Bài 5

CLIENTS TRONG WINDOWS COMMUNICATION FOUNDATION

Mục lục

1 Kiến trúc của client trong Windows Communication Foundation 2
1.1 Các đối tượng phía client 3
1.1.1 ICommunicationObject 3
1.1.2 IExtensibleObject 4
1.2 Các kênh client 4
1.2.1 IClientChannel 4
1.2.2 IContextChannel 4
1.3 Các Channel Factory 5
1.3.1 Constructor (Cấu tử) của ChannelFactory 5
1.3.2 Các thuộc tính của ChannelFactory 5
1.3.3 Các phương thức của ChannelFactory 5
2 Các cách liên lạc của client 6
2.1 Một chiều (One-Way) 6
2.2 Yêu cầu-Trả lời (Request-Reply) 7
2.3 Song công (Duplex) 7
2.3.1 Phía dịch vụ 7
2.3.2 Phía client 9
2.4 Dị bộ (Asynchronous) 10
2.4.1 Phía dịch vụ 11
2.4.2 Phía client 12
3 Câu hỏi ôn tập 13
4 Tài liệu tham khảo 14

Trong các bài trước, chúng ta đã tập trung phần lớn vào việc cài đặt dịch vụ với Windows Communication Foundation. Hai bài trước chúng ta đã được giới thiệu về ba thành phần quan trọng để xây dựng dịch vụ WCF là address (địa chỉ), bindings, và contracts. Bài này chúng ta tập trung về phía client (máy khách, hay là các chương trình sử dụng dịch vụ WCF, ta gọi chung là client). Chúng ta sẽ được giới thiệu các mục sau:

- Kiến trúc của client
- Các cách liên lạc của client
- Viết mã nguồn cho client
- Định nghĩa các bindings and endpoints (điểm cuối) cho client

1 Kiến trúc của client trong Windows Communication Foundation

Một client trong Windows Communication Foundation là một chương trình sử dụng các chức năng cung cấp bởi một dịch vụ WCF. Chương trình client sẽ liên lạc với dịch vụ thông qua điểm cuối dịch vụ. Để làm được việc này client cần phải biết một số thông tin về dịch vụ như địa chỉ của điểm cuối, binding mà dịch vụ sử dụng, và contract dịch vụ. Các thành phần này đã được thảo luận ở các bài trước.

Một trong những thứ bạn có thể thấy trong kiến trúc của client là kênh thông tin được xây dựng dựa trên các cấu hình binding, những cấu hình này được quy định trong tệp tin cấu hình. Những thông tin cấu hình này chính là các thông tin đã được giới thiệu trong phần nói về bindings. Những bindings này cho phép client và dịch vụ liên lạc với nhau một cách hiệu quả.

Điểm thứ hai bạn có thể thấy là cài đặt của giao diện `IClientChannel`. Giao diện này định nghĩa các thao tác cho phép nhà phát triển điều khiển các chức năng của kênh, như đóng phiên làm việc của client và huỷ một kênh (để thu hồi tài nguyên). Nó đưa ra các phương thức và hàm của lớp `System.ServiceModel.ChannelFactory`.

Cuối cùng là bạn có thể thấy contract dịch vụ được tạo ra tự động, contract dịch vụ sẽ cung cấp tính năng chuyển lời gọi hàm ở phía client thành các bản tin đi, và chuyển các bản tin tới thành thông tin mà chương trình client có thể sử dụng dưới dạng giá trị trả về hay tham số đầu ra của hàm.

Các client liên lạc với điểm cuối dịch vụ thông qua một proxy, như hình dưới. Sự liên lạc được thực hiện thông qua một kênh. Sau khi proxy và kênh được tạo ra, client có thể truy xuất các phương thức có ở điểm cuối đó.

