**Лекція 15. Контракти даних WCF – технології**

В основі роботи роботи WCF сервісів лежить протокол SOAP (Simple Object Access Protocol), який працює поверх протоколу HTTP і передбачає передачу повідомлень по мережі в форматі XML. Отже сервіс WCF при взаємодії з клієнтом повинен спочатку серіалізовать дані в XML, а потім передати їх клієнту через SOAP, клієнт у свою чергу повинен десеріалізовать отримане повідомлення. Відповідно, всі типи даних, які ми використовуємо в контрактах нашого сервісу повинні бути серіалізуемим

Контракт даних - формальна угода між службою та клієнтом, абстрактно описує дані, обмін якими відбувається. Це означає, що для взаємодії клієнт і служба не зобов'язані спільно використовувати одні й ті ж типи, досить спільно використовувати одні й ті ж контракти даних. Контракт даних для кожного параметра і повертається типу чітко визначає, які дані серіалізуются (перетворюються на XML) для обміну.

Основні відомості про контракти даних

За замовчуванням в Windows Communication Foundation (WCF) для серіалізациі і десеріалізациі даних (перетворення в XML і назад) використовується модуль серіалізациі, званий серіалізатор контракту даних. Всі типи-примітиви .NET Framework, такі як integer та string, а також деякі типи, які обробляються як примітиви, такі як DateTime і XmlElement, можуть бути серіалізовані без додаткової обробки і вважаються типами, які за умовчанням містять контракти даних. Багато типів .NET Framework також містять контракти даних. .

Для серіалізациі нових створених складних типів необхідно визначити контракти даних. За замовчуванням DataContractSerializer визначає контракт даних і серіалізуются всі відкриті типи. Всі відкриті властивості читання / запису і поля типу серіалізуются. Можна виключати члени з серіалізациі за допомогою IgnoreDataMemberAttribute. Також можна явно створювати контракт даних за допомогою атрибутів DataContractAttribute і DataMemberAttribute. Зазвичай це робиться за допомогою застосування атрибута DataContractAttribute до типу. Даний атрибут може бути застосований до класів, структурам і перечислениям. Після цього необхідно застосувати атрибут DataMemberAttribute до кожного члена типу контракту даних, щоб вказати, що він є членом даних, який необхідно серіалізовать.

Розглянемо наступний приклад, контракт сервісу:

[ServiceContract]

public interface IService

{

[OperationContract]

int Sum (int firstNumber, int secondNumber);

[OperationContract]

double Divide (int firstNumber, int secondNumber);

}

В даному випадку у механізму WCF не виникне жодних проблем з сериализацией, оскільки обидва методи нашого сервісу використовують тільки елементарні типи даних: int і double. Однак на практиці, настільки прості сервіси практично не зустрічаються, в більшості випадків між клієнтом і сервісом необхідно передавати складні, комплексні типи даних, наприклад:

public class MyComplexNumber

{

public int x; public int y; public int i; public string notes;

}

[ServiceContract]

public interface IService

{

[OperationContract]

MyComplexNumber Sum (MyComplexNumber firstNumber, MyComplexNumber secondNumber);

[OperationContract]

MyComplexNumber Divide (MyComplexNumber firstNumber, MyComplexNumber secondNumber);

}

Тут функції сервісу оперують нашим, похідним, типом даних. Відповідно, клас MyComplexNumber повинен коректно серіалізовані в формат XML. WCF надає нам досить гнучкий механізм управління цим процесом. Ми можемо налаштувати сериализацию нашого клас двома наступними способами:

) Використовувати стандартний механізм - DataContractSerializer. У цьому випадку ми можемо залишити код, наведений вище, як є і наш сервіс самостійно його серіалізуются. При цьому будуть серіалізовані всі поля, оголошені з модифікатором доступу public. А також всі публічні властивості, що мають відкриті методи get і set. Ми також можемо вказати, які поля ми не хочемо серіалізовать - їх необхідно позначити атрибутом IgnoreDataMemberAttribute, як показано нижче:

public class MyComplexNumber

{

public int x; public int y; public int i;

[IgnoreDataMember]

public string notes;

}

Атрибут IgnoreDataMemberAttribute знаходиться в збірці System.Runtime.Serialization, яку необхідно додатково підключити до проекту.

2) Використовувати атрибути DataContract і DataMember. У цьому випадку ми говоримо механізму WCF, що ми самі, явно, вкажемо всі поля і властивості, які необхідно серіалізовать. Напишемо код, який за результатом будемо аналогічний попередньому:

[DataContract]

public class MyComplexNumber

{

[DataMember]

public int x;

[DataMember]

public int y;

[DataMember]

public int i;

public string notes;

}

Атрибутом DataContract позначається весь клас, а атрибутом DataMember позначаються поля і властивості, які необхідно буде серіалізувати. Потрібно сказати кілька слів про взаємодію цих двох способів. Можна звичайно використовувати атрибут DataMember в класі, не зазначеному атрибутом DataContract або ставити атрибут IgnoreDataMember в класі, з атрибутом DataContract. Ці дії не викличуть помилку компіляції або виконання, проте бажаного ефекту не вийде, оскільки дані атрибути будуть ігноруватися.

