**Nesoft.ServiceBus部署使用说明**

目 录

[**一、** **部署概要说明** 2](#_Toc359167952)

[**二、** **部署存储消息的持久化容器（SQL Server / MongoDB）** 3](#_Toc359167953)

[1. 使用SQL Server来作为消息持久化容器 3](#_Toc359167954)

[2. 使用MongoDB来作为消息持久化容器 3](#_Toc359167955)

[**三、** **部署管理事件、订阅者信息和订阅关系的Portal（Asp.Net的Website）** 3](#_Toc359167956)

[1. 部署存储事件、订阅者信息和订阅关系的数据库 3](#_Toc359167957)

[2. 在一个有IIS 7.5的web服务器上部署Portal 3](#_Toc359167958)

[3. 配置队列 4](#_Toc359167959)

[4. 配置事件 5](#_Toc359167960)

[5. 配置订阅者 6](#_Toc359167961)

[6. 配置订阅关系 7](#_Toc359167962)

[**四、** **部署分发事件消息的DispatchHost（Windows Service）** 8](#_Toc359167963)

[1. 修改配置文件 8](#_Toc359167964)

[2. 安装windows服务 9](#_Toc359167965)

[**五、** **搭建事件消息订阅方的Service（Restful Service or Webservice）** 9](#_Toc359167966)

[1. Restful Service 9](#_Toc359167967)

[2. SOAP 1.1的Webservice 10](#_Toc359167968)

[**六、** **部署事件消息发起方（第三方的任何应用系统）** 10](#_Toc359167969)

[1. 引用程序集 10](#_Toc359167970)

[2. 修改配置文件 10](#_Toc359167971)

[3. 定义消息 10](#_Toc359167972)

[4. 代码中调用EventPublisher来发送消息 11](#_Toc359167973)

[**附1、消息队列中消息处理情况的实时管理** 11](#_Toc359167974)

[1. 统计查看整个Queue中的消息情况 12](#_Toc359167975)

[2. 按条件查询消息分发任务 12](#_Toc359167976)

[3. 查看消息序列化后的字符串内容 12](#_Toc359167977)

[4. 查看消息的详细处理日志（包括成功、失败的情况） 12](#_Toc359167978)

[5. 人工将以处理的消息重新放回待处理列表中，以再次进行处理 12](#_Toc359167979)

[**附2、使用统一服务代理** 12](#_Toc359167980)

[1. 完成前面第三点的1和2的部署 12](#_Toc359167981)

[2. 部署好服务提供方 13](#_Toc359167982)

[3. 在Portal中配置服务提供者 13](#_Toc359167983)

[4. 在服务消费方代码中使用ServiceProxy 14](#_Toc359167984)

1. **部署概要说明**

Nesoft的部署使用总共分为5个步骤，分别为：

* + 部署存储消息的持久化容器（SQL Server / MongoDB）；
  + 部署管理事件、订阅者信息和订阅关系的Portal（Asp.Net的Website）；
  + 部署分发事件消息的DispatchHost（Windows Service）；
  + 搭建事件消息订阅方的Service（Restful Service or Webservice）；
  + 在第三方的事件发起应用程序中使用EventPublisher，以实现事件消息的发布（在本地引用一个dll方式）；

这5个步骤中，前三个步骤是基础结构的部署，第一次部署好后将来没有特殊情况下就不用再动了，而第4、5步则是运维期间根据业务应用系统的需要来部署。接下来本文档会依次详细说明每个部署步骤需要做的事情。

1. **部署存储消息的持久化容器（SQL Server / MongoDB）**
2. 使用SQL Server来作为消息持久化容器
3. 安装SQL Server 2005或以上版本的数据库服务（需要支持Service Broker）；
4. 建立一个新的Database，命名为DBQueue（当然也可以自定义其他的名字，但如果是自定义的其他名字，则需要将第(3)步中所执行脚本开头的数据库名字[DBQueue] 替换为自定义的名字）；
5. 执行解决方案中Nesoft.ServiceBus\Database目录下的SQL脚本文件DBQueue.sql；

到此基于SQL Server的消息持久化容器的部署工作就完成了；

1. 使用MongoDB来作为消息持久化容器

因为MongoDB为Nosql数据库，是Schema-less模式的，所以我们需要做的就是安装好一个MongoDB的数据库服务即可，具体的Windows系统下的安装方式为：

1. 下载最新版的MongoDB，地址：<http://www.mongodb.org/downloads>，请注意选择正确的系统版本；
2. 解压所下载的文件，找到里面的bin目录，将bin目录复制到存放MongoDB数据库实例的目录下，然后建立2个和bin目录平级的空目录，分别命名为log和data；
3. 在bin目录所在的目录里建立bat文件install.bat，bat文件里的内容为：*D:\Project\MongoDB\DBInstance\bin\mongod --dbpath "D:\Project\MongoDB\DBInstance\data" --directoryperdb --logpath "D:\Project\MongoDB\DBInstance\log\logs.txt" --logappend --install --serviceName "MongoDB"*

