[**WCF学习笔记——WCF基础**](http://www.cnblogs.com/MeteorSeed/archive/2012/04/24/2399455.html)

目录

[一 WCF与SOA](http://www.cnblogs.com/MeteorSeed/archive/2012/04/24/2399455.html#SOA)

[二 WCF模型](http://www.cnblogs.com/MeteorSeed/archive/2012/04/24/2399455.html#mx)

[三 WCF体系架构](http://www.cnblogs.com/MeteorSeed/archive/2012/04/24/2399455.html#txjg)

[四 WCF程序的开发步骤](http://www.cnblogs.com/MeteorSeed/archive/2012/04/24/2399455.html#four)

[五 编码规范](http://www.cnblogs.com/MeteorSeed/archive/2012/04/24/2399455.html#jc)

**一 WCF与SOA**

　　SOA是一种通过为所有软件提供服务外观，并将这些服务的WSDL集中发布到一个地方的一种组织企业软件的方法。它通过使用明确定义的接口通过跨越边界传递消息来让多个自治的服务协同工作。SOA的真正价值是——允许开发者从代码中抽取出公共基础功能的实现，更多地关注业务逻辑和需要的功能特性。在开发SOA应用程序时，我们能够实现服务代码与客户端使用技术与平台的解耦，也与并发管理、事务传播和管理以及通信可靠性、协议和模式无关。

　　SOA的4个主要设计原则以及在WCF中的具现如下：

* **边界是明确的**SOA系统中的每个服务都必须被限定在某个明确的边界之内。服务边界指的就是服务的公共接口与其内部实现之间有清晰的界线。WCF服务的功能是通过定义明确的接口进行表达的，外部调用者和WCF服务通信的唯一方式就是通过这个接口，客户端无法知道服务的实现细节，以达到信息隐藏和解耦的目的。
* **服务是自治的**自治的服务对于版本问题、部署问题和安装问题来说应该是独立的不会影响整个SOA系统。服务的运行不需要依赖任何外部服务或者组件。我们为WCF扩展功能，只能通过编写新的接口来实现。
* **采用标准的契约定义和通信协议**服务通过契约而不是实现进行通信，服务契约定义和通信协议都是行业标准。WCF对常用协议和主流编码格式提供支持。
* **服务是自解释的**服务的内容必须是自解释的，服务必须以某种方式告诉客户端服务的功能。WCF提供了强类型的契约，并会根据绑定来生成WSDL文档。

　　以上原则比较抽象，为了更好的设计SOA程序，应该遵守以下原则：

* **服务是安全的** 服务与客户端必须使用安全的通信。
* **服务在系统中应保持一致的状态** 执行客户端请求时，禁止进行部分替换条件，即系统提供服务给客户端调用后必须保持一致的状态，如果有错误发生，系统状态不应该部分受到影响，必须被恢复到一致的状态。
* **服务是线程安全的** 服务必须设计为线程安全，才能维持多线程的并发访问。
* **服务是可靠的** 客户端以确定的方式获知服务是否接收了消息。
* **服务是健壮的** 服务与它的错误分离能够防止错误影响服务本身或其它服务。

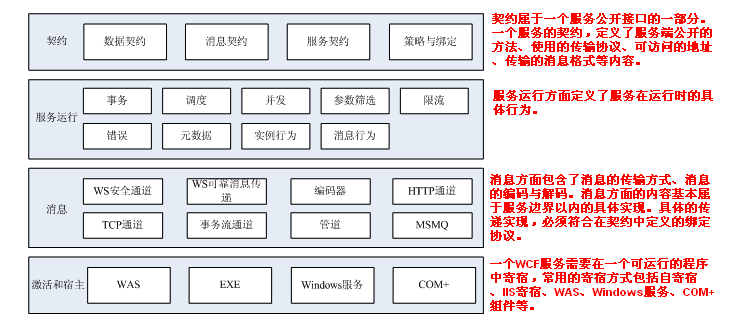
　　可选原则如下：

* **服务是平台无关的** 服务应该能够被任意的客户端调用，而不用考虑客户端技术。
* **服务的规模不可变** 不管客户端有多少，也不管服务的承载是多少，服务代码都应该相同。
* **服务是可用的** 服务总是能够接受客户端的请求，而不会因此停止。
* **服务是及时响应的和受限的** 服务开始处理客户端的请求时，不能让客户端等待的太久。服务执行的任意操作应该尽可能短，不能消耗太多时间去处理客户端的请求。

