本章节为了方便读者的理解，相关例子将使用HttpApiClient静态类来创建http接口的代理类，但在生产环境中，使用HttpApiFactory静态来创建http接口的代理类更合理，也是非常有必要的。

1. GET/HEAD请求

1.1 Get请求简单例子

```
public interface IMyWebApi : IHttpApi
{
    // GET http://www.mywebapi.com/webapi/user?account=laojiu
    [HttpGet("http://www.mywebapi.com/webapi/user")]
    ITask<HttpResponseMessage> GetUserByAccountAsync(string account);
}

var client = HttpApiClient.Create<IMyWebApi>();
var response = await client.GetUserByAccountAsync("laojiu");
```

1.2 使用[HttpHost]特性

```
[HttpHost("http://www.mywebapi.com/")]
public interface IMyWebApi : IHttpApi
{
    // GET /webapi/user?account=laojiu
    [HttpGet("webapi/user")]
    ITask<HttpResponseMessage> GetUserByAccountAsync(string account);
}
```

如果接口IMyWebApi有多个方法且都指向同一服务器，可以将请求的域名抽出来放到HttpHost特性。

1.3 响应的json/xml内容转换为强类型模型

1.3.1 隐式转换为强类型模型

```
[HttpHost("http://www.mywebapi.com/")]
public interface IMyWebApi : IHttpApi
{
    // GET /webapi/user?account=laojiu
    [HttpGet("webapi/user")]
    ITask<UserInfo> GetUserByAccountAsync(string account);
}
```

当方法的返回数据是UserInfo类型的json或xml文本，且响应的Content-Type为application/json或application/xml值时，方法的原有返回类型ITask(Of HttpResponseMessage)就可以声明为ITask(Of UserInfo)。

1.3.2 显式转换为强类型模型

```
[HttpHost("http://www.mywebapi.com/")]
public interface IMyWebApi : IHttpApi
{
    // GET /webapi/user?account=laojiu
```

```
    [HttpGet("webapi/user")]
    [JsonReturn] // 指明使用Json处理返回值为UserInfo类型
    ITask<UserInfo> GetUserByAccountAsync(string account);
}
```

当方法的返回数据是UserInfo类型的json或xml文本，但响应的Content-Type可能不是期望的application/json或application/xml值时，就需要显式声明[JsonReturn]或[XmlReturn]特性。

1.4 请求取消令牌参数CancellationToken

```
[HttpHost("http://www.mywebapi.com/")]
public interface IMyWebApi : IHttpApi
{
    // GET /webapi/user?account=laojiu
    [HttpGet("webapi/user")]
    ITask<UserInfo> GetUserByAccountAsync(string account, CancellationToken token);
}
```

CancellationToken.None表示永不取消；创建一个CancellationTokenSource，可以提供一个CancellationToken。

2.请求URL

2.1 URL的格式

无论是GET还是POST等哪种http请求方法，都遵循如下的URL格式：

{Scheme}://{UserName}:{Password}@{Host}:{Port}{Path}{Query}{Fragment}

例如：http://account:password@www.baidu.com/path1/?p1=abc#tag

2.2 动态PATH

```
public interface IMyWebApi : IHttpApi
{
    // GET http://www.webapiclient.com/laojiu
    [HttpGet("http://www.webapiclient.com/{account}"]
    ITask<string> GetUserByAccountAsync(string account);
}
```

某些接口方法将路径的一个分段语意化，比如GET http://www.webapiclient.com/{account}，这里不同的{account}代表不同账号下的个人信息，使用{参数名}声明路径，在请求前会自动从参数（或参数模型的同名属性）取值替换。

2.3 动态URL

```
public interface IMyWebApi : IHttpApi
{
    // GET {URL}
    [HttpGet]
    ITask<string> GetUserByAccountAsync([Url] string url);
```

```
    // GET {URL}?account=laojiu
    [HttpGet]
    ITask<string> GetUserByAccountAsync([Url] string url, string account);
}
```

如果请求URL在运行时才确定，可以将请求URL作为一个参数，使用[Url]特性修饰这个参数并作为第一个参数。注意：v0.2.4版本开始推荐使用UriAttribute

2.4 Query参数

2.4.1 多个query参数平铺

```
// GET /webapi/user?account=laojiu&password=123456
[HttpGet("webapi/user")]
ITask<UserInfo> GetUserAsync(string account, string password);
```

2.4.2 多个query参数合并到模型

```
public class LoginInfo
{
    public string Account { get; set; }
    public string Password { get; set; }
}

// GET /webapi/user?account=laojiu&password=123456
[HttpGet("webapi/user")]
ITask<UserInfo> GetUserAsync(LoginInfo loginInfo);
```

