Дмитрий Нестерук

Блог о программировании — C#, F#, C++, архитектура, и многое другое

Создание сервиса WCF REST с поддержкой JSONP

Мне всегда импонировали фреймворки и языки, которые делались для того, чтобы сделать “простые вещи простыми, а сложные вещи возможными”. Надеясь именно на подобный расклад вещей, я решил посмотреть на то, как нынче делаются REST сервисы в WCF без помощи каких-либо библиотек (OpenRasta, MindTouch DReAM) или шаблонов вроде WCF REST Starter Kit. В этом посте – мои заметки насчет того, как все вышло.

Мотивация

Почему REST? Все очень просто – мне нынче импонируют идеи сайтов, которые находятся полностью на клиенте и работают с серверами именно через REST а не через `WS-*` или какие-то кастомные RPC-байндинги. REST – это тривиальная парадигма и, казалось бы, любой более менее продвинутый фреймворк должен позволять быстро и эффективно создавать REST-сервисы.

Поэтому я решил испробовать простенький сценарий – взять мой диалоговый фреймворк написанный на WebSharper и перетащить все данные на сервер, осуществив взаимодействие через REST.

Сущности

Первое что я обычно делаю в проекте который хранит данные – это конечно `Install-Package norm` дабы добавить драйвер NoRM для MongoDB. Коллега Суворов конечно рекоммендует якобы официальный драйвер от 10gen, но у меня итак все работает, в т.ч. Linq, поэтому зачем напрягаться?

Как вы помните, NoRM немного замусоривает наш объект, но он все еще остается “почти POCO”. WCF же в долгу не остается, и “домусоривает” объект еще больше, прописывая свои аттрибуты. В результате получаем классы подобные этому:

```
[DataContract]
public class ConversationItem
{
  public ConversationItem()
  {
    Id = ObjectId.NewObjectId();
  }
  [DataMember, MongoIdentifier] public string Id { get; set; }
  [DataMember] public string PartyId { get; set; }
  [DataMember] public string Speech { get; set; }
  [DataMember] public List<string> EnableList { get; set; }
  [DataMember] public List<string> DisableList { get; set; }
}
```

Начинаем писать сервис

Первое что нужно сделать – удалить к черту сгенерировнный интерфейс – сервис проживет и без него, а `[ServiceContract]` можно навесить прямо на класс сервиса. Далее, можно создавать методы, но на них тоже нужно навесить аттрибуты, в частности:

- `OperationContract` для того чтобы пометить что это часть сервиса.
- `WebGet` дабы прописать шаблон вызова а также форматы запроса и ответа.
- `JSONPBehavior`, но он из коробки не поставляется, и мы о нем поговорим попозже.

Вот пример декорированного метода:

```
[OperationContract]
[WebGet(UriTemplate = "/c/{id}", RequestFormat = WebMessageFormat.Json,
[JSONPBehavior(callback = "callback")]
public Conversation GetInitialConversationItems(string id)
{
  ...
}
```

Итак, “из коробки” мы получаем достаточно простой метод, но не хватает двух вещей – обработки ошибок и поддержки JSONP, без которой вы ничего кроссдоменно не вызовете.

Обработка ошибок

Для того чтобы возвращать всякие статус коды вроде Not Found, нужно перехватить выходящий ответ из `WebOperationContext` и прописать в него информацию о том, что собственно пошло не так. Например:

```
using (var db = GetDB())
{
```

```
3     var conversation = db.Query<Conversation>().FirstOrDefault();
4     if (conversation == null)
5     {
6       var resp = WebOperationContext.Current.OutgoingResponse;
7       resp.StatusCode = HttpStatusCode.NotFound;
8       resp.StatusDescription = "Could not find conversation with id='{0}
9       return null;
10    }
11    else
12    {
13      return conversation;
14    }
15  }
```

JSONP

Феноменально, но факт – WCF не поставляется с поддержкой JSONP. К счастью, Microsoft дает такую поддержку в примерах, и то как она выглядит является хорошей демонстрацией того, как гибок WCF в плане расширения.

Всего для поддержки JSONP нужно добавить 5 классов. Детально я описывать их не буду, опишу только вкратце.