**Серіалізация перерахувань**

Отже, ми розглянули аспекти пов'язані з сериализацией полів і властивостей класу, які представляють прості типи даних. Тепер розглянемо ситуацію, коли тип даних одного з полів класу - створене нами перерахування (enum). Для перерахувань до WCF також передбачена система серіалізациі, дуже схожа на ту, що ми розглянули вище. Наведемо наступний приклад:

[ServiceContract]

public interface IService

{

[OperationContract]

MyInformation GetInformation ();

}

public class MyInformation

{

public string Message; public string Author;

public ProtectionLevelEnum protectionLevel;

}

public enum ProtectionLevelEnum

{

High, Medium, Low, Custom

};

Контрактом даних є клас MyInformation, одне з полів якого має тип перерахування. Щоб перерахування ProtectionLevelEnum могли використовувати і клієнт і сервіс - його також необхідно серіалізовать і передавати в повідомленні. Налаштувати його сериализацию можна двома способами:

1) Використовуючи атрибути DataContract і EnumMember:

[DataContract]

public enum ProtectionLevelEnum

{

[EnumMember]

High,

[EnumMember]

Medium,

[EnumMember]

Low,

Custom

};

При цьому будуть серіалізовані всі члени перерахування, відмічені атрибутом EnumMember.

2) Використовувати стандартний механізм серіалізациі - при цьому всі елементи перерахування будуть серіалізовані, так как-будто вони відзначені атрибутом EnumMember, за винятком тих значень, для яких ми встановили атрибут NonSerialized:

public enum ProtectionLevelEnum

{

High, Medium, Low,

[NonSerialized]

Custom

};

Серіалізация колекцій

Наступний важливий тип даних, який може бути членом контракту даних - це колекції. Розглянемо особливості серіалізациі цього типу:

1) Взаємозамінні колекції. Всі колекції-списки мають однаковий контракт даних, отже наступні три контракти даних будуть повністю еквівалентні:

[DataContract (Name = "RootElement")]

public class MyData

{

[DataMember]

string [] mylist;

}

[DataContract (Name = "RootElement")]

public class MyData

{

[DataMember]

List <string> mylist;

}

[DataContract (Name = "RootElement")]

public class MyData

{

[DataMember]

Collection <string> mylist;

}

Всі три контракти в результаті серіалізуются в наступний XML код:

<RootElement>

<mylist>

<string> ... </ string> <string> ... </ string> <string> ... </ string> ....

</ mylist>

</ RootElement>

Властивість Name, яке устанавл в атрибуті DataContract визначає назву кореневого елемента XML коду, аналогічно його можна використовувати і для атрибута DataMember.

2) Як параметри функцій сервісу не можуть виступати багатовимірні масиви, наступний код викличе виключення при запуску сервісу:

[ServiceContract]

public interface MyService

{Double GetElementsSum (double [,] mas); }

Однак підтримуватися вкладені колекції і вищенаведений код можна переписати таким чином:

[ServiceContract]

public interface MyService

{Double GetElementsSum (List <double []> mas); }

3) Налаштування типів колекції. Дана настройка застосовується за допомогою атрибуту CollectionDataContractAttribute, розглянемо наступний приклад:

public class MyCollection: Collection <string> {}

Якщо тепер використовувати MyCollection в якості поля контракту даних, то колекція серіалізуются наступним чином:

<ArrayOfstring>

<string> ... </ string> <string> ... </ string> <string> ... </ string>

</ ArrayOfstring>

Ім'я за замовчуванням контрактів даних колекцій списків, якщо воно не перевизначити, є рядком "ArrayOf ...". Якщо ж помітити нашу колекцію атрибутом CollectionDataContractAttribute:

[CollectionDataContract]

public class MyCollection: Collection <string> {}

то тепер ім'я і простір імен контракту даних колекції залежать від типу самої колекції і ми отримаємо приблизно наступний XML код:

<MyCollection>

<string> ... </ string> <string> ... </ string> <string> ... </ string> ....