2.4.3 多个query参数平铺+部分合并到模型

```
public class LoginInfo
{
    public string Account { get; set; }
    public string Password { get; set; }
}

// GET /webapi/user?account=laojiu&password=123456&role=admin
[HttpGet("webapi/user")]
ITask<UserInfo> GetUserAsync(LoginInfo loginInfo, string role);
```

2.4.4 显式声明[PathQuery]特性

```
// GET /webapi/user?account=laojiu&password=123456&role=admin
[HttpGet("webapi/user")]
ITask<UserInfo> GetUserAsync(
    [PathQuery]LoginInfo loginInfo,
    [PathQuery]string role);
```

对于没有任何特性修饰的每个参数，都默认被[PathQuery]修饰，表示做为请求路径或请求参数处理，[PathQuery]特性可以设置Encoding、IgnoreWhenNull和DateTimeFormat多个属性。

3.POST/PUT/DELETE请求

3.1 使用Json或Xml提交

- 使用[XmlContent]修饰强类型模型参数，表示提交xml

- 使用[JsonContent]修饰强类型模型参数，表示提交json

```
// POST webapi/user
// Body user的json文本
[HttpPost("webapi/user")]
ITask<UserInfo> AddUserWithJsonAsync([JsonContent] UserInfo user);

// PUT webapi/user
// Body user的xml文本
[HttpPut("/webapi/user")]
ITask<UserInfo> UpdateUserWithXmlAsync([XmlContent] UserInfo user);
```

3.2 使用x-www-form-urlencoded提交

- 使用[FormContent]修饰强类型模型参数
- 使用[FormField]修饰简单类型参数

```
// POST webapi/user
// Body Account=laojiu&Password=123456
[HttpPost("webapi/user")]
ITask<UserInfo> UpdateUserWithFormAsync(
    [FormContent] UserInfo user);

// POST webapi/user
// Body Account=laojiu&Password=123456&fieldX=xxx
[HttpPost("webapi/user")]
ITask<UserInfo> UpdateUserWithFormAsync(
    [FormContent] UserInfo user,
    [FormField] string fieldX);
```

3.3 使用multipart/form-data提交

- 使用[MulitpartContent]修饰强类型模型参数
- 使用[MulitpartText]修饰简单类型参数
- 使用MulitpartFile类型作为提交的文件

```
// POST webapi/user
[HttpPost("webapi/user")]
ITask<UserInfo> UpdateUserWithMulitpartAsync(
    [MulitpartContent] UserInfo user);

// POST webapi/user
[HttpPost("/webapi/user")]
ITask<UserInfo> UpdateUserWithMulitpartAsync(
    [MulitpartContent] UserInfo user,
    [MulitpartText] string nickName,
    MulitpartFile file);
```

3.4 使用具体的HttpContent类型提交

```
// POST webapi/user
// Body Account=laojiu&Password=123456
[HttpPost("webapi/user")]
```

```
ITask<UserInfo> UpdateUserWithFormAsync(
    FormUrlEncodedContent user);

// POST webapi/user
// Body Account=laoju&Password=123456&age=18
[HttpPost("webapi/user")]
ITask<UserInfo> UpdateUserWithFormAsync(
    FormUrlEncodedContent user,
    [FormField] int age);
```

如果参数是类型是HttpContent类型的子类，如StringContent、ByteArrayContent、StreamContent、FormUrlEncodedContent等等，则可以直接做为参数，但是必须放在其它参数的前面：

4 PATCH请求

json patch是为客户端能够局部更新服务端已存在的资源而设计的一种标准交互，在RFC6902里有详细的介绍json patch，通俗来讲有以下几个要点：

1. 使用HTTP PATCH请求方法；
2. 请求body为描述多个opration的数据json内容；
3. 请求的Content-Type为application/json-patch+json；

4.1 WebApiClient例子

```
public interface IMyWebApi : IHttpApi
{
    [HttpPatch("webapi/user")]
    Task<UserInfo> PatchAsync(string id, JsonPatchDocument<UserInfo> doc);
}


var doc = new JsonPatchDocument<UserInfo>();
doc.Replace(item => item.Account, "laojiu");
doc.Replace(item => item.Email, "laojiu@qq.com");
var client = HttpApiClient.Create<IMyWebApi>();
await client.PatchAsync("id001", doc);
```

4.2 Asp.net 服务端例子

```
[HttpPatch]
public async Task<UserInfo> Patch(string id, [FromBody] JsonPatchDocument<UserInfo>
doc)
{
    // 此处user是从db查询获得
    var user = await GetUserInfoFromDbAsync(id);
    doc.ApplyTo(user);
    return user;
}
```

