## Использование контрактов сообщений и ошибок

## Контракты сообщений

Обычно при построении приложений Windows Communication Foundation (WCF) разработчики уделяют пристальное внимание структурам данных и вопросам сериализации; при этом им не требуется заниматься структурой сообщений, в которых передаются данные. Для таких приложений создание контрактов данных для параметров или возвращаемых значений представляет собой достаточно простую задачу. (Дополнительные сведения см. вЗадание передачи данных в контрактах служб.)

Тем не менее, иногда полный контроль над структурой сообщения SOAP столь же важен, сколь и контроль над его содержимым. Это особенно актуально, если важно обеспечить взаимодействие или контролировать определенные вопросы безопасности на уровне сообщения или части сообщения. В таких случаях можно создать контракт сообщения, позволяющий указать точную структуру для нужного сообщения SOAP.

В этом разделе рассматривается использование различных атрибутов контрактов сообщений для создания конкретного контракта сообщения для данной операции.

Использование контрактов сообщений в операциях

WCF поддерживает операции, моделируемые на основе либо стиля удаленного вызова процедур (RPC), либо стиля сообщений. В операции в стиле RPC можно использовать любой сериализуемый тип и иметь в своем распоряжении функции, доступные локальным вызовам, такие как множественные параметры и параметры ref и out. В этом стиле выбранная форма сериализации определяет структуру данных в соответствующих сообщениях, а сами сообщения для поддержки операции создаются средой выполнения WCF. Это позволяет разработчикам, не знакомым с протоколом SOAP и сообщениями SOAP, быстро и просто создавать и использовать приложения служб.

В приведенном ниже примере кода показана операция службы, смоделированная на основе стиля RPC.

[OperationContract]

public BankingTransactionResponse PostBankingTransaction(BankingTransaction bt);

Как правило, контракта данных достаточно для определения схемы для сообщений.Например, в предыдущем примере для большинства приложений будет достаточно, если BankingTransaction и BankingTransactionResponse будут иметь контракты данных для определения содержимого соответствующих сообщений SOAP.Дополнительные сведения о контрактах данных см. в разделе Использование контрактов данных.

Но время от времени необходимо точно контролировать процесс передачи структуры сообщения SOAP.Наиболее распространенным сценарием в таком случае является вставка настраиваемых заголовков SOAP.Другой часто используемый сценарий — определить свойства безопасности для заголовков и тела сообщения, т. е. решить, будут ли эти элементы снабжаться цифровой подписью и шифроваться.Наконец, для некоторых сторонних стеков SOAP необходимо, чтобы сообщения имели конкретный формат.Такого рода контроль обеспечивается операциями в стиле сообщений.

Операция в стиле сообщений имеет максимум один параметр и одно возвращаемое значение, причем оба типа являются типами сообщений, т. е. они сериализуются непосредственно в заданную структуру сообщения SOAP.Это может быть любой тип, отмеченный атрибутом MessageContractAttribute, или тип Message.В приведенном ниже примере кода показана операция, схожая с показанной выше операцией в стиле RPC, однако смоделированная на основе стиля сообщений.

Например, если и BankingTransaction, и BankingTransactionResponse представляют собой типы, являющиеся контрактами сообщений, код в следующих операциях является допустимым.

[OperationContract]

BankingTransactionResponse Process(BankingTransaction bt);

[OperationContract]

void Store(BankingTransaction bt);

[OperationContract]

BankingTransactionResponse GetResponse();

В то же время следующий код является недопустимым.

[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 каждой операции с участием типа с контрактом сообщения, не соответствующей одному из допустимых шаблонов, вызывается исключение.Конечно, на операции без участия типов с контрактами сообщений эти ограничения не распространяются.

Если тип имеет и контракт сообщения, и контракт данных, при использовании типа в операции во внимание принимается только его контракт сообщения.

Определение контрактов сообщений

Чтобы определить контракт сообщения для типа (т. е. определить сопоставление между типом и конвертом SOAP), примените к типу атрибут MessageContractAttribute. Затем примените атрибут MessageHeaderAttribute к тем членам типа, которые требуется превратить в заголовки SOAP, и примените атрибут MessageBodyMemberAttribute к тем членам, которые требуется превратить в части тела сообщения SOAP.

Приведенный ниже код представляет собой пример использования контракта сообщения.

[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 использовании этого типа в качестве параметра операции формируется следующий 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>

Обратите внимание, что operation и transactionDate выглядят, как заголовки SOAP, а тело SOAP состоит из элемента-оболочки BankingTransaction, содержащего элементы sourceAccount, targetAccount и amount.

Атрибуты MessageHeaderAttribute и MessageBodyMemberAttribute можно применять ко всем полям, свойствам и событиям, независимо от того, являются они открытыми, закрытыми, защищенными или внутренними.

Атрибут MessageContractAttribute позволяет указать атрибуты WrapperName и WrapperNamespace, от которых зависит имя элемента-оболочки в теле сообщения SOAP.

