这部分将介绍一些相对深入的知识点，包括通过并发限流来保证服务的可用性，通过可靠会话机制保证会话信息的可靠性，通过队列服务来解耦客户端和服务端，提高系统的可服务数量并可以起到削峰的作用，最后还会对之前的事务知识做一定补充。

对于WCF服务来说，其寄宿在一个资源有限的环境中，为了实现服务性能最大化，需要提高其吞吐量即服务的并发性。然而在不进行流量控制的情况下，并发量过多，会使整个服务由于资源耗尽而崩溃。因此为相对平衡的并发数和系统可用性，需要设计一个闸门(Throttling)控制并发的数量。

由于WCF的并发出来属于服务自身的行为，因此通过服务行为对其进行控制，ServiceBehaviorAttribute（之后回调的CallbackBehaviorAttribute与此类似）中的ConcurrencyModel属性定义了Single、Reentrant和Multiple三种典型的并发模式，Single表示一个实例上下文InstanceContext只能用于单一请求的处理，Reentrant也表示一个实力上下文某一时刻只能用于对单一请求的处理，但若涉及回调，也可以用于其他服务调用请求的处理，Multiple表示一个实力上下文可以同时处理多个服务请求。此外，当将服务行为的ReleaseServiceInstanceOnTransactionComplete属性设置为true，其同步模式必须是Single，因为不能在其他请求还在处理中时释放实例。此外，可以通过设置UserSynchronizationContext属性为false来禁止并发操作。

可以通过设置System.ServiceModel.Description.ServiceThrottlingBehavior的相关属性来限流，包括：

MaxConcurrentCalls:当前ServiceHost能够处理的最大并发消息数量，默认为单核16。

MaxConcurrentInstances：当前ServiceHost允许存在的服务实例上下文的数量，默认为116。

MaxConcurrentSessions：当亲ServiceHost允许的最大并发会话数量，默认为100。

实际上WCF在其内部构建一个专门的内部组件FlowThrottle，其包含一个Capacity属性，表示最大流量，以及一个队列和计数器。ServiceThrottle三个流量限制器就像是设置在信道分发器中的三道闸门，第一道限制并发会话的流量，第二道限制并发请求的数量，第三道限制并发实例上下文的数量。**为什么是这样的判断顺序，仍然有疑问**？

作为一个通信基础平台，需要保持消息的可靠性，由于网络环境限制，网络层往往不能保证消息的有效交付，因此需要在应用层通过可靠会话机制来实现端到端的可靠信息传输。对于可能传输来说，常见的问题包括重复消息和无序消息的问题。说到这，不得不提TCP协议，其就是用在解决IP层消息传输不可能和无连接问题的，其通过3此握手建立长连接，通过消息确认和超时重传机制来保证消息的可靠性。那么它与现在要提及的WCF中RM有什么区别呢？主要包含以下4点：WCF可靠消息是基于SOAP的，而TCP是基于报文段的；其与传输协议无关，并不限于TCP协议；并没有具体传输会话限制，可以跨越多个传输连接或会话；TCP在当前连接内提供端到端可靠传输，而WS-RM提供两个SOAP终结点见的可靠传输，不受传输连接限制。

在实质上，WS-RM可靠传输的原理与TCP的活动窗口机制相似，其首先也会创建连接（CreateSequence），服务端会返回去一个Identifier，之后开始数据传输，Sequence的头部包含Identifier和MessageNo，后者用于识别顺序，此外还包含AcksTo、Expires、Offer等参数，最后会TerminateSequence。主要注意的是，在请求-回复模式的通信中，为了减少通信量，消息将包含2部分，一部分是对上条消息的回复（SequenceAcknowledgement），一部分是新的消息，常见的可靠会话配置如下所示。

|  |
| --- |
| <system.serviceModel>  <bindings>  <customBinding>  <binding name="orderedDelivery">  <reliableSession ordered="true"/>  <binaryMessageEncoding>  <readerQuotas maxArrayLength="2147483647"/>  </binaryMessageEncoding>  <tcpTransport maxBufferSize="2147483647" maxReceivedMessageSize="2147483647"></tcpTransport>  </binding>  </customBinding>  </bindings>  </system.serviceModel> |