注意需要将命令中“D:\Project\MongoDB\DBInstance”这部分路径替换为你自己当前bin目录所在的目录路径；

1. 双击运行该bat文件，则会将MongoDB的数据库服务作为Windows服务安装到系统中，data目录为数据库数据文件存放的目录，而log目录里则是存放的日志文件；

（如果要卸载该MongoDB的Windows服务，则执行命令：【*D:\Project\MongoDB\DBInstance\bin\mongod --remove --serviceName "MongoDB"*】；注意需要将命令中“D:\Project\MongoDB\DBInstance”这部分路径替换为你自己当前bin目录所在的目录路径）

到此基于MongoDB的消息持久化容器的部署工作就完成了；

1. **部署管理事件、订阅者信息和订阅关系的Portal（Asp.Net的Website）**
2. 部署存储事件、订阅者信息和订阅关系的数据库
3. 选择一个SQL Server数据库服务器实例（可以和前面用来持久化消息的SQL Server数据库实例使用同一个，也可以不同，它们之间没有任何关系），在数据库实例上创建一个新的Database，命名为SubscriptionConfig，当然也可以命名为其他任意名称；
4. 执行解决方案中Nesoft.ServiceBus\Database目录下的SQL脚本文件SubscriptionConfig.sql；注意如果第二步里创建的数据库是自定义的其他名称，那么需要修改脚本文件中开头的USE [SubscriptionConfig]；
5. 在一个有IIS 7.5的web服务器上部署Portal
6. 部署解决方案中Nesoft.ServiceBus\Portal\Nesoft.ServiceBus.WebPortal这个网站到IIS里（独立网站或网站下的一个应用程序都可以），需要使用.net 4.0集成模式的应用程序池；
7. 修改所部署网站的配置文件web.config中的name为”SubscriptionConfig”的数据库连接字符串，将其value设置为前面一步中所部署的数据库的连接字符串；
8. 配置队列

（现在还没有UI功能，只能直接到数据库中添加配置，后期会继续开发配置功能）

1. 在dbo.Queue表中添加队列信息：

**ID：**自增主键；

**Title**：必填，队列的名称，会显示在Portal的界面上；

**Description**：选填，对队列的描述信息，也会显示在Portal的界面上；

**QueueClassType**：必填，表示队列所使用的EventQueue抽象类的具体实现类类型名称；如果是基于SQL Server实现的队列，则填Nesoft.ServiceBus.Queue.SqlServer.SqlDbQueue,Nesoft.ServiceBus.Queue；如果基于MongoDB实现的队列，则填Nesoft.ServiceBus.Queue.MongoDb.MongoDbQueue,Nesoft.ServiceBus.Queue；

**PublisherClassType**：必填，表示队列所使用的EventPublisher抽象类的具体实现类类型名称；如果是基于SQL Server实现的队列，则填Nesoft.ServiceBus.Consumer.SqlDbEventPublisher, Nesoft.ServiceBus.Consumer；如果基于MongoDB实现的队列，则填Nesoft.ServiceBus.Consumer.MongoDbEventPublisher,Nesoft.ServiceBus.Consumer；

**ManagerClassType**：选填，表示队列所使用的IQueueManager接口的具体实现类类型名称；如果是基于SQL Server实现的队列，则填Nesoft.ServiceBus.QueueManagement.SqlServer.SqlDbQueueManager,Nesoft.ServiceBus.QueueManagement；如果基于MongoDB实现的队列，则填Nesoft.ServiceBus.QueueManagement.MongoDb.MongoDbQueueManager,Nesoft.ServiceBus.QueueManagement；

**Status**：必填，A表示正常可用状态，D表示禁用状态；

1. 在dbo.QueueParameter表中为队列配置键值对参数：

**TransID**：自增主键；

**QueueID**：所配置的Queue的在dbo.Queue表中的ID字段的值；

**ParamName**：键值对的键名；

**ParamValue**：键值对的值；

对于从ReliableQueue派生出来的队列的实现，有如下的配置项：

checkReservedMsgIntervalSeconds：选择配置；检查那些已被获取处理中的消息是否处理超时的轮询的间隔时间，默认为每30秒检查一次，如果有超时的则会将该消息重新放回待处理消息队列中；

checkAlertMsgIntervalSeconds：选择配置；检查Alert队列的轮询的间隔时间，默认为每60秒定时检查一次Alert队列表获取需要进行提醒的任务已进行提醒操作；

checkWaittingMsgIntervalSeconds：选择配置；检查待处理消息队列的轮询的间隔时间，默认为每60秒定时检查一次消息队列表获取待处理消息；

checkIncomeMsgIntervalSeconds：选择配置；检查新进入的待路由的消息队列的轮询的间隔时间，默认为每60秒定时检查一次新进的消息队列表获取待路由的消息；