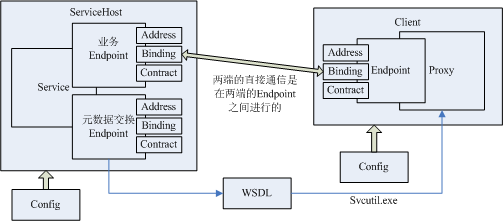
　　微软的WCF框架中的基础设施为我们解决了以上原则中的多数，使我们可以将关注点重新回归的业务逻辑层上。面向服务技术展示出的魅力，促使越来越多的开发者投入其中。

**二 WCF模型**

　　WCF的体系架构如下：



　　WCF的客户端与服务端模型如下：



**1 地址**

　　WCF的每一个服务都有唯一的地址。地址包括服务位置和传输协议（传输样式）。服务位置包括目标机器名、站点、网络、端口、管道或队列，以及一个可选的特定路径或者URI。

　　地址常用格式为：[基地址]/[可选的URI]，如“net.tcp://localhost:8081/MyService”。

　　基地址常用格式为：[传输协议]://[机器名或域名][:可选端口]，如“net.tcp://localhost:8081”。

　　WCF支持多种传输样式：

* **HTTP** 采用http、https协议进行传输，默认端口号80。
* **TCP** 采用net.tcp协议进行传输，默认地址为808。
* **Peer** **network（对等网）** 采用net.p2p进行传输，采用Windows的对等网传输机制。
* **IPC（内部进程通信）** 采用net.pipe进行传输，使用Windows的命名管道机制，只能接收来自同一台机器的调用，且每台机器只能打开一个命名管道。
* **MSMQ** 采用net.msmq进行传输，使用Windows的MSMQ机制，必须指定队列名，如果是处理私有队列，则必须指定队列类型。

**2 绑定**

绑定将通信模式与交互方式直接的组合进行规范，将这些通信特征合理地组合在一起。一个绑定封装了诸如传输协议、消息编码、可靠性、安全性、事务传播以及互操作性等相关选项的组合，使得它们保持一致。

　　WCF定义了六种常用的绑定：

* **基本绑定** 由BasicHttpBinding类提供，将WCF服务公开为Web服务。
* **TCP绑定** 由NetTcpBinding类提供，使用TCP协议通信，支持多种特性，包括可靠性、事务性、安全性及WCF之间通信的优化，缺点是客户端必须使用WCF。
* **IPC绑定** 由NetNamedPipeBinding类提供，使用命名管道为同一机器的通信进行传输，支持的特性与TCP绑定类似，是性能和安全性最佳的绑定。
* **Web服务绑定** 由WSHttpBinding类提供，WS绑定使用HTTP或HTTPS进行传输，提供了诸如可靠性、事务性与安全性在内的多种特性，这些特性遵循WS-\*标准，该绑定被设计用来与支持WS-\*标准的系统进行互操作。
* **WS双向绑定** 由WSDualHttpBinding类提供，支持双向通信。
* **MSMQ绑定** 由NetMsmqBinding类提供，使用MSMQ进行传输。

　　常用绑定的传输协议与编码格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 名字 | 传输协议 | 编码格式 | 互操作性 |
| BasicHttpBinding | HTTP/HTTPS | Text,MTOM | yes |
| NetTcpBinding | TCP | Binary | no |
| NetNamedPipeBinding | IPC | Binary | no |
| WSHttpBinding | HTTP/HTTPS | Text,MTOM | yes |
| WSDualHttpBinding | HTTP | Text,MTOM | no |
| NetMsmqBinding | MSMQ | Binary | no |

　　注意：TCP、IPC和MSMQ绑定使用的二进制编码器是WCF专有的，不要试图为其编写针对其他平台的自定义解析器。

**3 契约**

　　WCF的所有服务都公开为契约，契约与平台无关，是描述服务功能的标准方式。WCF包含4类契约：

* **服务契约** 描述了客户端能够执行的服务操作。
* **数据契约** 定义了与服务交互的数据类型。
* **错误契约** 定义了服务抛出的错误，以及服务处理错误和传递错误到客户端的方式。
* **消息契约**消息契约允许服务直接与消息交互，用于定制专有的消息格式，也意味着要自己的应用程序上下文，因为会增加代码的复杂度，所以不属于常见用法。

