方法运行匿名访问。比如：

[复制代码](javascript:void(0);)

　　public class ChargingController : BaseApiController

{

/// <summary>

/// 得到所有数据

/// </summary>

/// <returns>返回数据</returns>

[HttpGet]

public string GetAllChargingData()

{

return "Success";

}

/// <summary>

/// 得到当前Id的所有数据

/// </summary>

/// <param name="id">参数Id</param>

/// <returns>返回数据</returns>

[HttpGet]

[AllowAnonymous]

public string GetAllChargingData(string id)

{

return "ChargingData" + id;

}

　　}

[复制代码](javascript:void(0);)

[回到顶部](http://www.cnblogs.com/landeanfen/p/5287064.html#_labelTop)

# 五、总结

以上结合一个实例讲解了下Basic认证的实现原理以及简单使用，本文观点都是来自博主自己的理解，如果有不全面的地方，还望园友们斧正。如果本文能够或多或少帮到你，不妨帮忙**推荐**，博主一定继续努力~~