

SOAP и веб-сервисы XML на платформе .Net

Веб-сервисы XML предлагают такой уровень совместимости и взаимодействия в отношении операционных систем, платформ и языков, который ранее был просто недоступен.

Эндрю Троелсен (обладатель титула MVP (Most Valuable Professional in Microsoft))



Алексей Бойко

Если вы еще не работали с веб-сервисами XML, то наверняка слышали слово «SOAP». Настало время разобраться с этими понятиями.

Intro

Если вас интересует Интернет или сети поменьше, скорее всего рано или поздно вы столкнетесь с веб-сервисами XML. Веб-сервис XML – это не только веб-приложение, способное выводить информацию в браузер. Скорее это технология удаленного взаимодейст-

вия, позволяющая вызывать методы и свойства объекта в сети с помощью стандартных HTTP-запросов.

На практике это означает, что клиенты такого сервиса могут быть написаны на различных языках и для разных операционных систем.

В качестве информационного

«транспорта» между сервисом и клиентом можно использовать HTTP-методы GET или POST.

А можно «наложить» поверх еще один протокол – SOAP (Simple Object Access Protocol). Обычно так и поступают, так как в этом случае возможна передача сложных типов (включая поль-

зовательские). А классические методы GET и POST поддерживают только перечни, простые массивы и строки.

Пример SOAP-взаимодействия

SOAP-сообщение – это XML-документ, помещенный в тело HTTP-запроса.

Листинг 1. Структура SOAP-сообщения

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/
  soap-envelope" xmlns:tem="http://site.ru/">
  <soap:Header>
    <!-- Необязательная информация заголовков -->
  </soap:Header>
  <soap:Body>
    <!-- Информация вызова методов -->
  </soap:Body>
</soap:Envelope>
```

Взаимодействие клиента и сервиса происходит следующим образом:

- клиент формирует SOAP-запрос и отправляет его сервису;
- сервис на удаленном компьютере выполняет процедуру и отправляет SOAP-ответ.

Например, вот так может выглядеть SOAP-запрос, вызывающий метод HelloWorld() удаленного веб-сервиса XML:

Листинг 2. Пример SOAP-запроса

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/
  soap-envelope" xmlns:tem="http://site.ru/">
  <soap:Header/>
  <soap:Body>
    <tem:HelloWorld/>
  </soap:Body>
</soap:Envelope>
```

Метод HelloWorld(), как и полагается, возвращает строку «Здравствуй, мир!»:

Листинг 3. Пример SOAP-ответа

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/
  soap-envelope" xmlns:xsi="http://www.w3.org/2001/
 /XMLSchema-instance" xmlns:xsd="http://www.w3.org/
  2001/XMLSchema">
  <soap:Body>
    <HelloWorldResponse xmlns="http://site.ru/">
      <HelloWorldResult>Здравствуй, мир! </HelloWorldResult>
    </HelloWorldResponse>
  </soap:Body>
</soap:Envelope>
```

Создание веб-сервиса XML на платформе .NET 2.0

Создать сервис можно различными способами, мы будем использовать Visual Studio 2005. Нажмите «File → New → Web Site», в раскрывшемся окне выберите «ASP.NET web Service». По указанному при создании адресу вы обнаружите следующие файлы и директорию (см. рис. 1).

В принципе, сервис может содержать только один-единственный файл с расширением *.asmx.

(Для обозначения веб-сервисов .Net используется расширение *.asmx.) В данном случае это не так, посмотрите содержимое файла Service.asmx:

Листинг 4. В Service.asmx определен внешний файл поддержки

```
<%@ WebService Language="C#"
  CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

Атрибут CodeBehind указывает внешний файл, расположенный в папке App_Code, в котором размещен программный код, реализующий метод HelloWorld():

Листинг 5. Файл Service.cs, реализующий метод HelloWorld()

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}
```

Возможно создать единственный файл Service.asmx без кода поддержки, обладающий теми же функциональными возможностями:

Листинг 6. Service.asmx без внешнего кода поддержки