最后通过一个表格，简单的介绍一下可靠会话的最佳实践。

|  |  |
| --- | --- |
| 方式 | 诠释 |
| 设置MaxTransferWindowSize | 用于指示传输窗口可以保存多少信息，默认为8条 |
| 有效使用网络 | 如果网络延迟大，可以考虑增大传输窗口，已达到提升网络使用率的目的 |
| 满负荷运行服务 | 使用缓冲区可以提高服务的可用性，推荐发送方和接受方使用相同的MaxTransferWindowSize |
| 启动流控制 | 为了确保发送方和接收方步调一致，推荐将FlowControlEnabled设置为true |
| 设置MaxPendingChannels | 一旦可靠传输建立，PendingChannel就会加1，因此需要为MaxPendingChannels设置一个合适的值，太低服务利用率不高，太高会影响到工作集的状态，默认为4 |
| 可靠会话和宿主 | 可靠会话是有状态的，有AppDomain维护，在双工场景下，默认每个客户端需要两条HTTP连接，因此可能出现资源使用过量死锁的情况，这是需要如下设置来处理。  <system.net>  <connectionManagement><add name = “\*” maxconnection=”XX”/>  </connectionManagement >  <system.net> |

这部分将介绍微软MSMQ，虽然很老了，但仍然需要做一个基础的了解，当然了，现在比较推荐RabbitMQ的开源队列框架，不管怎么说在互联网场景下，消息队列是解决峰谷平衡的目前最好解决方案。其类似于人类传统的书信通信，寄件人只需要收件人地址、邮编和姓名信息，仍然将信件放入邮筒即可，而收件人只要定时检查收信即可。为了使用消息队列，首先通过Windows功能安装MSMQ，包括AD服务集成、HTTP支持、触发器、多播支持和DCOM代理等组件。消息队列的信息将保存在%Windir%\System32\msmq\storage中，常见的，队列分为如下几种队列。

普通队列：具体应用创建，基于业务的队列，分为公有和私有，公有队列被注册在AD域中，其基于域账号的Windows认证机制。

管理队列：确认消息被存储在管理队列中，包括成功确认和失败确认。

回复队列：MSMQ完全采用单向的消息交换模式，消息发送后是没有回复消息返回给发送端的，但有些场景下，需要包括简单确认外的回复内容，这是就涉及服务队列

日志队列：当消息成功发送或接受后，MSMQ可以将消息的拷贝作为发送或接受日志存储起来，分为源日志和目标日志。

事务性队列：MSMQ和SQL Server一样，属于事务管理器（RM，ResourceManager），可以登记到一个分布式事务中。

死信队列：存放限定时间内无法投递信息

报表队列：是公有队列，存储路由跟踪的报表信息

子队列：是一种消息容器，针对消息队列的一个常见操作是将消息从一个队列转移到另一个，其典型应用包括：有序递交，可以将乱序的消息暂时放在子队列中，排序后有序提交；毒性队列，当频繁出错时，可以将其暂存在相应的子队列中以使其他消息得到及时处理。

基于MSMQ的API都集中在System.Messaging中，其常见的路径格式及其示例如下。

|  |  |
| --- | --- |
| 队列类型 | 模式 |
| 日志队列（公有） | {MachineName}\{QueueName}\Journal$ |
| 日志队列（私有） | {MachineName}\Private$\{QueueName}\Journal$ |
| 系统日志队列 | {MachineName}\ Journal$ |
| 系统死信队列 | {MachineName}\ DeadLetter$ |
| 系统事务死信队列 | {MachineName}\ XactDeadLetter$ |
| 公有队列 | DIRECT=HTTP://Xionger-PC/msmq/MyQueue |
| 示例 | String path = @”.\Private$\MyQueue”;  MessageQueue queue = new MessageQueue(path); |

其构造函数，除了path，还包括SharedModeDenyReceive表示当前应用独占目标队列，enableCache表示创建连接缓存，queueAccessModel表示对象由于何种操作(Peek, Receive, Send)。

整个构架包括：消息队列的创建和删除；消息队列的查询；创建一个MessageQueue对象；消息队列的格式名称；消息的发送；MSMQ消息；消息的接收与查看

其事务模型、事务批量操作、会话、错误处理等操作相对复杂，这部分暂时不进行介绍。

事务处理协议，包括OleTx和WS-AT两种：前者采用RPC作为通信手段，使用二进制编码，是最高效的分布式处理协议，但只能用于Windows平台；后者WS-AT是WS-\*的一员，支持跨平台。