**4 终结点**

　　终结点是WCF进行通信的唯一手段，ChannelFactory<T>本质上是通过制定的终结点创建用于进行服务调用的服务代理。终结点在WCF体系中通过Systtm.ServiceModel.Description.ServiceEndpoint类型表示，其包含三个核心属性——地址、绑定、契约。终结点是用来发送和接收消息的构造，终结点是真正意义上的接口，它包含了一个对象接口所需的全部信息。每个服务至少必须公开一个业务终结点，每个终结点有且只能拥有一个契约。服务上的所有终结点都包含了唯一的地址，而一个单独的服务则可以公开多个终结点。这些结点可以使用相同或不同的绑定，公开相同或不同的契约。

**5 元数据**

服务的元数据描述服务的特征，外部实体需要了解这些特征以便与该服务进行通信。服务的元数据包括XML、架构文档（用于定义服务的数据协定）和WSDL文档（用于描述服务的方法）。启用元数据后，WCF通过检查服务及其终结点自动生成服务的元数据。

**6 上下文**

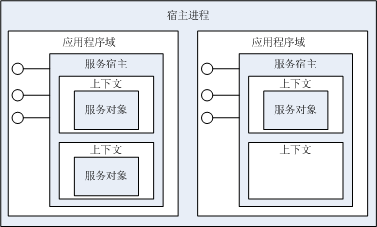
　　WCF上下文将服务宿主与公开本地CLR类型为服务的上下文组合在一起，上下文是服务实例最核心的执行范围，上下文可以为空，即不包含任何服务实例。

**7 宿主**

WCF服务类不能凭空存在，每个WCF服务类必须托管在某个宿主进程中。单个宿主进程可以托管多个服务，而相同的服务类型也可以托管在多个宿主进程中，如果服务与客户端驻留在相同的进程中，则称为进程内托管。常用宿主如下：

* Web站点
* Windows窗体应用程序
* Windows服务
* Windows激活服务（WAS）

　　WCF宿主体系结构如下：



 　　每个.NET宿主进程都包含了多个应用程序域，每个应用程序域包含零到多个宿主实例，每个服务宿主实例专门对于于一个特殊的服务类型。因此，创建一个宿主实例，实际上就是为对应于基地址的宿主机器的类型注册一个包含了所有终结点的服务宿主实例。每个服务宿主实例拥有零到多个上下文。一个上下文可以与零个或一个服务实例关联。

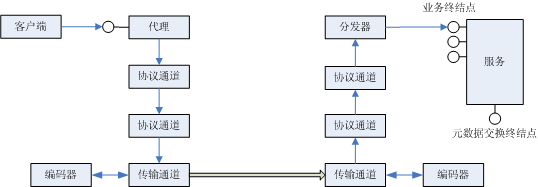
**8 代理**

　　WCF不允许客户端直接与服务交互，客户端使用代理将调用转发给服务。即使对象是本地的，WCF仍然使用远程编程模型的实例化方式，并且使用代理。

**三 WCF体系架构**

**（一） 通道栈模型**

　　WCF通道模型：



　　无论交互的另一方的具体位置在哪里，WCF都会为消息的发送和接收建立一套完整的消息管道，这个消息管道被成为通道栈。通道栈中的每一个通道组件都有机会对消息进行处理，而整个通道栈是可编辑且可插入的，这就确保WCF的通道模型具有相当大的灵活性。另外，WCF通道模型是完全和上层程序隔离的，任何一个服务/客户端都可以轻松配置到不同的通道模型上去。通道模型可以被分为两部分——协议通道和传输通道。一个通道栈可以拥有任意多个协议通道，但一般只拥有一个传输通道。传输通道负责把消息进行编码并且发送到远端，编码时需要使用调用栈的编码器。一般而言，协议通道负责维护消息的非业务逻辑功能，包括：事务、日志、可靠消息、安全性等。开发者可以自定义协议通道并插入到通道栈中。在一个通道栈中，必须包含至少一个传输通道和编码器，传输通道负责把消息编码和发送。（传输通道会尝试从BindingContext对象中查找编码器，如果没有找到则会使用默认的编码器，在完成消息的编码之后，传输通道负责把消息发送到远端，不同的传输通道将使用不同的传输协议，如HTTP、TCP、IPC等。）