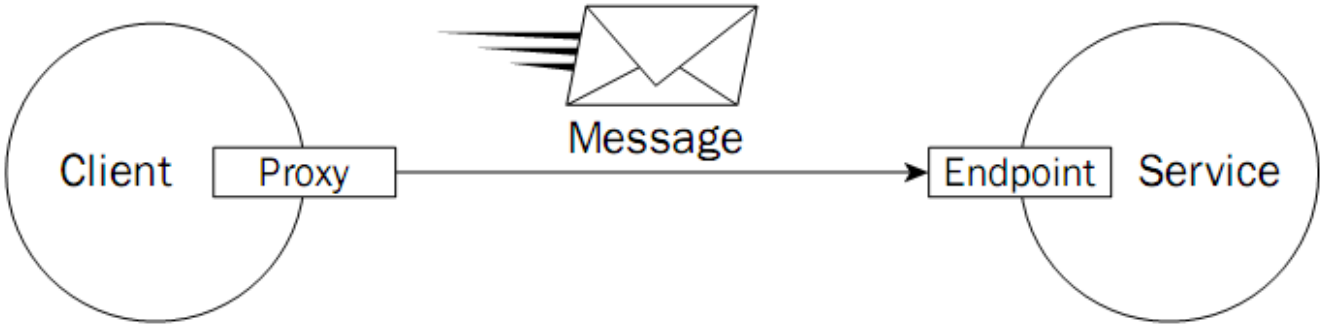

Figure 1 WCF Client Proxy

Có hai cách để tạo ra các client proxy:

- Cách đầu tiên là tạo proxy từ mã nguồn được tạo ra tự động từ thông tin siêu dữ liệu. Siêu dữ liệu này được cung cấp bởi dịch vụ. Việc tạo mã nguồn được thực hiện thông qua việc sử dụng công cụ Svcutil.exe, công cụ này đi kèm với bộ cài Visual Studio 2008 hoặc có thể lấy được ở trên trang web của Microsoft.
- Cách thứ hai là tạo ra proxy thông qua mã nguồn sử dụng đối tượng `ChannelFactory`. Đối tượng này được cung cấp từ lớp `System.ServiceModel.ChannelFactory`. Phương pháp này cho phép nhà phát triển khả năng điều khiển nhiều hơn, ví dụ như tạo các kênh mới từ một channel factory có sẵn.

1.1 Các đối tượng phía client

Một client trong Windows Communication Foundation phải chứa hai giao diện cơ sở là `ICommunicationObject` và `IExtensibleObject`.

1.1.1 ICommunicationObject

Giao diện `ICommunicationObject` là một trong những thành phần lõi để định nghĩa chức năng liên lạc. Nhiệm vụ của đối tượng này là định nghĩa contract cho trạng thái cơ bản của tất cả các đối tượng trong hệ thống; ví dụ, trạng thái đối tượng liên lạc đóng hay mở, hoặc trong trạng thái đang mở hoặc đang đóng. Những đối tượng này bao gồm các kênh, các đầu nghe (listeners), các dispatchers, factories, và chứa dịch vụ (service host).

Một chuyển trạng thái là việc chuyển từ trạng thái này sang trạng thái khác; ví dụ kênh liên lạc chuyển từ trạng thái "đang mở" sang trạng thái "mở".

Giao diện này định nghĩa các phương thức cho việc khởi tạo chuyển trạng thái:

- **Open**: Làm cho đối tượng liên lạc chuyển trạng thái từ "`Created`" sang "`Opened`".
- **Close**: Làm cho đối tượng liên lạc chuyển từ trạng thái hiện tại sang trang thái "`Opened`".
- **Abort**: Làm cho đối tượng liên lạc chuyển ngay lập tức từ trạng thái hiện tại sang trạng thái "`Closed`".

Ngoài các phương thức khởi tạo ở trên, giao diện `ICommunicationObject` còn định nghĩa các sự kiện để thông báo việc thay đổi trạng thái:

- **Opening**: Sự kiện này được kích hoạt khi đối tượng chuyển trạng thái từ `Created` sang `Opened`, sự kiện này xảy ra khi hàm `Open` hoặc `BeginOpen` được gọi.
- **Closing**: Sự kiện này được kích hoạt khi đối tượng chuyển trạng thái từ `Opened` sang `Closed`, sự kiện này xảy ra khi hàm `Close` hoặc `BeginClose` được gọi.
- **Opened**: Sự kiện này được kích hoạt khi đối tượng kết thúc việc chuyển trạng thái từ `Opening` sang `Opened`
- **Closed**: Sự kiện này được kích hoạt khi đối tượng kết thúc việc chuyển trạng thái từ `Closing` sang `Closed`
- **Faulted**: Sự kiện này được kích hoạt khi đối tượng chuyển sang trạng thái `Faulted`.

Ngoài ra còn có một tập phương thức dị bộ cho các phương thức `Open` và `Close`:

- **BeginOpen**: Bắt đầu thao tác dị bộ để mở một đối tượng liên lạc
- **BeginClose**: Bắt đầu thao tác dị bộ để đóng một đối tượng liên lạc
- **EndOpen**: Hoàn tất thao tác dị bộ để mở một đối tượng liên lạc

- **EndClose**: Hoàn tất thao tác dị bộ để đóng một đối tượng liên lạc

Giao diện `ICommunicationObject` có một thuộc tính trạng thái `State`, với kiểu là `CommunicationState`, thuộc tính này được dùng để chỉ ra trạng thái hiện tại của đối tượng.