5 参数及属性注解

这些注解特性的命名空间在WebApiClient.DataAnnotations，用于影响参数的序列化行为。

5.1 参数别名

```
public interface IMyWebApi : IHttpApi
{
    // GET http://www.mywebapi.com/webapi/user?_name=laoji
    [HttpGet("http://www.mywebapi.com/webapi/user")]
    ITask<string> GetUserByAccountAsync(
        [AliasAs("_name")] string account);
}
```

5.2 参数模型属性注解

```
public class UserInfo
{
    public string Account { get; set; }

    // 别名
    [AliasAs("a_password")]
    public string Password { get; set; }

    // 时间格式，优先级最高
    [DateTimeFormat("yyyy-MM-dd")]
    [IgnoreWhenNull] // 值为null则忽略序列化
    public DateTime? BirthDay { get; set; }

    // 忽略序列化
    [IgnoreSerialized]
    public string Email { get; set; }

    // 时间格式
    [DateTimeFormat("yyyy-MM-dd HH:mm:ss")]
    public DateTime CreateTime { get; set; }
}
```

6 参数及参数属性输入验证

这些验证特性都有相同的基类ValidationAttribute，命名空间为System.ComponentModel.DataAnnotations，由netfx或corefx提供。

6.1 参数值的验证

```
[HttpGet("webapi/user/GetById/{id}")]
ITask<HttpResponseMessage> GetByIdAsync(
    [Required, StringLength(10)] string id);
```

id的参数要求必填且最大长度为10的字符串，否则抛出ValidationException的异常。

6.2 参数的属性值验证

```
public class UserInfo
{
    [Required]
    [StringLength(10, MinimumLength = 1)]
    public string Account { get; set; }

    [Required]
    [StringLength(10, MinimumLength = 6)]
    public string Password { get; set; }
}

[HttpPut("webapi/user/UpdateWithJson")]
ITask<UserInfo> UpdateWithJsonAsync(
    [JsonContent] UserInfo user);
```

当user参数不为null的情况，就会验证它的Account和Password两个属性，HttpApiConfig有个UseParameterPropertyValidate属性，设置为false就禁用验证参数的属性值。

6.3 两者同时验证

对于上节的例子，如果我们希望user参数值也不能为null，可以如下声明方法：

```
[HttpPut("webapi/user/UpdateWithJson")]
ITask<UserInfo> UpdateWithJsonAsync(
    [Required, JsonContent] UserInfo user);
```

7 特性的范围和优先级

7.1 特性的范围

有些特性比如[Header]，可以修饰于接口、方法和参数，使用不同的构造器和修饰于不同的地方产生的含义和结果是有点差别的：

- 修饰接口时，表示接口下的所有方法在请求前都会添加这个请求头；
- 修饰方法时，表示此方法在请求前添加这个请求头；
- 修饰参数时，表示参数的值将做为请求头的值，由调用者动态传入；

7.2 特性的优先级

- 方法级比接口级优先级高；
- AllowMultiple为true时，方法级和接口级都生效；
- AllowMultiple为false时，方法级的生效，接口级的无效；

8 完整接口声明示例

> 本示例的接口为swagger官方的v2/swagger.json，参见swagger官网接口，对于swagger文档，可以使用WebApiClient.Tools.Swagger工具生成客户端代码。

8.1 IPetApi

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel.DataAnnotations;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using WebApiClient;
using WebApiClient.Attributes;
using WebApiClient.DataAnnotations;
using WebApiClient.Parameterables;
namespace petstore.swagger
{
    /// <summary>
    /// Everything about your Pets
    /// </summary>
    [TraceFilter]
    [HttpHost("https://petstore.swagger.io/v2/")]
    public interface IPetApi : IHttpApi
    {
        /// <summary>
        /// Add a new pet to the store
        /// </summary>
        /// <param name="body">Pet object that needs to be added to the store</param>
        [HttpPost("pet")]
        ITask<HttpResponseMessage> AddPetAsync([Required] [JsonContent] Pet body);

        /// <summary>
        /// Update an existing pet
        /// </summary>
        /// <param name="body">Pet object that needs to be added to the store</param>
        [HttpPut("pet")]
        ITask<HttpResponseMessage> UpdatePetAsync([Required] [JsonContent] Pet body);

        /// <summary>
        /// Finds Pets by status
        /// </summary>
        /// <param name="status">Status values that need to be considered for
filter</param>
        /// <returns>successful operation</returns>
        [HttpGet("pet/findByStatus")]
        ITask<List<Pet>> FindPetsByStatusAsync([Required] IEnumerable<Anonymous>
status);

        /// <summary>
        /// Finds Pets by tags
        /// </summary>
        /// <param name="tags">Tags to filter by</param>
        /// <returns>successful operation</returns>
        [Obsolete]
```