1. Во-первых, нужна реализация аттрибута `JSONPBehavior` который я уже описывал. Фактически, этот аттрибут навешивает на операцию объект типа `IParameterInspector`, который перед вызовом прописывает в свойства исходящего сообщение свойство типа `IMessageProperty` для JSON.
2. Класс `JSONMessageProperty` – это всего лишь обертка для лишнего кусочка метаданных который поставляются поведением. Применим этот довесок только для callback-параметра. Для тех кто забыл, callback-параметр это то с помошью чего данные `data` возвражаются через запрос `http://somewhere.com/x?callback=y` как `y(data)`, прописываются в `<script>` и исполняются.
3. Далее формируется фабрика `JSONPEncoderFactory`, которая производит энкодеры типа `JSONPEncoder`. Сам энкодер, казалось бы, тривиален – все, что он должен сделать так это обернуть вызов в название callback'а и вернуть его. Но поскольку его метод `WriteMessage()` перегружен, его приходится вызывать в нескольких местах.

Для того чтобы получить всю поддержку JSONP полностью, нужно скачать набор примеров по WCF с MSDN.

Web.config

Несмотря на то, что все, в принципе можно сделать в коде, на практике приходится достаточно сильно шаманить с `Web.config` ом дабы все заработало. В частности, нужно прописать новый binding:

```
<bindings>
  <customBinding>
    <binding name="jsonpBinding">
      <jsonpMessageEncoding/>
      <httpTransport manualAddressing="true"/>
    </binding>
  </customBinding>
</bindings>
```

А также добавить расширение для кодировки JSONP:

```
<extensions>
  <bindingElementExtensions>
    <add name="jsonpMessageEncoding"
    type="ConversationServer.JsonpSupport.JsonpBindingExtension, Conver
  </bindingElementExtensions>
</extensions>
```

Удаление .svc

Для REST-сервисов окончание `.svc` на конце сервиса как-то нелепо. К счастью его очень просто удалить. Для этого, мы можем создать свой собственный модуль который игнорирует это окончание:

```
public class RestModule : IHttpModule
{
  public void Init(HttpApplication context)
  {
    context.BeginRequest += (sender, args) =>
    {
      var ctx = HttpContext.Current;
      var path = ctx.Request.AppRelativeCurrentExecutionFilePath;
      int i = path.IndexOf('/', 2);
      if (i > 0)
      {
        var svc = path.Substring(0, i) + ".svc";
        var rest = path.Substring(i, path.Length - i);
        var qs = ctx.Request.QueryString.ToString();
        ctx.RewritePath(svc, rest, qs, false);
      }
    };
  }
  public void Dispose()
  {
  }
}
```

А далее просто прописываем его в секцию `system.web` в `Web.config`:

```
<system.web>
  <compilation debug="true" targetFramework="4.0" />
  <customErrors mode="Off"/>
  <httpModules>
    <add name="NoMoreSVC" type="ConversationServer.RestModule, Conversa
  </httpModules>
</system.web>
```

Вот собственно и все.

Заключение

Мой небольшой эксперимент по использованию WCF REST показал что, как и во многих других случаях, приходится даже для простенького сервиса городить громозкие структуры. К счастью, сделав это один раз, можно потом копировать реализацию в различные проекты.

Written by Dmitri

24 Март 2011 в 19:58

Опубликовано в .NET

Tagged with jsonp, rest, wcf

Комментариев: 6

Subscribe to comments with RSS.

Как насчет нового WCF Web API?
http://wcf.codeplex.com
не пробовали?

0

0

i
Rate This

Vladimir

26 Март 2011 at 10:43

Вот недавно скачал; буду разбираться.

0

0

i
Rate This

Dmitri

26 Март 2011 at 22:23

Феноменально, но факт – WCF не поставляется с поддержкой JSONP

3 версия не поставляется, а вот в 4 все уже из коробки работает.

0

0

i
Rate This

Алексей Калдузов

28 Март 2011 at 8:13

Точно! Спасибо за подсказку. Вот что бывает если импульсивно гуглить и сразу писать код :)

0

0

i
Rate This

Dmitri

28 Март 2011 at 11:33

А как встроить поддержку REST в ASP.NET приложение?
Интересует проблема в плане того чтобы сервис не был доступен не прошедшим автроизацию и обращение к методам сервиса прошедшим авторизацию было бы прозрачным

4

0

i
Rate This

Вадим

23 Февраль 2012 at 22:24

Добрый день, Дмитрий.
А как опубликовать сервис с удаленным окончанием svc на IIS? у меня не получилось

0

0

i
Rate This

Александр

16 Июль 2012 at 11:43

Обсуждение закрыто.

Подписаться на Дмитрий Нестерук