## Обработка ошибок в Windows Communication Foundation (WCF)

## Вступление

С обработкой ошибок сталкивался и сталкивается каждый день любой разработчик. И мы к этому привыкли, привыкли делать проверки на null и прочее, чтобы избежать наших «любимых» исключений или сформировать и сгенерировать собственное, более информативное исключение. Но когда речь идет о распределенных приложениях, тут следует учитывать их специфику.

В распределенной системе в качестве источника ошибок может выступать как клиентская, так и сервисная сторона.

Любая операция сервиса может вызвать исключение, о котором бы хотелось как-то оповестить клиента. На этом этапе и начинает работать специфика, о которой я говорил ранее, - специфика передачи ошибок между клиентом и сервисом. О ней и будет разговор.

## Концепция обработки ошибок.

Говоря о WCF, следует четко понимать основную концепцию сервисов. Сервис в WCF рассматривается, как нечто самодостаточное и совершенно независимое от своих клиентов. Помимо этого, сервис должен быть отказоустойчивым, иначе говоря, он не должен переходить в ошибочное состояние в случае какой-либо некритичной для него ситуации. Если в сервисе и происходит исключение, то клиенту не обязательно знать обо всех тонкостях этого исключения, ему важен факт ошибки и основная причина ее возникновения. Все тонкости важны только на этапе отладки, но в реальной жизни они не должны выходить за пределы сервиса. Для того чтобы как-то следить за жизнью сервиса, обычно пользуются журналом событий или определяют собственную реализацию логирования.

**Ошибки и исключения WCF**

В WCF работа hosting-процесса организована таким образом, что исключения, возникающие в WCF-сервисе, не нарушают работу этого процесса, а также работу других запущенных сервисов и клиентов, не имеющих отношения к этим исключениям. Как и в любой другой технологии удаленного взаимодействия, ошибка сервиса (**fault**) передается только тому клиенту, который стал ее инициатором.

Ясно, что для передачи ошибки, между клиентом и сервисом должна существовать некоторая договоренность, благодаря которой клиент однозначно воспринимает сервисную информацию и наоборот. Частью такой договоренности служат ошибки, о которых знает и клиентская и сервисная сторона.

Пытаясь обратиться к сервису, клиент, фактически, может столкнуться с тремя типами ошибок:

* Коммуникационные ошибки. Возникают, например, когда нет подключения к сети, когда указан неправильный адрес сервиса, когда не запущен host-процесс и т.д. Эти исключения определены на клиентской стороне классом исключения CommunicationException.
* Ошибки состояния канала. Связаны с состоянием прокси и каналов. Такой ошибкой может быть, например, попытка доступа к уже закрытому прокси, которая оканчивается исключением класса ObjectDisposedException, или, например, несоответствие в контракте и связывании уровня защиты безопасности.
* Ошибки запросов. Происходят при запросах к сервису.

С первыми двумя типами ошибок, как правило, все понятно, к тому же они обеспечиваются самой инфраструктурой WCF. Что касается ошибок, возникающих при обращении к сервису, – тут все намного сложнее и интереснее, поскольку существует множество причин возникновения ошибки, при выполнении сервисного запроса. В связи с этим, именно ошибки запросов, и представляют наибольший интерес, собственно им и посвящается данная статья.

Текущая реализация логической модели WCF на .NET-платформе такова, что исключения, вызванные на стороне сервиса, как правило, достигают клиента в виде FaultException.

public

class FaultException : CommunicationException

{ ... }

### Экземпляры сервиса и Singleton

Когда на сервисе происходит исключение, WCF не разрушает сервисный процесс, однако, может воздействовать на экземпляр удаленного объекта и возможность клиента продолжать использовать коммуникационный канал.

Если запрос вызывает непредвиденное исключение, то прокси клиента генерирует FaultException. Как правило, сервисные исключения переводят канал в ***состояние ошибки***, так что даже если отловить такое исключение на клиенте, последующие запросы вызовут CommunicationObjectFaultedException. Однако такое поведение справедливо не всегда, и все во многом зависит от используемого связывания. Так, например, если определить BasicHttpBinding, то после исключения канал остается работоспособным.

Насколько я знаю, многие разработчики часто делают на клиентской стороне singleton-ссылки на WCF-сервисы, чтобы каждый раз не создавать канал. И, несомненно, все, кто так делал, хоть раз сталкивались с вышеуказанной ошибкой. Поэтому следует быть очень внимательным при использовании singleton’ов и, в случае возникновения ошибки канала, заново пытаться получить доступ к сервису. Следующий пример, отчасти демонстрирует подобную логику, хотя и не претендует на звание лучшего.