```
<%@ WebService Language="C#" Class="Service" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}
```

Нам это ни к чему, и так поступать мы не станем.

Как видите, единственный метод нашего веб-сервиса помечен атрибутом [WebMethod], информирующим среду выполнения ASP.NET о том, что этот метод доступен для поступающих HTTP-запросов. Не обозначенные таким атрибутом члены не будут доступны клиентским программам.

Для наших экспериментов вполне подходит такой простой сервис, осталось только опубликовать его.

Публикация веб-сервиса XML с помощью IIS

Речь пойдет не о публикации в Интернете, а о создании условий для тестирования нашего сервиса на локальном компьютере.



Рисунок 1. Структура веб-сервиса

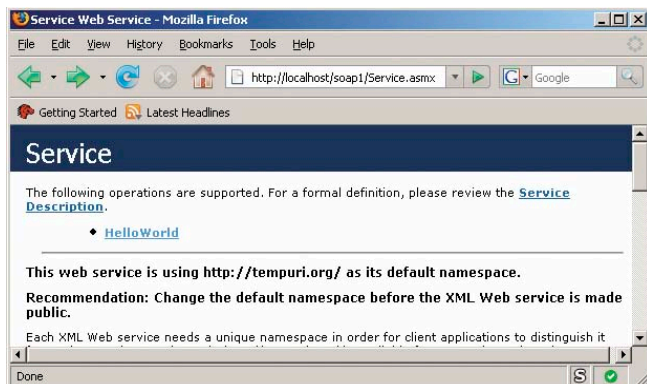


Рисунок 2. Страница тестирования веб-сервиса

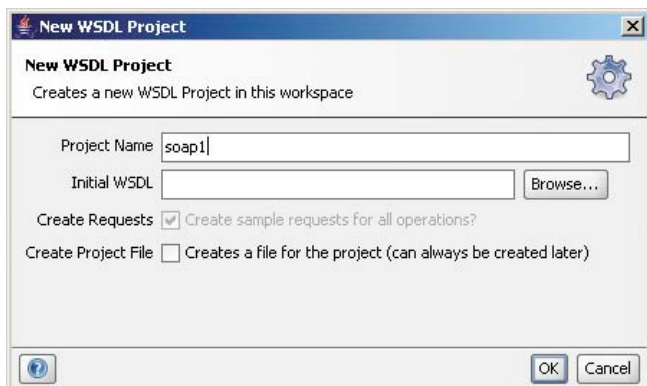


Рисунок 3. Создание проекта в soapUI

Первым делом установите IIS (Internet Information Server). Для этого откройте окно «Установка и удаление программ» и выберите в нем «Установка компонентов Windows». (Некоторые версии Windows не предполагают установку IIS, например Windows XP Home Edition.)

Примечание: сервер IIS лучше устанавливать раньше, чем .Net Framework, иначе придется настроить IIS на поддержку .Net-приложений с помощью запуска утилиты командной строки `aspnet_regiis.exe` (с флагом `/i`).

Создайте виртуальный каталог. Если вы используете Windows XP Pro, войдите в «Панель управления → Администрирование → Internet Information Services». В раскрывшемся окне выберите «Действие → Создать → Виртуальный каталог».

Будет запущен мастер создания виртуальных каталогов. Укажите в качестве псевдонима Soap1 и задайте путь к каталогу, в котором хотите разместить сервис, например `C:\Soap1`. Теперь скопируйте туда содержимое нашего веб-сервиса.

Наберите в адресной строке браузера `http://localhost/soap1/Service.asmx`, и вы должны увидеть страницу тестирования сервиса (см. рис. 2).

Просмотр SOAP-сообщений

Страница тестирования не позволяет отправлять и читать SOAP-сообщения.

По этой причине придется воспользоваться сторонними разработками, я рекомендую использовать soapUI. (Это бесплатный продукт, доступный по адресу <http://www.soapui.org>.)

После установки soapUI создайте новый проект под названием soap1, оставив поле Initial WSDL пустым (см. рис. 3).

Правой кнопкой мышки щелкните на только что созданном проекте и выберите «Add WSDL from URL». В открывшемся диалоговом окне введите `http://localhost/soap1/Service.asmx?wsdl`. Теперь у нас есть возможность отправлять SOAP-запросы к нашему сервису и просматривать полученные ответы. (Запросы будут сгенерированы soapUI автоматически.)

