**Лекція 16. WCF Використання контрактів повідомлень і помилок**

**Контракти повідомлень**

Зазвичай при побудові додатків Windows Communication Foundation (WCF) розробники приділяють пильну увагу структурам даних і питань серіалізациі; при цьому їм не потрібно займатися структурою повідомлень, в яких передаються дані. Для таких додатків створення контрактів даних для параметрів або повертаються значень являє собою досить просту задачу. (Докладніше див. ВЗаданіе передачі даних в контрактах служб.)

Тим не менш, іноді повний контроль над структурою повідомлення SOAP настільки ж важливий, як і контроль над його вмістом. Це особливо актуально, якщо важливо забезпечити взаємодію або контролювати певні питання безпеки на рівні повідомлення або частини повідомлення. У таких випадках можна створити контракт повідомлення, що дозволяє вказати точну структуру для потрібного повідомлення SOAP.

У цьому розділі розглядається використання різних атрибутів контрактів повідомлень для створення конкретного контракту повідомлення для даної операції.

Використання контрактів повідомлень в операціях

WCF підтримує операції, що моделюються на основі або стилю віддаленого виклику процедур (RPC), або стилю повідомлень. В операції в стилі RPC можна використовувати будь серіалізуемим тип і мати у своєму розпорядженні функції, доступні локальним викликам, такі як множинні параметри і параметри ref та out. У цьому стилі обрана форма серіалізациі визначає структуру даних у відповідних повідомленнях, а самі повідомлення для підтримки операції створюються середовищем виконання WCF. Це дозволяє розробникам, не знайомим з протоколом SOAP і повідомленнями SOAP, швидко і просто створювати і використовувати додатки служб.

У наведеному нижче прикладі коду показана операція служби, змодельована на основі стилю RPC.

[OperationContract]

public BankingTransactionResponse PostBankingTransaction (BankingTransaction bt);

Як правило, контракту даних достатньо для визначення схеми для сообщеній.Напрімер, в попередньому прикладі для більшості додатків буде досить, якщо BankingTransaction і BankingTransactionResponse матимуть контракти даних для визначення вмісту відповідних повідомлень SOAP.Дополнітельние відомості про контракти даних див. Розділ Використання контрактів даних .

Але час від часу необхідно точно контролювати процес передачі структури повідомлення SOAP.Наіболее поширеним сценарієм в такому випадку є вставка настроюються заголовків SOAP.Другой часто використовуваний сценарій - визначити властивості безпеки для заголовків і тіла повідомлення, т. Е. Вирішити, чи будуть ці елементи забезпечуватися цифровим підписом і шіфроваться.Наконец, для деяких сторонніх стеків SOAP необхідно, щоб повідомлення мали конкретний формат.Такого роду контроль забезпечується операціями в стилі повідомлень.

Операція в стилі повідомлень має максимум один параметр і одне повертається значення, причому обидва типи є типами повідомлень, т. Е. Вони серіалізуются безпосередньо в задану структуру повідомлення SOAP.Ето може бути будь-який тип, зазначений атрибутом MessageContractAttribute, або тип Message.В наведеному нижче прикладі коду показана операція, схожа з показаної вище операцією в стилі RPC, проте змодельована на основі стилю повідомлень.

Наприклад, якщо і BankingTransaction, і BankingTransactionResponse являють собою типи, які є контрактами повідомлень, код в наступних операціях є допустимим.

[OperationContract]

BankingTransactionResponse Process (BankingTransaction bt);

[OperationContract]

void Store (BankingTransaction bt);

[OperationContract]

BankingTransactionResponse GetResponse ();

У той же час наступний код є неприпустимим.

[OperationContract]

bool Validate (BankingTransaction bt);

// Invalid, the return type is not a message contract.

[OperationContract]

void Reconcile (BankingTransaction bt1, BankingTransaction bt2);

// Invalid, there is more than one parameter.

Для кожної операції за участю типу з контрактом повідомлення, що не відповідає одному з допустимих шаблонів, викликається ісключеніе.Конечно, на операції без участі типів з контрактами повідомлень ці обмеження не поширюються.