Khi một đối tượng cài đặt `ICommunicationObject` được khởi tạo, trạng thái mặc định là `Created` chứ không phải là trạng thái `Opened`. Trong khi đang ở trạng thái `Created`, ta chỉ có thể thiết lập cấu hình cho đối tượng `ICommunicationObject` chứ không thể dùng nó để gửi hay nhận bản tin. Để có thể gửi hay nhận bản tin, đối tượng cần phải chuyển sang trạng thái `Opened`, nhưng khi ở trạng thái này, ta không thể thiết lập cấu hình cho nó nữa.

Để chuyển đối tượng sang trạng thái `Opened`, ta gọi hàm `Open`. Đối tượng sẽ giữ trạng thái này cho tới khi nó chuyển hoàn toàn sang trạng thái `Closed`. Hàm `Close` được dùng để chuyển đối tượng sang trạng thái `Closed`, tuy nhiên nó đợi cho các thao tác hiện tại của đối tượng hoàn tất mới thực hiện chuyển trạng thái. Hàm `Abort` thì không như vậy, nghĩa là những thao tác chưa hoàn tất sẽ bị huỷ bỏ, và đối tượng được chuyển ngay sang trạng thái `Closed`.

1.1.2 IExtensibleObject

Giao diện `IExtensibleObject` cung cấp cách mở rộng cho client. Trong WCF, cách mở rộng đối tượng được sử dụng để thêm các tính năng mới cho các lớp thực thi đã có, qua đó mở rộng các thành phần hiện tại cũng như thêm mới các trạng thái cho một đối tượng.

Giao diện này được đưa ra duy nhất một thuộc tính để cung cấp tính năng này, `Extensions`, với kiểu là `IExtensionCollection`. Thuộc tính này được sử dụng để trả về một tập hợp các đối tượng mở rộng có thể được sử dụng để mở rộng các lớp thực thi đã có.

1.2 Các kênh client

Các client trong Windows Communication Foundation chứa hai giao diện cơ bản cho kênh, giao diện `IClientChannel` và `IContextChannel`.

1.2.1 IClientChannel

Giao diện `IClientChannel` định nghĩa các thao tác được hỗ trợ bởi tất cả các kênh. Các kênh này được tạo ra bởi lời gọi hàm `ChannelFactory(TChannel).CreateChannel`. Giao diện này chứa các phương thức và thuộc tính có thể sử dụng để định nghĩa các hành xử kênh cho các yêu cầu bên ngoài và hành xử kênh cho yêu cầu/trả lời của chương trình client.

Ví dụ, thuộc tính `AllowInitializationUI` có thể được sử dụng để báo cho WCF mở một kênh mà không cần có lời gọi để mở nó.

1.2.2 IContextChannel

Giao diện IContextChannel định nghĩa trạng thái của phiên làm việc của một kênh. Những thông tin này bao gồm SessionId, phiên Input và phiên Output, cùng với các điểm cuối cục bộ và từ xa đang được sử dụng để liên lạc với client trong phiên làm việc. IContextChannel có các thuộc tính sau:

- **InputSession**: Trả về phiên đầu vào cho kênh
- **OutputSession**: Trả về phiên đầu ra cho kênh
- **LocalAddress**: Trả về điểm cuối cục bộ cho kênh
- **RemoteAddress**: Trả về địa chỉ từ xa nối với kênh
- **SessionId**: Trả về định danh của phiên hiện tại
- **OperationTimeout**: Trả về/thiết lập thời gian mà một thao tác phải hoàn thành. Nếu thao tác không hoàn thành trong thời gian quy định, sẽ có exception (ngoại lệ) xảy ra.

- **AllowOutputBatching:** Báo cho WCF lưu các bản tin trước khi chuyển chúng tới tầng vận chuyển.

1.3 Các Channel Factory

Việc hiểu các đối tượng client và đối tượng kênh client rất quan trọng do chúng đều sử dụng đối tượng ChannelFactory. Đối tượng ChannelFactory có trách nhiệm tạo và hỗ trợ các thực thi phía client.

Như đã nói ở phần đầu, bạn có thể tạo các client khi cần bằng cách sử dụng ChannelFactory hoặc sử dụng công cụ svcutil.exe. Công cụ svcutil tự động tạo ra đoạn mã nguồn làm việc với ChannelFactory, để tạo ra các kênh khi cần.