Что же такое этот WSDL? Документ WSDL описывает возможности взаимодействия клиентов с веб-сервисом. Там описывается, какие методы сервиса доступны для внешнего вызова, какие параметры они принимают и что возвращают, а также прочая информация, необходимая для удаленного взаимодействия. Такой документ можно составить вручную, а можно доверить его генерацию серверу, для этого достаточно приписать суффикс `?wsdl` к URL, указывающему на файл `*.asmx`.

Чтобы посмотреть WSDL-документ нашего сервиса, введите в браузере `http://localhost/soap1/Service.asmx?wsdl`. Информацию именно из этого документа использует soapUI для автоматической генерации SOAP-запросов.

SOAP Extensions

Как вы, наверное, заметили для создания веб-сервиса XML (как и клиента) не обязательно заботиться о виде SOAP-сообщений. Достаточно пометить нужные методы атрибутом `[WebMethod]`, и среда выполнения ASP.NET сама составит пакеты нужного формата.

На рис. 4 показаны запрос и ответ веб-сервиса, полученные с помощью программы soapUI.

Повторим еще раз. Запрос сгенерирован soapUI – при создании реальных клиентов для сервиса тоже не требуется вручную форматировать SOAP-пакеты. В создании ответа сервиса мы также не принимали непосредственного участия. Все это происходит автоматически.

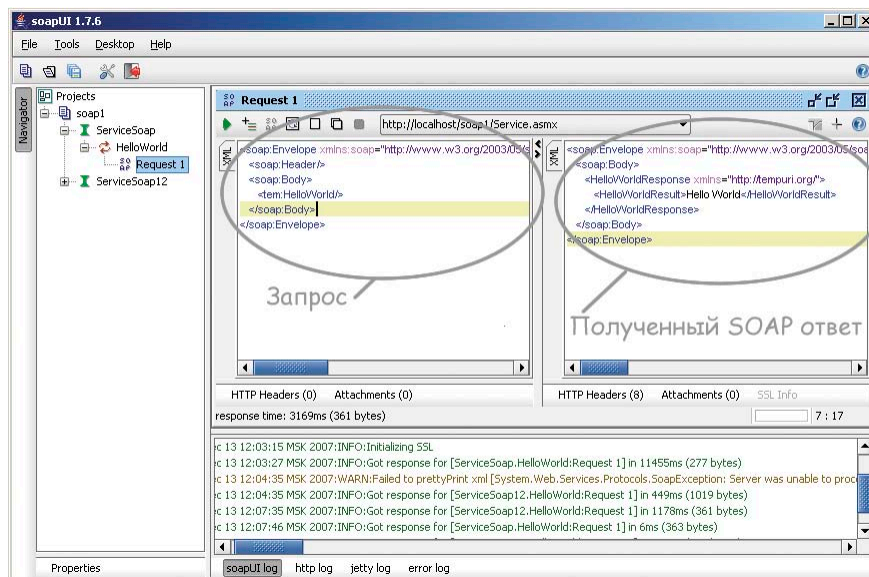


Рисунок 4. Запрос и ответ веб-сервиса, полученные с помощью программы soapUI

Однако вполне вероятно, что вам потребуется самостоятельно модифицировать эти пакеты. Например, произвести сжатие или шифрование передаваемых данных. В этом и есть назначение SOAP Extensions.

SOAP Extensions – это механизм, позволяющий произвольным образом изменять принимаемые и отправляемые SOAP-сообщения.

«Путь» SOAP-сообщения

Чтобы приступить к программированию, нужно рассмотреть путь, который проходит SOAP-сообщение, прежде чем оно будет получено и обработано соответствующим методом (см. **рис. 5**).

SOAP-сообщение можно рассматривать как XML-документ, описывающий передаваемый по сети объект. Перед тем как использовать переданный таким образом объект, его нужно восстановить (или, если хотите, собрать) из этого описания. Этой цели служит XML-сериализатор.

Входящие пакеты проходят десериализацию (восстановление объекта из XML-описания), а отправляемые сериализацию (создание XML-описания объекта).

На **рис. 5** показаны четыре точки (BeforeSerialize, AfterDeserialize, BeforeDeserialize, AfterSerialize), в которых мы, с помощью SOAP Extensions, можем перехватить SOAP-сообщение. Модифицировать его и отправить дальше.

Реализация SOAP Extension

Для начала определимся с заданием: пусть мы хотим изменить отправляемые веб-сервисом SOAP-пакеты как показано в **листинге 7**:

Листинг 7. «Старый» и новый ответы web-сервиса XML

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/">
      <HelloWorldResult>Hello World</HelloWorldResult>
    </HelloWorldResponse>
  </soap:Body>
</soap:Envelope>

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    зашифрованный текст
  </soap:Body>
</soap:Envelope>
```