[[http://rsdn.ru/images/ls2.gif](http://rsdn.ru/article/dotnet/FaultsWCF.xml)Пример 1 – Пример избегания ошибки коммуникации](http://rsdn.ru/article/dotnet/FaultsWCF.xml)

private ICalculatorContract \_сalculatorService;

public ICalculatorContract CalculatorService

{

get

{

if (\_сalculatorService == null)

{

// получение ссылки на удаленный объект

}

return \_сalculatorService;

}

}

public

void ClientMethod()

{

try

{

CalculatorService.Divide(9, 0);

}

catch (Exception)

{

// Коммуникационный канал разрушен

\_сalculatorService = null;

}

}

### Сервисные ошибки

Ошибки, возникающие на стороне сервиса, передается клиентской стороне по сети. Для их передачи в WCF используется протокол SOAP. Таким образом, ошибка, возникшая на сервисе, перед тем, как будет передана клиенту, должна быть приведена к некоторой промежуточной форме, наиболее удобной для передачи. Поскольку клиент может иметь абсолютно любую реализацию (не обязательно на платформе .NET), «обычное» CLR-исключение не может быть передано, как есть, вместо этого по сети передаются только указанные выше типы ошибок.

На первый взгляд такой подход может показаться совершенно неудобным и непрозрачным, но это только первое впечатление. Так или иначе, мы имеем дело с распределенностью. Помимо таких причин, как затраты на сериализацию и десериализацию исключений, объем передаваемых данных и прочее, основной причиной ограничения на типы передаваемых исключений является все же суть построения самих WCF-сервисов: сервис должен быть максимально независим, отказоустойчив, а о тонкостях возникающих в нем ошибок знать никто не должен, поскольку это просто небезопасно.

К счастью разработчиков, есть generic-класс сервисного исключения FaultException<T>, благодаря которому существует возможность передавать некоторые детали возникшей в сервисе ошибки. Этот класс определен следующим образом:

public

class FaultException<TDetail> : FaultException

{

public FaultException(TDetail detail);

public FaultException(TDetail detail, string reason);

// ...

}

TDetail – тип, который описывает детали ошибки. Уточняющий тип TDetail не обязательно должен быть представлен классом исключения, но *он обязательно должен быть сериализуем и/или определен контрактом данных.*

[[http://rsdn.ru/images/ls2.gif](http://rsdn.ru/article/dotnet/FaultsWCF.xml)Пример 2 – Применение класса исключения FaultException&lt;T&gt;](http://rsdn.ru/article/dotnet/FaultsWCF.xml)

[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("Деление на ноль!");

throw

new FaultException<DivideByZeroException>(

exception, exception.Message);

}

return number1 / number2;

}

// ...

}

В качестве основания ошибки можно передать, например, сообщение исходного исключения, или экземпляр класса FaultReason, который удобно использовать, когда требуется локализация.

Клиент сервиса, приведенного в примере 2, может перехватывать исключение следующим образом:

ICalculator service;

// ...

try

{

service.Divide(7, 0);

}

catch (FaultException<DivideByZeroException>)

{

// обработка деления на ноль

}

catch (FaultException)

{

// обработка прочих сервисных исключений

}

catch (Exception)

{

// обработка иных исключений

}

## Контракт ошибок сервиса

По умолчанию любое исключение, возникающее при выполнении сервисной операции, доходит до клиента в виде FaultException. Для того чтобы сервис мог передавать клиенту иные ошибки, они должны быть частью контракта. Контракт ошибок WCF – это способ для сервиса указать список ошибок, которые он может генерировать.

Контракт ошибок является частью контракта сервиса. Для указания исключения, которое может вызвать сервисная операция, последняя описывается атрибутом FaultContractAttribute. Если возвращаться к примеру с калькулятором, то можно было сделать следующее (см. пример 3).

[[http://rsdn.ru/images/ls2.gif](http://rsdn.ru/article/dotnet/FaultsWCF.xml)Пример 3 – Пример определения контракта ошибок](http://rsdn.ru/article/dotnet/FaultsWCF.xml)

|  |
| --- |
| [ServiceContract]  interface ICalculator  {  [OperationContract]  [FaultContract(typeof(DivideByZeroException))]  double Divide(double number1, double number2);  // ...  } |

Определив контракт ошибок, вы декларативно задаете, какие исключения может генерировать сервис. Исходя из этой информации, клиентская сторона может правильно трактовать возникающие на сервисе ошибки и как-то их обрабатывать. С другой стороны, это накладывает свои ограничения. Сервисная операция может генерировать исключения только тех типов, которые определены в контракте (нельзя даже использовать подклассы определенного в контракте типа исключения). Для указанного примера, генерация исключения на сервисе могла бы выглядеть так:

|  |
| --- |
| DivideByZeroException exception =  new DivideByZeroException("Деление на ноль!");  throw  new FaultException<DivideByZeroException>(exception, exception.Message); |

Все контрактные ошибки доходят до клиента в виде FaultException<T>, где T – тип исключения, объявленного в контракте ошибок. Если сервис вызовет ошибку, не определенную контрактом, она дойдет до клиента, как FaultException.