Якщо тип має і контракт повідомлення, і контракт даних, при використанні типу в операції до уваги береться лише його контракт повідомлення.

Визначення контрактів повідомлень

Щоб визначити контракт повідомлення для типу (т. Е. Визначити зіставлення між типом і конвертом SOAP), застосуйте до типу атрибут MessageContractAttribute. Потім застосуєте атрибут MessageHeaderAttribute до тих членів типу, які потрібно перетворити на заголовки SOAP, і застосуєте атрибут MessageBodyMemberAttribute до тих членів, які потрібно перетворити в частині тіла повідомлення SOAP.

Наведений нижче код являє собою приклад використання контракту повідомлення.

[MessageContract]

public class BankingTransaction

{

   [MessageHeader] public Operation operation;

   [MessageHeader] public DateTime transactionDate;

   [MessageBodyMember] private Account sourceAccount;

   [MessageBodyMember] private Account targetAccount;

   [MessageBodyMember] public int amount;

}

При використанні цього типу як параметр операції формується наступний конверт SOAP:

Xml

<s: Envelope xmlns: s = "http://schemas.xmlsoap.org/soap/envelope/">

   <s: Header>

     <h: operation xmlns: h = "http://tempuri.org/" xmlns = "http://tempuri.org/"> Deposit </ h: operation>

     <h: transactionDate xmlns: h = "http://tempuri.org/" xmlns = "http://tempuri.org/"> 2012-02-16T16: 10: 00 </ h: transactionDate>

   </ s: Header>

   <s: Body xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns: xsd = "http://www.w3.org/2001/XMLSchema">

     <BankingTransaction xmlns = "http://tempuri.org/">

       <amount> 0 </ amount>

       <sourceAccount xsi: nil = "true" />

       <targetAccount xsi: nil = "true" />

     </ BankingTransaction>

   </ s: Body>

</ s: Envelope>

Зверніть увагу, що operation і transactionDate виглядають, як заголовки SOAP, а тіло SOAP складається з елемента-оболонки BankingTransaction, що містить елементи sourceAccount, targetAccount і amount.

Атрибути MessageHeaderAttribute і MessageBodyMemberAttribute можна застосовувати до всіх полів, властивостям і подіям, незалежно від того, є вони відкритими, закритими, захищеними або внутрішніми.

Атрибут MessageContractAttribute дозволяє вказати атрибути WrapperName і WrapperNamespace, від яких залежить ім'я елемента-оболонки в тілі повідомлення SOAP.

Обробка помилок в Windows Communication Foundation (WCF)

вступ

З обробкою помилок стикався і стикається кожен день будь-який розробник. І ми до цього звикли, звикли робити перевірки на null та інше, щоб уникнути наших «улюблених» виключень або сформувати і згенерувати власне, більш інформативне виняток. Але коли мова йде про розподілених додатках, тут слід враховувати їх специфіку.

У розподіленої системі в якості джерела помилок може виступати як клієнтська, так і сервісна сторона.

Будь-яка операція сервісу може викликати виключення, про який би хотілося якось оповістити клієнта. На цьому етапі і починає працювати специфіка, про яку я говорив раніше, - специфіка передачі помилок між клієнтом і сервісом. Про неї і буде розмова.

Концепція обробки помилок.

Говорячи про WCF, слід чітко розуміти основну концепцію сервісів. Сервіс в WCF розглядається, як щось самодостатнє і абсолютно незалежне від своїх клієнтів. Крім цього, сервіс повинен бути ВІДМОВОСТІЙКО, інакше кажучи, він не повинен переходити в помилкове стан в разі будь-якої некритичною для нього ситуації. Якщо в сервісі і відбувається виключення, то клієнтові не обов'язково знати про всі тонкощі цього винятку, йому важливий факт помилки і основна причина її виникнення. Всі тонкощі важливі тільки на етапі налагодження, але в реальному житті вони не повинні виходити за межі сервісу. Для того щоб якось стежити за життям сервісу, зазвичай користуються журналом подій або визначають власну реалізацію логування.