План действий для реализации SOAP Extension:

- Создаем dll с классом, наследуемым от SoapExtension.
- Добавляем к нашему веб-сервису папку bin и кладем туда созданную dll.
- Добавляем к сервису файл web.config и вносим в него нужные изменения.

Роль папки bin и файла web.config рассмотрим позже.

Создание DLL с SOAP Extension

Создайте новый проект «Class Library» под названием SoapExtensionLib. В этом проекте потребуется реализовать

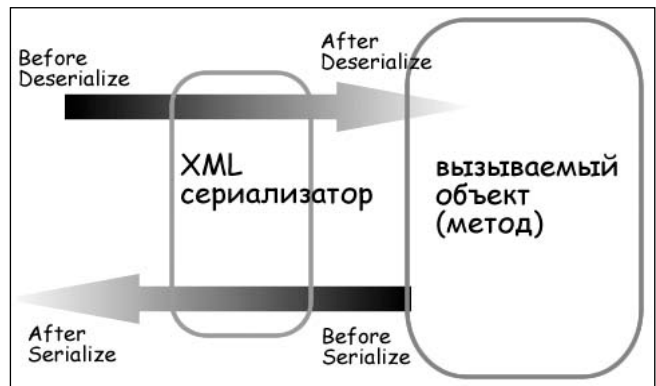


Рисунок 5. «Путь» SOAP-сообщения

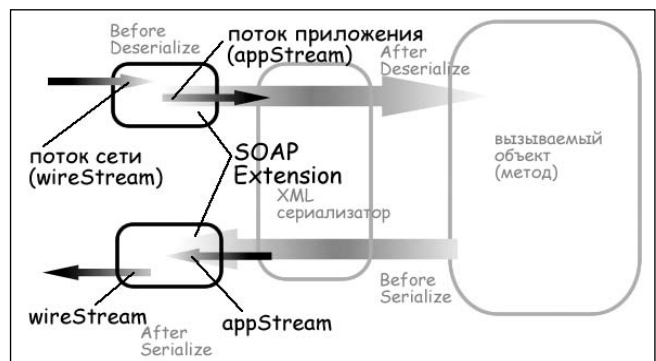


Рисунок 6. Представление SOAP extension как «врезки» в точках BeforeDeserialize и AfterSerialize

только один класс, который будет выполнять необходимые нам модификации SOAP-пакетов. Этот класс должен быть наследованным от класса SoapExtension.

Листинг 8. Создание класса, наследуемого от SoapExtension

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.IO;
using System.Net;
using System.Xml;

public class TraceExtension : SoapExtension
{
}
```

Каждое SOAP Extension получает как параметр поток, содержащий передаваемый по сети объект (до или после сериализации). И обязано возвращать поток.

SOAP Extension можно представить как «врезку», которую можно поставить в одной или всех четырех точках (BeforeSerialize, AfterDeserialize, BeforeDeserialize, AfterSerialize), показанных на **рис. 5**. В каждой точке таких «врезок» может быть сколько угодно (см. **рис. 6**).

Для получения этих потоков используется метод ChainStream.

Листинг 9. Реализация метода ChainStream

```
public class TraceExtension : SoapExtension
{
    Stream wireStream;
    Stream appStream;

    // метод в качестве входного параметра
    // получает поток, содержащий передаваемый объект
```



```

public override Stream ChainStream(Stream stream)
{
    wireStream = stream;
    appStream = new MemoryStream();
    return appStream;
}
...
}

```

В точке BeforeDeserialize поток wireStream содержит принятый из сети SOAP-запрос. Этот SOAP-запрос нужно передать потоку приложения (потоку appStream).

А в точке AfterSerialize необходимо передать потоку wireStream отправляемый в сеть SOAP-ответ, который будет размещаться в appStream.

Для работы с потоками в каждой из четырех точек нужно реализовать метод ProcessMessage.

Листинг 10. Реализация метода ProcessMessage, не модифицирующего SOAP-сообщения