Đoạn mã nguồn sau mô tả việc sử dụng ChannelFactory để tạo ra một kênh cho mọt dịch vụ bằng cách đưa vào tên contract dịch vụ:

```
EndpointAddress ea = new EndpointAddress("tcp.net://localhost:8000/WCFService");
BasicHttpBinding bb = new BasicHttpBinding();
IServiceClass client = ChannelFactory<IServiceClass>.CreateChannel(bb, ea);

client.PlaceOrder(Val1);
```

Lớp `ChannelFactory` bao gồm các thành phần quan trọng là các cấu tử (constructors), các thuộc tính, và các phương thức.

1.3.1 Constructor (Cấu tử) của ChannelFactory

Lớp `ChannelFactory` có một cấu tử gọi là `ChannelFactory`. Nó được sử dụng để khởi tạo mới một thể hiện của lớp `ChannelFactory`. Đoạn mã để tạo ra một thể hiện `ChannelFactory` như sau:

```
IServiceClass client = ChannelFactory<IServiceClass>.CreateChannel(bb, ea);
```

1.3.2 Các thuộc tính của ChannelFactory

Lớp `ChannelFactory` có các thuộc tính sau:

- **Credentials**: Trả về các credential được sử dụng bởi client để liên lạc với điểm cuối dịch vụ thông qua kênh được tạo ra bởi factory
- **Endpoint**: Trả về điểm cuối mà kênh được tạo ra
- **State**: Trả về trạng thái hiện tại của đối tượng liên lạc.

Việc sử dụng credential yêu cầu có tham chiếu tới không gian tên `System.ServiceModel.Description`, thông qua việc sử dụng câu lệnh using:

```
using System.ServiceModel.Description;
```

Bạn có thể thiết lập credential cho client và dịch vụ cũng như cung cấp các credential cho việc xác thực ở phía proxy. Ví dụ sau đây biểu diễn cách cung cấp credential cho xác thực phía proxy khi tạo một kênh:

```
TCP.ServiceClassClient("WSHttpBinding_IServiceClass");
ChannelFactory<TCP.IServiceClass> factory = new
ChannelFactory<TCP.IServiceClass>("WSHttpBinding_IServiceClass");
TCP.IServiceClass channel = factory.CreateChannel();
ClientCredentials cc = new ClientCredentials();
cc.UserName.UserName = "scooter";
cc.UserName.Password = "wcfrocks";
factory.Credentials = cc;
```

1.3.3 Các phương thức của ChannelFactory

`ChannelFactory` có các phương thức sau:

- **Abort**: Chuyển ngay lập tức trạng thái của đối tượng liên lạc từ trạng thái hiện tại sang trạng thái `Closing`
- **BeginClose**: Bắt đầu thao tác dị bộ để đóng đối tượng liên lạc hiện tại
- **BeginOpen**: Bắt đầu thao tác dị bộ để mở đối tượng liên lạc
- **Close**: Chuyển trạng thái của đối tượng từ trạng thái hiện tại sang trạng thái `Closed`
- **EndClose**: Kết thúc thao tác đóng dị bộ của đối tượng liên lạc hiện tại
- **EndOpen**: Kết thúc thao tác mở dị bộ của đối tượng liên lạc hiện tại
- **Open**: Chuyển trạng thái của đối tượng từ trạng thái `Created` sang trạng thái `Opened`

Ví dụ sau tạo ra một kênh rồi mở kênh để thực hiện liên lạc:

```
WCFClientApp.TCP.ServiceClassClient("WSHttpBinding_IServiceClass");
ChannelFactory<TCP.IServiceClass> factory = new
ChannelFactory<TCP.IServiceClass>("WSHttpBinding_IServiceClass");
TCP.IServiceClass channel = factory.CreateChannel();
factory.Open();
channel.DoSomething();
```

Khi làm việc xong, bạn cần phải đóng channel factory bằng cách gọi hàm `Close` như sau

```
factory.Close();
```

2 Các cách liên lạc của client

2.1 Một chiều (One-Way)

Liên lạc một chiều là liên lạc chỉ theo một hướng duy nhất. Là hướng từ phía client tới dịch vụ. Không có trả lời từ phía dịch vụ và client hoàn toàn không trông mong nhận được phản hồi. Theo cách này, client gửi đi một bản tin và tiếp tục thao tác của mình.