Помилки і виключення WCF

У WCF робота hosting-процесу організована таким чином, що виключення, що виникають в WCF-сервісі, не порушують роботу цього процесу, а також роботу інших запущених сервісів і клієнтів, що не мають відношення до цих винятків. Як і в будь-який інший технології віддаленого взаємодії, помилка сервісу (fault) передається тільки тому клієнту, який став її ініціатором.

Ясно, що для передачі помилки, між клієнтом і сервісом повинна існувати деяка домовленість, завдяки якій клієнт однозначно сприймає сервісну інформацію і навпаки. Частиною такої домовленості служать помилки, про які знає і клієнтська і сервісна сторона.

Намагаючись звернутися до сервісу, клієнт, фактично, може зіткнутися з трьома типами помилок:

♣ Комунікаційні помилки. Виникають, наприклад, коли немає підключення до мережі, коли вказано неправильну адресу сервісу, коли запущений host-процес і т.д. Ці винятки визначені на клієнтській стороні класом винятку CommunicationException.

♣ Помилки стану каналу. Пов'язані зі станом проксі і каналів. Такий помилкою може бути, наприклад, спроба доступу до вже закритому проксі, яка закінчується винятком класу ObjectDisposedException, або, наприклад, невідповідність в контракті і зв'язуванні рівня захисту безпеки.

♣ Помилки запитів. Відбуваються при запитах до сервісу.

З першими двома типами помилок, як правило, все зрозуміло, до того ж вони забезпечуються самою інфраструктурою WCF. Що стосується помилок, що виникають при зверненні до сервісу, - тут все набагато складніше і цікавіше, оскільки існує безліч причин виникнення помилки, при виконанні сервісного запиту. У зв'язку з цим, саме помилки запитів, і становлять найбільший інтерес, власне їм і присвячується ця стаття.

Поточна реалізація логічної моделі WCF на .NET-платформі така, що винятки, викликані на стороні сервісу, як правило, досягають клієнта у вигляді FaultException.

public

class FaultException: CommunicationException

{...}

Примірники сервісу і Singleton

Коли на сервісі відбувається виняток, WCF не руйнує сервісний процес, однак, може впливати на екземпляр віддаленого об'єкта і можливість клієнта продовжувати використовувати комунікаційний канал.

Якщо запит викликає непередбачене виключення, то проксі клієнта генерує FaultException. Як правило, сервісні винятку переводять канал в стан помилки, так що навіть якщо відловити таке виключення на клієнті, наступні запити викличуть CommunicationObjectFaultedException. Однак така поведінка справедливо не завжди, і все багато в чому залежить від використовуваного зв'язування. Так, наприклад, якщо визначити BasicHttpBinding, то після виключення канал залишається працездатним.

Наскільки я знаю, багато розробників часто роблять на клієнтській стороні singleton-посилання на WCF-сервіси, щоб кожен раз не створювати канал. І, безсумнівно, всі, хто так робив, хоч раз стикалися з вищевказаної помилкою. Тому слід бути дуже уважним при використанні singleton'ов і, в разі виникнення помилки каналу, заново намагатися отримати доступ до сервісу. Наступний приклад, почасти демонструє подібну логіку, хоча і не претендує на звання кращого.

  Приклад 1 - Приклад уникнення помилки комунікації

private ICalculatorContract \_сalculatorService;

public ICalculatorContract CalculatorService

{

   get

   {

     if (\_сalculatorService == null)

     {

       // Отримання посилання на віддалений об'єкт

     }

     return \_сalculatorService;

   }

}

public

void ClientMethod ()

{

   try

   {

     CalculatorService.Divide (9, 0);

   }

   catch (Exception)

   {

     // Комунікаційний канал зруйнований

     \_сalculatorService = null;

   }

}

Сервісні помилки

Помилки, що виникають на стороні сервісу, передається клієнтській стороні по мережі. Для їх передачі в WCF використовується протокол SOAP. Таким чином, помилка, що виникла на сервісі, перед тим, як буде передана клієнту, повинна бути приведена до деякої проміжної формі, найбільш зручній для передачі. Оскільки клієнт може мати абсолютно будь-яку реалізацію (не обов'язково на платформі .NET), «звичайне» CLR-виключення не може бути передано, як є, натомість по мережі передаються тільки зазначені вище типи помилок.