```

// ProcessMessage, выполняющий обязательное копирование
// потоков в двух точках (BeforeDeserialize и AfterSerialize)
public override void ProcessMessage(SoapMessage message)
{
    switch (message.Stage)
    {
        // в точке BeforeDeserialize необходимо передать
        // SOAP-запрос из потока сети (wireStream)
        // в поток приложения (appStream)
        case SoapMessageStage.BeforeDeserialize:
            Copy(wireStream, appStream);
            appStream.Position = 0;
            break;

        // в точке AfterSerialize необходимо передать
        // SOAP-ответ из потока приложения в поток сети
        case SoapMessageStage.AfterSerialize:
            appStream.Position = 0;
            Copy(appStream, wireStream);
            break;
    }

    void Copy(Stream from, Stream to)
    {
        TextReader reader = new StreamReader(from);
        TextWriter writer = new StreamWriter(to);
        writer.WriteLine(reader.ReadToEnd());
        writer.Flush();
    }
}

```

Листинг 10 можно воспринимать как болванку для дальнейших экспериментов. В такой реализации метода ProcessMessage нет никакого смысла – исходящий SOAP-ответ никак не модифицируется. Исправим это:

Листинг 11. Реализация метода ProcessMessage, модифицирующего SOAP-ответ

```

public override void ProcessMessage(SoapMessage message)
{
    switch (message.Stage)
    {
        case SoapMessageStage.AfterSerialize:
            WriteOutput(message);
            break;

        // часть кода вырезана для экономии места
    }

    // перепишем SOAP-ответ
    public void WriteOutput(SoapMessage message)
    {
        appStream.Position = 0;
        // создадим XML документ из потока
        XmlDocument document = new XmlDocument();
        document.Load(appStream);

        // Для использования XPath нужно определить

```

```

// NamespaceManager
XmlNamespaceManager nsmgr =
    new XmlNamespaceManager(document.NameTable);
nsmgr.AddNamespace("soap",
    "http://schemas.xmlsoap.org/soap/envelope/");

// получим ссылку на узел <soap:Body>
XmlNode ResultNode =
    document.SelectSingleNode("//soap:Body", nsmgr);
// заменим содержимое узла
ResultNode.InnerText = "зашифрованный текст";
// очистим поток и запишем в него новый SOAP-ответ
appStream.SetLength(0);
appStream.Position = 0;
document.Save(appStream);

// ОБЯЗАТЕЛЬНОЕ ДЕЙСТВИЕ
// передадим SOAP-ответ из потока приложения (appStream)
// в поток сети (wireStream)
appStream.Position = 0;
Copy(appStream, wireStream);
}

```

Далее нужно определить два метода (один из которых является перегруженным, т.е. может вызываться с разными наборами параметров), которые в нашем случае не нужны. Однако мы обязаны определить их в соответствии с правилами наследования от класса SoapExtension.

Листинг 12. Другие обязательно реализуемые методы

```

// В соответствии с правилами наследования мы обязаны
// определить эти методы, однако мы их никак
// не используем
public override object
    GetInitializer(LogicalMethodInfo methodInfo,
        SoapExtensionAttribute attribute)
{
    return null;
}

public override object GetInitializer(Type WebServiceType)
{
    return null;
}

public override void Initialize(object initializer)
{
    return;
}

```

Всё, компилируем проект. Наконец-то мы получили dll с SOAP Extension. Полный листинг SoapExtensionLib.dll находится на сайте журнала www.samag.ru в разделе «Исходный код».

Настройка веб-сервиса для работы с SOAP Extension

Опять откройте проект нашего веб-сервиса XML с помощью Visual Studio. Нажмите «WebSite → Add Reference» и выберите созданный ранее SoapExtensionLib.dll.

К проекту автоматически будет добавлена папка Bin. На файлы *.dll, расположенные в папке Bin, приложение ссылается автоматически.

Теперь положите в директорию веб-сервиса файл Web.Config следующего содержания:

Листинг 13. Файл Web.Config

```

<?xml version="1.0"?>

<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>

```

```

<!--
Весь текст этого файла,
за исключением секции webServices,
сгенерирован автоматически
-->
<webServices>
  <soapExtensionTypes>
    <add type="TraceExtension, J
      SoapExtensionLib"
      priority="1"
      group="0" />
  </soapExtensionTypes>
</webServices>