Do không có phản hồi từ dịch vụ trong liên lạc một chiều, client sẽ không biết được liệu có lỗi xảy ra trong quá trình liên lạc hay không, và nó cũng hoàn toàn không biết liệu yêu cầu có thành công hay không. Để tạo ra dịch vụ cho liên lạc một chiều, ta đặt tham số `IsOneWay` của thuộc tính mô tả `OperationContract` là True. Điều này sẽ thông báo cho dịch vụ biết là không cần có phản hồi. Đoạn mã nguồn sau đây biểu diễn cách thiết lập một liên lạc một chiều, thao tác `AddPerson`, `EditPerson`, và `DeletePerson`:

```
    [ServiceContract]
    public interface IStaffInformation
    {
        [OperationContract]
        bool HasPerson(int personId);
        [OperationContract]
        Person GetPerson(int personId);
        [OperationContract]
        Person[] GetAll();

        [OperationContract(IsOneWay=true]
        void AddPerson(Person person);
        [OperationContract(IsOneWay=true]
        void EditPerson(int personId, Person person);
        [OperationContract(IsOneWay=true]
        void DeletePerson(int personId);
    }
```

2.2 Yêu cầu-Trả lời (Request-Reply)

Liên lạc theo kiểu yêu cầu-trả lời thực hiện như sau: client gửi đi một bản tin cho dịch vụ, nó sẽ chờ để nhận một phản hồi từ dịch vụ. Kiểu liên lạc này đồng thời còn có nghĩa là khi client gửi đi bản tin, nó sẽ không thực hiện thao tác khác cho tới khi nó nhận được một phản hồi từ phía dịch vụ.

Trong Windows Communication Foundation, có hai cách để quy định kiểu liên lạc yêu cầu-trả lời. Cách thứ nhất là đặt giá trị false cho tham số `IsOneWay` của `OperationContract`. Thực chất giá trị mặc định của tham số `IsOneWay` là false, do vậy cách thứ hai để quy định kiểu liên lạc yêu cầu-trả lời là không dùng tham số `IsOneWay` nữa. Do vậy các thao tác trên dịch vụ của WCF mặc định là liên lạc kiểu hỏi-trả lời.

Ví dụ đoạn mã nguồn sau đây khai báo 4 thao tác theo kiểu yêu cầu-trả lời: `HasPerson`, `GetPerson`, và `GetAll`:

```
[ServiceContract]
public interface IStaffInformation
{
    [OperationContract]
    bool HasPerson(int personId);
    [OperationContract]
    Person GetPerson(int personId);
    [OperationContract]
    Person[] GetAll();

    [OperationContract(IsOneWay=true]
    void AddPerson(Person person);
    [OperationContract(IsOneWay=true]
    void EditPerson(int personId, Person person);
    [OperationContract(IsOneWay=true]
    void DeletePerson(int personId);
}
```

Kiểu liên lạc hỏi-trả lời là kiểu liên lạc mặc định của WCF nên thông thường bạn cài đặt dịch vụ WCF hay sử dụng dịch vụ WCF, bạn liên lạc theo kiểu hỏi-trả lời. Các ví dụ ở các bài trước đều theo kiểu làm việc này. Hai kiểu liên lạc tiếp theo đây là liên lạc song công và liên lạc dị bộ bạn ít gặp hơn.

2.3 Song công (Duplex)

Liên lạc song công là khả năng mà cả client và dịch vụ đều có thể khởi tạo liên lạc, cũng như phản hồi các bản tin đến; nói cách khác đây là liên lạc hai chiều. Với liên lạc song công, dịch vụ không chỉ có thể trả lời các bản tin đến mà còn có thể khởi tạo liên lạc với client bằng cách gửi bản tin yêu cầu một phản hồi từ phía client.

Để cấu hình một liên lạc song công, cần có thay đổi từ cả hai phía: dịch vụ và client.

2.3.1 Phía dịch vụ

Để thực hiện liên lạc song công, phía dịch vụ cần có hai giao diện. Mục tiêu của giao diện thứ nhất là sử dụng trong liên lạc từ client tới dịch vụ, nghĩa là nó được sử dụng để nhận các bản tin từ phía client, giống như các ví dụ từ trước tới nay. Giao diện thứ hai, còn gọi là callback interface, được sử dụng trong liên lạc từ dịch vụ tới client để gửi bản tin từ phía dịch vụ tới client. Điều quan trọng là các thao tác ở trong các giao diện phải được định nghĩa là các thao tác một chiều.