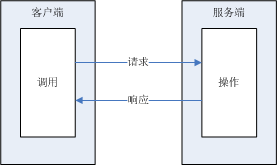
　　WCF体系体系架构是基于拦截机制的，通过代理与客户端的交互意味着WCF总是处于服务与客户端之间，拦截所有的调用，执行调用前和调用后的处理。当代理将调用栈帧序列化到消息中，并将消息通过通道链向下传递时，WCF就开始执行拦截。通道相当于一个拦截器，目的在于执行一个特定的任务。每个客户端通道都会执行消息的调用前处理。链的组成与结构主要依赖于绑定。客户端的最后一个通道是传输通道，根据配置的传输方式发送消息给宿主。在宿主端，消息同样通过通道链进行传输，它会对消息执行宿主端的调用前处理。宿主端的第一个通道是传输通道，接收传输过来的消息。随后的通道执行不同的任务。宿主端的最后一个通道负责将消息传递给分发器。分发器将消息转换到一个栈帧，并调用服务实例。事实上，服务会被本地客户端——分发器调用。客户端与服务端的拦截器确保了它们能够获得运行时环境，以便于它们执行正确的操作。

　　服务实例会执行调用，然后将控制权返回给分发器。分发器负责将返回值以及错误信息转换为一条返回消息。分发器获得控制权，执行的过程刚好相反：分发器通过宿主端通道传递消息，执行调用后的处理，如管理事务、加密等。为了执行客户端调用后的处理，包括解密、解码、提交或取消事务等任务，传输通道会将返回消息发送到客户端通道。最后一个通道将消息传递给代理。代理将返回消息转化到栈帧，然后将控制权返回给客户端。

**（二） 消息交换模式**

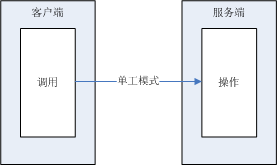
　　WCF定义的6种消息交换模式：

**1  请求-响应模式**



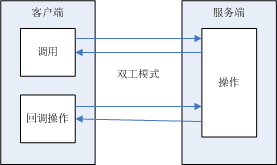
　　客户端向服务器发送一个请求，等待服务器的响应，等待时间是一个预定义的时间，是最常用的模式。WCF中通过在接口上定义[ServiceContact]和[OperationContract]来表示请求和响应的细节。

**2 单工消息处理模式**



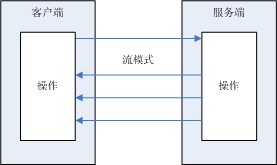
 　　只按一个方向把消息从客户端发送到服务器，服务器只处理消息，但不返回处理结果。这种模式最大的价值在于能够实现异步通信以及保证消息传送环境的能力，利用单工模式，可以通过MSMQ来发送消息。WCF通过把[OperationContract]的IsOneWay设置为True，来实现单工模式。

**3 双工消息处理模式**



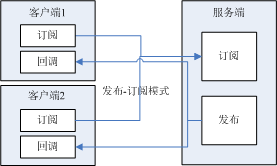
 　　客户端和服务端相互调用，客户端和服务端的角色也在不断交替。为了完成回调，服务端必须知道回调操作的方法签名，而且该回调操作将同时出现在客户端和服务端中。为此，在WCF中实现双工模式，需要在它的绑定信息中说明，在客户端实现的方法签名需要在服务层定义，同时需要这些方法在客户端实现，这样服务端才能调用他们。

**4 流模式**



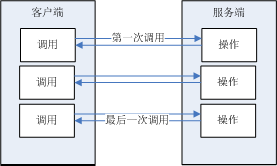
　　客户端发起一个请求，请求一个非常大的数据，服务把数据分割为小块，并把这些数据块按使用顺序逐个地发送给客户端。在这种模式下，一个数据请求紧跟着多个响应，每个响应都包含了全部数据的一个子集，由发送者在最后的消息中标识数据流的结束，以告知客户端不需要继续等待更多的数据。