  <compilation J
    debug="true"/>
  <authentication J
    mode="Windows"/>
</system.web>
</configuration>

```

Теперь структура нашего сервиса выглядит так, как показано на **рис. 7**.

С помощью файла Web.Config мы информируем среду ASP.NET о том, что мы добавили к веб-сервису XML Soap Extension, реализованное в классе TraceExtension который расположен в файле SoapExtensionLi.dll.

Листинг 14. Секция webServices в файле Web.Config

```

<webServices>
  <soapExtensionTypes>
    <add type="TraceExtension, SoapExtensionLib"
      priority="1"
      group="0" />
  </soapExtensionTypes>
</webServices>

```

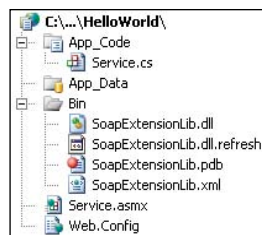


Рисунок 7. Структура веб-сервиса

Как вы уже знаете, можно сделать множество SOAP Extension, и поток, несущий в себе передаваемый объект (до или после сериализации), будет проходить через каждое из них. Порядок прохождения потоком различных SOAP Extension задается с помощью атрибутов priority и group.

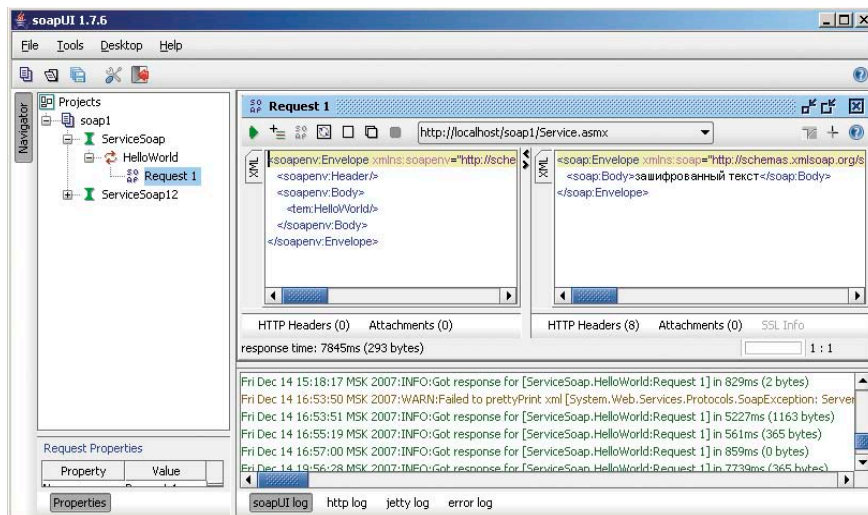


Рисунок 8. Просмотр результата в soapUI

Стоит отметить, что сконфигурировав данным способом Web.Config, мы информируем среду о том, что наше SOAP Extension будет вызываться для каждого метода сервиса, помеченного атрибутом [WebMethod]. Существует возможность создать свой атрибут и пометить им только те методы, для которых нужно вызвать SOAP Extension.

Листинг 15. Пример использования собственного атрибута

```

[WebMethod]
[MyExtension]
public string HelloWorld() {
    return "Hello World";
}

```

Для этого требуется добавить в SoapExtensionLi.dll класс, наследованный от SoapExtensionAttribute (см. **рис. 8**).

Заключение

В данной статье отражены основные моменты построения и функционирования веб-сервисов XML на платформе .Net. Я надеюсь, что изложенного материала будет вполне достаточно, чтобы при необходимости вы смогли заняться более глубоким изучением темы.

Электронная версия журнала

**СИСТЕМНЫЙ
администратор**

- Удобное чтение online
- Архивы с 2003 года
- Оперативный выход новых номеров
- Более 60 номеров и 700 статей
- Бесплатные статьи

Стоимость подписки:

- 327 рублей за 3 месяца (109 рублей за номер)
- 606 рублей за 6 месяцев (101 рубль за номер)

Мы принимаем к оплате WebMoney, PayPal и кредитные карты.

<http://av5.com/journals>

Все материалы лицензированы.