Ví dụ sau mô tả cách định nghĩa một contract dịch vụ song công. Bước đầu là định nghĩa giao diện cho phía dịch vụ:

```
[ServiceContract(SessionMode = SessionMode.Required)]
public interface IDuplexService
```

```
    {
        [OperationContract(IsOneWay = true)]
        void Add(int bignumber);
        [OperationContract(IsOneWay = true)]
        void Subtract(int bignumber);
    }
```

Bước tiếp theo là định nghĩa callback interface. Giao diện này được dùng để gửi các kết quả tới client:

```
    public interface IDuplexServiceCallback
    {
        [OperationContract(IsOneWay = true)]
        void Calculate(int bignumber);
    }
```

Bước thứ 3 là đưa callback interface vào trong contract dịch vụ của giao diện đầu tiên. Việc này tạo ra mối liên kết giữa hai giao diện:

```
    [ServiceContract(SessionMode = SessionMode.Required,
        CallbackContract=typeof(IDuplexServiceCallback))]
    public interface IDuplexService
    {
        [OperationContract(IsOneWay = true)]
        void Add(int bignumber);
        [OperationContract(IsOneWay = true)]
        void Subtract(int bignumber);
    }
```

Cuối cùng là tạo ra một lớp để cài đặt contract dịch vụ song công. Để làm điều này, trước hết cần phải thêm một quy định hành xử cho dịch vụ (ServiceBehavior) vào lớp cài đặt bằng cách sử dụng thuộc tính mô tả `ServiceBehavior`, và đặt giá trị `PerSession` cho tham số `InstanceContextMode` cho thuộc tính này.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerSession)]
public class DuplexService : IDuplexService
{
    private int answer = 0;
    IDuplexServiceCallback Callback
    {
        get
        {
            return
OperationContext.Current.GetCallbackChannel<IDuplexServiceCallback>();
        }
    }

    public void Add(int bignumber)
    {
        answer += bignumber;
        Callback.Calculate(answer);
    }

    public void Subtract(int bignumber)
    {
        answer -= bignumber;
        Callback.Calculate(answer);
    }
}
```

2.3.2 Phía client

Để thực hiện liên lạc song công, phía client cũng phải nhận một phần trách nhiệm trong việc cài đặt, và cũng phải cài đặt một callback contract. Client thực hiện bằng cách cài đặt callback interface của duplex contract:

```
    public class ServiceCallback : IDuplexServiceCallback
    {
        private TextBox resultHolder;
        public ServiceCallback(TextBox textbox)
        {
            resultHolder = textbox;
        }

        #region IDuplexServiceCallback Members
        public void Calculate(int bignumber)
        {
            resultHolder.Text = bignumber.ToString();
        }
        #endregion
    }
```

Bước cuối cùng cho client là tạo ra cách để xử lý bản tin ở callback interface. Điều này có thể thực hiện bằng cách tạo ra một thể hiện của lớp InstanceContext trong lớp ở client:

```
InstanceContext ic = new InstanceContext(new ServiceCallback(resultText));
```

Sau khi có được một thể hiện của lớp InstanceContext, bạn có thể tạo client và thực hiện gọi hàm của dịch vụ:

```
client = new DuplexServiceClient(ic);
int value = Convert.ToInt32(inputText.Text);
client.Add(value);
```

Sau đây là ví dụ hoàn chỉnh phía client để thực hiện liên lạc song công:

```
using System;
using System.ServiceModel;
using System.Windows.Forms;
using DuplexClient.DService;

namespace DuplexClient
{
    public class ServiceCallback : IDuplexServiceCallback
    {
        private TextBox resultHolder;
        public ServiceCallback(TextBox textbox)
        {
            resultHolder = textbox;
        }

        #region IDuplexServiceCallback Members
        public void Calculate(int bignumber)
        {
            resultHolder.Text = bignumber.ToString();
        }
        #endregion
    }

    public partial class MainForm : Form
    {
        DuplexServiceClient client;
```

```
        public MainForm()
        {
            InitializeComponent();
            InstanceContext ic = new InstanceContext(new
ServiceCallback(resultText));
            client = new DuplexServiceClient(ic);
        }

        private void buttonAdd_Click(object sender, EventArgs e)
        {
            int value = Convert.ToInt32(inputText.Text);
            client.Add(value);
        }