```
        [HttpGet("pet/findByTags")]
        ITask<List<Pet>> FindPetsByTagsAsync([Required] IEnumerable<string> tags);

        /// <summary>
        /// Find pet by ID
        /// </summary>
        /// <param name="petId">ID of pet to return</param>
        /// <returns>successful operation</returns>
        [HttpGet("pet/{petId}")]
        ITask<Pet> GetPetByIdAsync([Required] long petId);

        /// <summary>
        /// Updates a pet in the store with form data
        /// </summary>
        /// <param name="petId">ID of pet that needs to be updated</param>
        /// <param name="name">Updated name of the pet</param>
        /// <param name="status">Updated status of the pet</param>
        [HttpPost("pet/{petId}")]
        ITask<HttpResponseMessage> UpdatePetWithFormAsync([Required] long petId,
[FormContent] string name, [FormContent] string status);

        /// <summary>
        /// Deletes a pet
        /// </summary>
        /// <param name="api_key"></param>
        /// <param name="petId">Pet id to delete</param>
        [HttpDelete("pet/{petId}")]
        ITask<HttpResponseMessage> DeletePetAsync([Header("api_key")] string api_key,
[Required] long petId);

        /// <summary>
        /// uploads an image
        /// </summary>
        /// <param name="petId">ID of pet to update</param>
        /// <param name="additionalMetadata">Additional data to pass to server</param>
        /// <param name="file">file to upload</param>
        /// <returns>successful operation</returns>
        [TraceFilter(Enable = false)]
        [HttpPost("pet/{petId}/uploadImage")]
        ITask<ApiResponse> UploadFileAsync([Required] long petId, [MulitpartContent]
string additionalMetadata, MulitpartFile file);

    }
}
```

8.2 IUserApi

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using WebApiClient;
using WebApiClient.Attributes;
using WebApiClient.DataAnnotations;
using WebApiClient.Parameterables;
namespace petstore.swagger
{
    /// <summary>
    /// Operations about user
    /// </summary>
    [TraceFilter]
    [HttpHost("https://petstore.swagger.io/v2/")]
    public interface IUserApi : IHttpApi
    {
        /// <summary>
        /// Create user
        /// </summary>
        /// <param name="body">Created user object</param>
        /// <returns>successful operation</returns>
        [HttpPost("user")]
        ITask<HttpResponseMessage> CreateUserAsync([Required] [JsonContent] User
body);

        /// <summary>
        /// Creates list of users with given input array
        /// </summary>
        /// <param name="body">List of user object</param>
        /// <returns>successful operation</returns>
        [HttpPost("user/createWithArray")]
        ITask<HttpResponseMessage> CreateUsersWithArrayInputAsync([Required]
[JsonContent] IEnumerable<User> body);

        /// <summary>
        /// Creates list of users with given input array
        /// </summary>
        /// <param name="body">List of user object</param>
        /// <returns>successful operation</returns>
        [HttpPost("user/createWithList")]
        ITask<HttpResponseMessage> CreateUsersWithListInputAsync([Required]
[JsonContent] IEnumerable<User> body);

        /// <summary>
```

```
        /// Logs user into the system
        /// </summary>
        /// <param name="username">The user name for login</param>
        /// <param name="password">The password for login in clear text</param>
        /// <returns>successful operation</returns>
        [HttpGet("user/login")]
        ITask<string> LoginUserAsync([Required] string username, [Required] string
password);

        /// <summary>
        /// Logs out current logged in user session
        /// </summary>
        /// <returns>successful operation</returns>
        [HttpGet("user/logout")]
        ITask<HttpResponseMessage> LogoutUserAsync();

        /// <summary>
        /// Get user by user name
        /// </summary>
        /// <param name="username">The name that needs to be fetched. Use user1 for
testing.</param>
        /// <returns>successful operation</returns>
        [HttpGet("user/{username}")]
        ITask<User> GetUserByNameAsync([Required] string username);

        /// <summary>
        /// Updated user
        /// </summary>
        /// <param name="username">name that need to be updated</param>
        /// <param name="body">Updated user object</param>
        [HttpPut("user/{username}")]
        ITask<HttpResponseMessage> UpdateUserAsync([Required] string username,
[Required] [JsonContent] User body);

        /// <summary>
        /// Delete user
        /// </summary>
        /// <param name="username">The name that needs to be deleted</param>
        [HttpDelete("user/{username}")]
        ITask<HttpResponseMessage> DeleteUserAsync([Required] string username);

    }
}
```

Pages

- Home
- WebApiClient QA
- WebApiClient基础

- WebApiClient进阶
- WebApiClient高级