</ MyCollection>

Також ми можемо в атрибуті CollectionDataContractAttribute явно вказати назви елементів в XML дереві, це робиться за допомогою атрибутів Name і ItemName:

[CollectionDataContract (Name = "mysongs", ItemName = "song")]

public class MyCollection: Collection <string> {}

XML код в результаті буде наступним:

<mysongs>

<song> ... </ song> <song> ... </ song> <song> ... </ song>

</ mysongs>

3) Налаштування словників. Ми також можемо управляти параметрами серіалізациі колекцій-словників, встановлювати імена для ключів і значень словника можна за допомогою того ж атрибуту CollectionDataContractAttribute і параметрів KeyName і ValueName. Розглянемо приклад:

[CollectionDataContract (Name = "mysongs", ItemName = "song", KeyName = "id", ValueName = "text")]

public class MyDictionary: Dictionary <int, string> {}

результатом серіалізациі буде код наступний XML код:

<mysongs>

<song>

<id> 1 </ id> <text> abc </ text>

</ song>

<song> <id> 2 </ id> <text> def </ text>

</ song>

</ mysongs>

4) Також розглянемо деякі варіанти використання атрибута CollectionDataContractAttribute з колекціями в контрактах даних, що призводять до помилки InvalidDataContractException при запуску сервісу:

Застосування атрибуту DataContractAttribute до типу, який був позначений атрибутом CollectionDataContractAttribute, або до одного з успадкованих від нього типів;

- Застосування атрибуту CollectionDataContractAttribute до типу, яка не є колекцією;

- Спроба задати параметри KeyName або ValueName атрибуту CollectionDataContractAttribute, застосовуваним до типу, яка не є словником.

**Значення елементів контракту даних за замовчуванням**

Один зі способів зменшення розмірів контракту даних - це виключення з нього полів мають значення за замовчуванням (наприклад null), тобто, якщо поле контракту даних при передачі через сервіс буде мати значення за замовчуванням, то воно не буде включено в серіалізовані повідомлення. Для цього необхідно встановити властивість EmitDefaultValue атрибута DataMember в значення false.

Також в атрибуті DataMember є властивість IsRequired, при установці якого в значення true, ми говоримо механізму WCF, що це поле повинно бути обов'язково присутніми в контракті даних. Цілком логічно. що одночасне застосування властивостей EmitDefaultValue = false і IsRequired = true неможливо, так як вони мають протилежний ефект. Відомі типи контрактів даних

Зазвичай при передачі даних між клієнтом і сервісом обидві сторони спільно використовують типи даних, що відносяться до контрактів даними (наприклад коли клієнт і сервіс знаходяться в одному рішенні). Проте бувають ситуації, коли це не так. Розглянемо наступний код оголошення сервісу:

[DataContract]

public class Shape {}

[DataContract]

public class Circle: Shape {}

[DataContract]

public class Triangle: Shape {}

[DataContract]

public class FigureData

{

[DataMember]

public Shape figure;

}

[ServiceContract]

public interface FigureOperations

{FigureData GetFigure (); }

Контрактом даних тут виступатиме клас FigureData. У випадку, якщо реалізація сервісу буде наступною, наш код відмінно спрацює:

public class MyService: FigureOperations

{Public FigureData GetFigure ()

{Return new FigureData () {figure = new Shape ()}; }

}

Однак, при наступній реалізації виникне помилка, при передачі даних:

public class MyService: FigureOperations

{

public FigureData GetFigure ()

{Return new FigureData () {figure = new Circle ()}; }

}

Присвоєння полю figure значення типу Circle абсолютно коректно з точки зору поліморфізму в .NET, проте неприпустимо в WCF. Справа в тому, що клієнт сервісу знає лише про тих типу даних, які явно вказані в контракті даних (у нашому випадку - про клас Shape). Щоб це виправити необхідно використовувати атрибут KnownTypeAttribute наступним чином:

[DataContract]

[KnownType (typeof (Circle))]

[KnownType (typeof (Triangle))]

public class FigureData

{

public Shape figure;

}

Тим самим ми вказуємо значення яких типів можуть брати поля нашого контракту даних.

**Спадкування і контракти даних**

У багатьох випадках клас, який, в кінцевому рахунку, буде виступати контрактом даних нашого сервісу може бути частиною ієрархії класів. Давайте розглянемо обмеження, які накладає механізм WCF в такій ситуації. Тут є одне основне правило: класи, без атрибуту DataContractAttribute можуть успадковуватися від класів, позначених атрибутом DataContractAttribute, проте зворотне неможливо і наступний код викличе помилку при запуску сервісу:

public class BaseData

{Public string author; }

[DataContract]

public class MyData: BaseData

{

[DataMember]

public int id;

[DataMember]

public string text;

}

Щоб це спрацювало необхідно позначити клас BaseData атрибутом DataContract, як показано далі:

[DataContract]

public class BaseData

{Public string author; }

[DataContract]

public class MyData: BaseData

{

[DataMember]

public int id;

[DataMember]

public string text;

}