**5 发布-订阅模式**



　　发布-订阅模式中，发布者不关心是否有订阅者，它只关心从自己的系统中发送数据。这种模式表示了一种驱动方法，即发布者在客户端应用程序中启动一个事件，客户端针对这些事件作出响应。WCF不直接支持发布-订阅模式，而是利用与双工模式相同的构造来实现发布-订阅模式。发布-订阅模式的实现通过一个订阅服务，此服务在内存中保存订阅者回调通道的引用。当服务需要向所有订阅者发送数据时，它便循环地访问这些订阅者，并调用一个方法来发送数据。这里没有使用轮询的方法，是因为订阅者起宿主的作用，它在等待被服务调用的方法。

**6 隐式顺序调用模式**



　　在该模式中，客户端以某个顺序调用服务端的多个操作。隐式顺序调用模式允许定义逻辑顺序中最后调用的方法，此后客户端将不能再调用其它方法。WCF使用[OperationContract]定义操作的调用顺序，把方法的IsInitiating设置为false，可以起动一个会话；把IsTerminating设置为true，表示调用这个方法以结束会话。（另一种实现方式，是借助工作流。）

**（三） 通道形状**

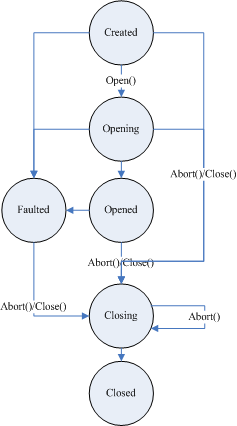
　　WCF中通过通道形状来实现具体的消息交换模式，通道形状则是只继承自IChannel的一组接口，WCF会根据上层协议来自动选取需要的通道形状。消息交换模式与通信协议是密切相关的，通过在传输通道上层添加特殊的协议通道，可以强制使用某种传输通道不支持的消息交换模式。（OneWayBindingElement单工模式和CompositeDuplexElement双工模式）

**（四） 通道管理器**

　　WCF中实现了两类通道管理器，分别实现IChannelListener<T>和IChannelFactory<T>。IChannelListener<T>负责接收端的信息交换控制工作，负责侦听消息、建立通道栈，并且通过通道栈顶层通道的引用，它可以独立于它的通道栈而关闭。ServiceHost内部就是使用了IChannelListener<T>来实现通道的管理。IChannelListener<T>负责在发送端控制消息的发送以及创建并管理通道栈，并且负责关闭通道栈，一般使用ClientBase<T>来代替直接使用IChannelFactory<T>。

**（五）ICommunicationObject**

　　WCF的ICommunicationObject定义了状态机制的统一模型。基本所有的通信组件，包括通道和通道管理器都实现了ICommunicationObject。ICommunicationObject用于状态管理，其定义了改变状态的方法、状态改变触发的事件已经当前状态的获取。ICommunicationObject的State为CommunicationState枚举类型，用于表示“状态”，有6种可用状态，分别为：Created（已创建），Opening（打开中），Opened（已打开），Closing（关闭中），Closed（已关闭），Faulted（错误），其状态变化顺序如下：



 　　由上图可见，状态机的改变是只进的，状态的改变是不可逆的。

**四 WCF程序的开发步骤**

1. 使用.NET接口定义契约。
2. 编写实现契约的服务类。
3. 添加相应特性对WCF的行为进行控制。
4. 开发宿主程序承载服务。
5. 设置服务端配置文件，定制个性化需要。
6. 客户端添加服务引用，生成代理类。
7. 设置客户端配置文件，定制个性化需要。

**五 编码规范**

* 应该将服务代码放入到类库中，而不是放到宿主程序中。
* 不要为服务类型提供带参数的构造器，除非托管的服务是明确的单例服务。
* 在相关绑定中启用可靠性。
* 为契约提供有意义的命名空间。
* 对于运行在Windows XP或Windows Server 2003上的局域网应用程序，最好选用自托管，而不是IIS托管。
* 在Windows Vista和Windows Server 2008或更高版本中，最好使用WAS托管，而不是自托管。
* 启用元数据交换。
* 为客户端配置文件中的所有终结点命名。
* 尽量自己编写配置文件。
* 不要复制代理的代码，可以将代理分解到单独的类库中。
* 及时关闭和释放代理。