На перший погляд такий підхід може здатися зовсім незручним і непрозорим, але це тільки перше враження. Так чи інакше, ми маємо справу з распределенностью. Крім таких причин, як витрати на сериализацию і десеріалізацію винятків, обсяг переданих даних та інше, основною причиною обмеження на типи переданих винятків є все ж суть побудови самих WCF-сервісів: сервіс повинен бути максимально незалежний, отказоустойчів, а про тонкощі виникають у ньому помилок знати ніхто не повинен, оскільки це просто небезпечно.

На щастя розробників, є generic-клас сервісного винятку FaultException <T>, завдяки якому існує можливість передавати деякі деталі виникла в сервісі помилки. Цей клас визначений таким чином:

public

class FaultException <TDetail>: FaultException

{

   public FaultException (TDetail detail);

   public FaultException (TDetail detail, string reason);

   // ...

}

TDetail - тип, який описує деталі помилки. Уточнюючий тип TDetail не обов'язково повинен бути представлений класом винятки, але він обов'язково повинен бути серіалізуем та / або визначений контрактом даних.

  Приклад 2 - Застосування класу винятку FaultException & lt; T & gt;

[ServiceContract]

interface ICalculator

{

   [OperationContract]

   double Divide (double number1, double number2);

   // ...

}

class Calculator: ICalculator

{

   public

double Divide (double number1, double number2)

   {

     if (number2 == 0)

     {

       DivideByZeroException exception =

         new DivideByZeroException ("Ділення на нуль!");

       throw

new FaultException <DivideByZeroException> (

exception, exception.Message);

     }

     return number1 / number2;

   }

   // ...

}

В якості підстави помилки можна передати, наприклад, повідомлення вихідного винятки, або екземпляр класу FaultReason, який зручно використовувати, коли потрібно локалізація.

Клієнт сервісу, наведеного в прикладі 2, може перехоплювати виключення наступним чином:

ICalculator service;

// ...

try

{

   service.Divide (7, 0);

}

catch (FaultException <DivideByZeroException>)

{

   // Обробка ділення на нуль

}

catch (FaultException)

{

   // Оброблення інших сервісних винятків

}

catch (Exception)

{

   // Обробка інших винятків

}

Контракт помилок сервісу

За замовчуванням будь виняток, що виникає при виконанні сервісної операції, доходить до клієнта у вигляді FaultException. Для того щоб сервіс міг передавати клієнту інші помилки, вони повинні бути частиною контракту. Контракт помилок WCF - це спосіб для сервісу вказати список помилок, які він може генерувати.

Контракт помилок є частиною контракту сервісу. Для вказівки винятку, яке може викликати сервісна операція, остання описується атрибутом FaultContractAttribute. Якщо повертатися до прикладу з калькулятором, то можна було зробити наступне (див. Приклад 3).

  Приклад 3 - Приклад визначення контракту помилок

[ServiceContract]

interface ICalculator

{

   [OperationContract]

   [FaultContract (typeof (DivideByZeroException))]

   double Divide (double number1, double number2);

   // ...

}

Визначивши контракт помилок, ви декларативно задаєте, які виключення може генерувати сервіс. Виходячи з цієї інформації, клієнтська сторона може правильно трактувати виникають на сервісі помилки і якось їх обробляти. З іншого боку, це накладає свої обмеження. Сервісна операція може генерувати виключення тільки тих типів, які визначені в контракті (не можна навіть використовувати підкласи визначеного в контракті типу винятки). Для зазначеного прикладу, генерація виключення на сервісі могла б виглядати так:

DivideByZeroException exception =

   new DivideByZeroException ("Ділення на нуль!");

throw

new FaultException <DivideByZeroException> (exception, exception.Message);

Всі контрактні помилки доходять до клієнта у вигляді FaultException <T>, де T - тип винятку, оголошеного в контракті помилок. Якщо сервіс викличе помилку, не визначену контрактом, вона дійде до клієнта, як FaultException.