WS-Coordination通过一个协调器和若干协调协议定义了一个可扩展的框架去协调一个分布式活动的所有参与者，其核心是协调器，提供激活服务(Activation Service)、注册服务(Registration Service)和协议服务(Protocol Service)。两个不同的应用的事务模型的建立最终会归结为协议服务之间的终结点引用的交换，

接下来，进入实践意义很强的WCF事务编程部分，其主要包含如下3个概念：

**通过服务契约决定事务流转(Transaction Flow)的策略**，通过定义TransactionFlowAttribute来决定事务策略，NotAllowed表示客户端的事务不允许被流转到服务端，服务端也不会视图去接受流入的事务，Allowed则相反，Mandatory表示客户端必须在事务中进行服务调用。

**通过绑定实施事务的流转**，相关配置如下所示。

|  |
| --- |
| <system.serviceModel>  <bindings>  <netTcpBinding>  <binding name ="transactionalTcpBinding" transactionFlow="true" transactionProtocol="WSAtomicTransactionOctober2004"/>  </netTcpBinding>  <ws2007HttpBinding>  <binding name="transactionalHttpBinding" transactionFlow="true"></binding>  </ws2007HttpBinding>  </bindings>  </system.serviceModel> |

**通过服务/操作行为控制事务的相关行为**，相关配置如下所示。

|  |
| --- |
| [ServiceBehavior(TransactionIsolationLevel = IsolationLevel.ReadCommitted, TransactionTimeout = "00:05:00", TransactionAutoCompleteOnSessionClose = true)] |

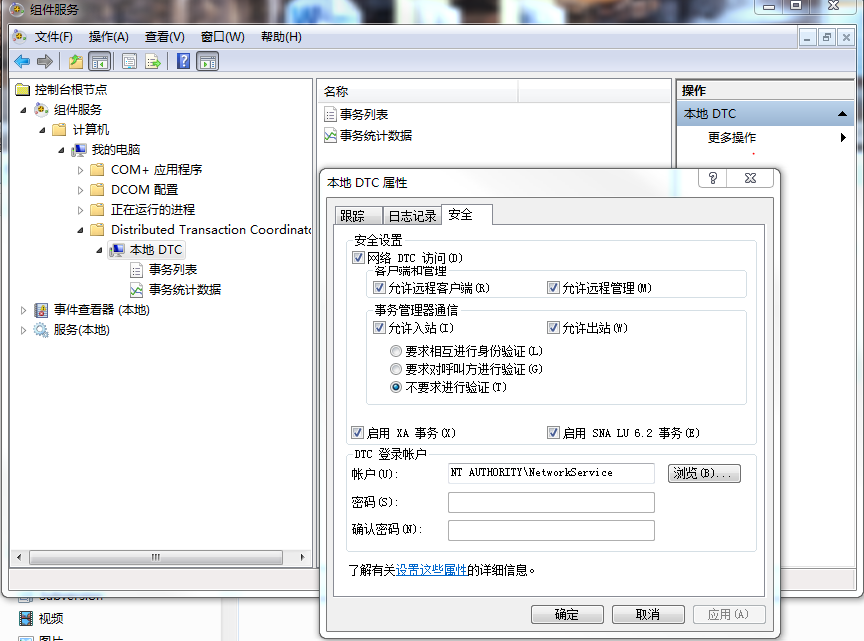
接下来通过一个示例来了解如何创建一个事务型服务（一个银行转账操作，涉及一个银行的取和另一个一个银行的存）。

步骤1服务契约和服务的实现，在服务接口方法上添加特性[TransactionFlow(TransactionFlowOption.Allowed)]，在服务实现方法上添加[OperationBehavior(TransactionScopeRequired=true)]。

步骤2部署服务。

步骤3调用BankingService。

步骤4设置DTC，在控制面板的管理工具中设置，如下图所示。



步骤5采用WS-AT协议，即在binding配置项中添加：<transactionFlow transactionProtocol=”WSAtomicTransactiona11”>

Tip:到目前为止，仍然觉得分布式事务这部分比较复杂，最好虚拟出多机的分布式环境进行实践再学习。

参考资料：

[1]蒋金楠. WCF全面解析[M]. 上海:电子工业出版社, 2012.