        private void buttonSubtract_Click(object sender, EventArgs e)
        {
            int value = Convert.ToInt32(inputText.Text);
            client.Subtract(value);
        }
    }
}
```

Kết quả thực hiện như sau:

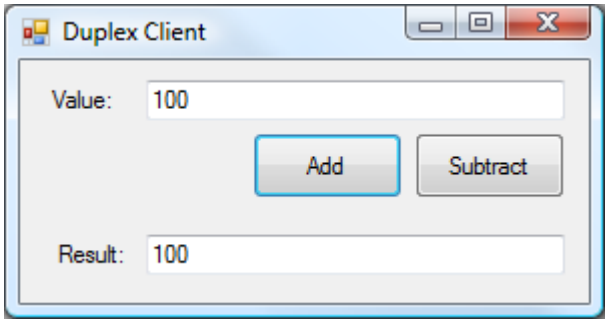

Figure 2 Thực hiện thao tác Add(100)

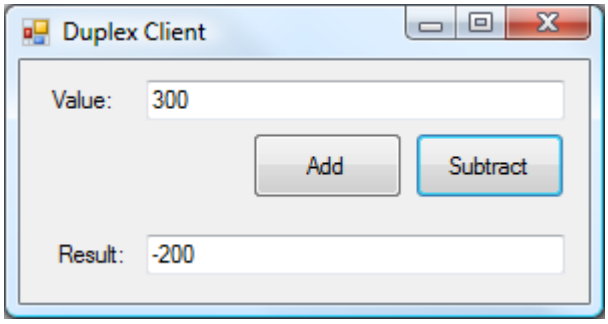

Figure 3 Thực hiện thao tác Subtract(300) từ giá trị hiện tại

2.4 Dị bộ (Asynchronous)

Liên lạc dị bộ được thực hiện bằng cách gọi các hàm dị bộ. Việc liên lạc dị bộ cho phép chương trình tiếp tục thực hiện những việc khác trong khi hàm được gọi vẫn đang thực hiện.

Cũng giống như trong liên lạc song công, các thao tác dị bộ yêu cầu một số thay đổi về phía client và dịch vụ.

2.4.1 Phía dịch vụ

Các thao tác dị bộ chia một thao tác ra thành hai thao tác rời nhau nhưng có liên quan tới nhau. Thao tác đầu tiên là thao tác Begin (bắt đầu), thao tác này được client gọi tới khi bắt đầu thao tác xử lý. Trong thao tác Begin, cần thêm 2 tham số cho thao tác này và giá trị trả về là một đối tượng `System.IAsyncResult`.

Thao tác thứ hai bắt đầu với End, nhận một tham số là đối tượng `System.IAsyncResult`, và trả về một giá trị. Thao tác End không cần phải có thuộc tính `OperationContract`.

Đoạn mã sau khai báo một dịch vụ với liên lạc dị bộ:

```
namespace AsyncService
{
    [ServiceContract]
    public interface IAsynchronousService
    {
        [OperationContract(AsyncPattern=true)]
        IAsyncResult BeginAdd(int val1, int val2, AsyncCallback cb, object state);

        int EndAdd(IAsyncResult result);
    }
}
```

Tiếp theo ta cài đặt một lớp cho khai báo dịch vụ ở trên:

```
    [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class AsynchronousService : IAsynchronousService
    {
        public IAsyncResult BeginAdd(int val1, int val2, AsyncCallback cb, object
state)
        {
            MathAsyncResult asyncResult = new MathAsyncResult(cb, state);
            asyncResult.Value1 = val1;
            asyncResult.Value2 = val2;

            ThreadPool.QueueUserWorkItem(new WaitCallback((Callback)), asyncResult);

            return asyncResult;
        }

        public int EndAdd(IAsyncResult asyncResult)
        {
            int result = 0;

            using (MathAsyncResult mathAsyncResult = asyncResult as MathAsyncResult)
            {
                mathAsyncResult.AsyncWaitHandle.WaitOne();
                result = mathAsyncResult.Result;
            }

            return result;
        }

        public int Add(int value1, int value2)
        {
            Thread.Sleep(1000);
            return value1 + value2;
        }

        private void Callback(object asyncResult)
```

```
        {
            MathAsyncResult mathAsyncResult = (MathAsyncResult)asyncResult;
            try
            {
                mathAsyncResult.Result = Add(
                    mathAsyncResult.Value1, mathAsyncResult.Value2);
            }
            finally
            {
                mathAsyncResult.OnCompleted();
            }
        }
    }
```

2.4.2 Phía client

Phía client của liên lạc dị bộ chỉ đơn giản là đưa vào tham số đúng và đảm bảo rằng giá trị trả về là kiểu IAsyncResult. Để truy nhập các thao tác dị bộ của dịch vụ, client bắt đầu với việc gọi hàm Begin, trong ví dụ sau là BeginAdd. Trong lời gọi hàm đó, client đưa vào một hàm callback để nhận giá trị trả về, trong ví dụ là callbackAdd. Khi hàm callback được gọi, client gọi hàm End để lấy ra kết quả, trong ví dụ là hàm EndAdd.

Mã nguồn hoàn chỉnh của phía client như sau:

```
using System;
using System.ServiceModel;
using System.Windows.Forms;
using AsyncClient.AService;

namespace AsyncClient
{
    public partial class MainForm : Form
    {
        IAsynchronousService client;
        public MainForm()
        {
            InitializeComponent();
            ChannelFactory<IAsynchronousService> factory =
                new ChannelFactory<IAsynchronousService>("AsynchronousService");
            factory.Open();
            client = factory.CreateChannel();
        }

        private void buttonAdd_Click(object sender, EventArgs e)
        {
            int val1 = Convert.ToInt32(inputText1.Text);
            int val2 = Convert.ToInt32(inputText2.Text);

            client.BeginAdd(val1, val2, new AsyncCallback(OnEndAdd), null);
            resultText.Text = "Calculating ...";
        }

        public void OnEndAdd(IAsyncResult asyncResult)
        {
            this.Invoke(
                new MethodInvoker(delegate()
                    {
                        resultText.Text = client.EndAdd(asyncResult).ToString();
                    }));
        }
```

```
    }
}
```

Kết quả thực hiện liên lạc dị bộ như sau:

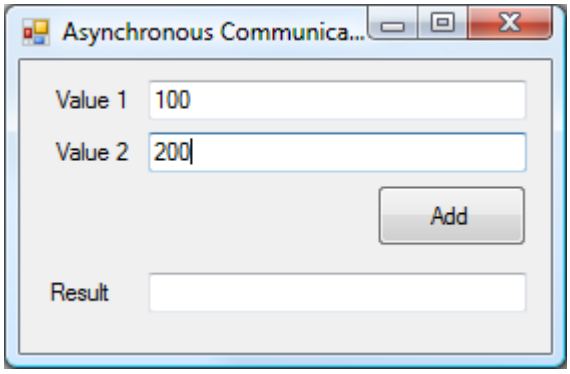

Figure 4 Nhập dữ liệu cho client

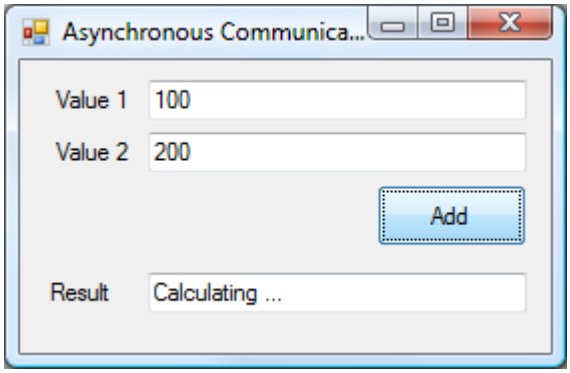

Figure 5 Bắt đầu thực hiện hàm Add

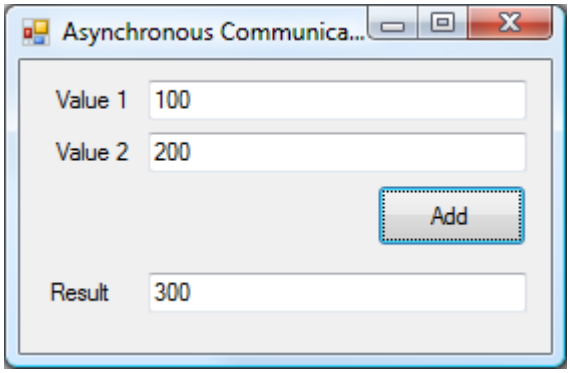

Figure 6 Kết thúc thực hiện hàm Add, lấy kết quả trả về

3 Câu hỏi ôn tập

1. Liệt kê các kiểu liên lạc có trong WCF

Có các kiểu liên lạc sau:

- Liên lạc một chiều từ client tới dịch vụ
- Liên lạc yêu cầu-trả lời
- Liên lạc song công
- Liên lạc dị bộ

4 Tài liệu tham khảo

1. Building Clients (URL: http://msdn.microsoft.com/en-us/library/ms730825.aspx)
2. Synchronous and Asynchronous Operations (URL: http://msdn.microsoft.com/en-us/library/ms734701.aspx)
3. How to: Call WCF Service Operations Asynchronously (URL: http://msdn.microsoft.com/en-us/library/ms730059.aspx)