reserveTimeoutSeconds：选择配置；设置消息被处理的超时时间，默认为300秒，即取出消息后5分钟内没有Ack该消息则会被认为处理超时；

maxAllowedFailedTimes：选择配置；最大允许失败的次数（总共会执行的次数为该次数加1），默认为允许处理失败119次，即总共最多会执行120次处理，第120次处理后如果还出错就会将消息分发任务移入到Failed Message队列中；

alertPerFailedTimes：选择配置；针对一条消息任务，每处理失败多少次后就产生一个Alert记录，默认为每3次失败后就产生一个Alert

failedMsgToDelaySeconds：选择配置；每处理失败一次，累加多少秒后消息才能被获取，默认延后累加60秒（即1分钟），如果配置的为0或小于0，则不做延时累加；

对于基于SQL Server的队列实现，有如下的特有键值对可配置：

connectionString：必须配置；队列所在的SQL Server服务器实例的ADO.Net数据库连接字符串；

table：必须配置；队列所用的库表名；

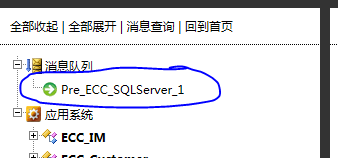
对于基于MongoDB的队列实现，有如下的特有键值对可配置：

connectionString：MongoDB的连接字串，如mongodb://localhost:27017/?safe=true

database：队列所在的数据库名

queueName：队列的名字（会作为系列表的表名前缀）

到此管理事件、订阅者信息和订阅关系的Web Portal就部署好了，可以在浏览器中访问了，并且可以在左边的树状节点上看到所配置好的队列的信息；



1. 配置事件

选择该事件消息的发起方应用系统（如果该应用系统不存在，可以在Portal中先创建一个）,添加一个事件，需要指定事件的名称、主题（subject）和事件的描述信息；其中主题（subject）需要保证全局的唯一性，它将是后续订阅关系的基础；而名称和描述则是纯粹给人看的，程序不会使用到。



添加好的事件将在左边的树上看到，并可以选择以进行维护：

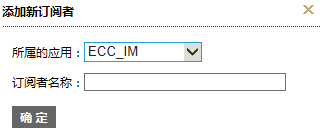


另外我们有定义了一份事件命名规范，需要大家必须遵守；

对于事件的描述信息，我们也有定义了相关规范，要求添加事件后必须要按规范要求完善其事件的描述信息，以便后期维护方便。

1. 配置订阅者

选择该订阅者所属的应用系统（如果该应用系统不存在，可以在Portal中先创建一个）,然后添加一个订阅者，添加时需要指定该订阅者的名称（该名称为文本信息，仅仅是给人看的，对系统程序没有任何逻辑意义）：



添加好的事件将在左边的树上看到，并可以选择以进行维护，其中最重要的是，是需要为该订阅者配置其服务的端点（Endpoint），通过点击“添加处理程序”按钮来添加，一个订阅者可以添加多个处理程序（即可以有多个Endpoint），这样可以实现软load balance和防止单点失败（消息分发服务会自动进行循环的load balance和retry）：



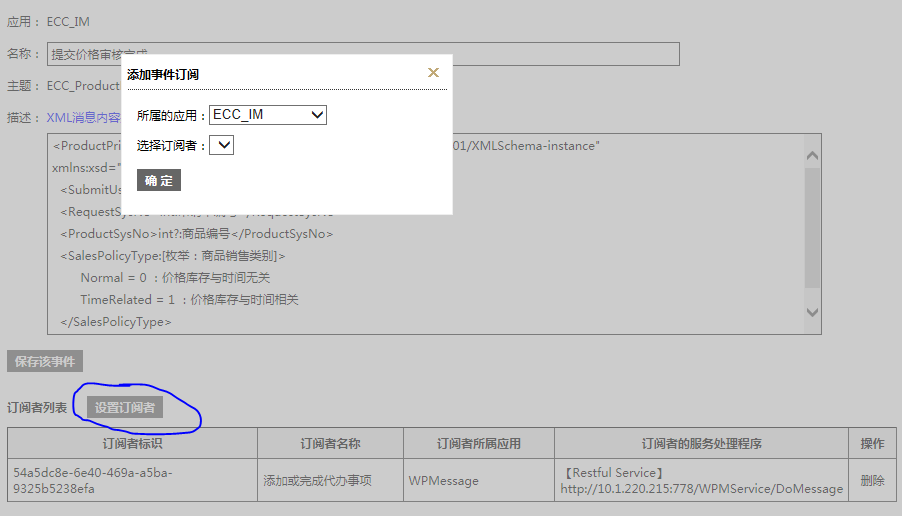
点击“添加处理程序”后：（当前支持restful和soap 1.1的两种处理程序，将来可以扩展支持更多类型）

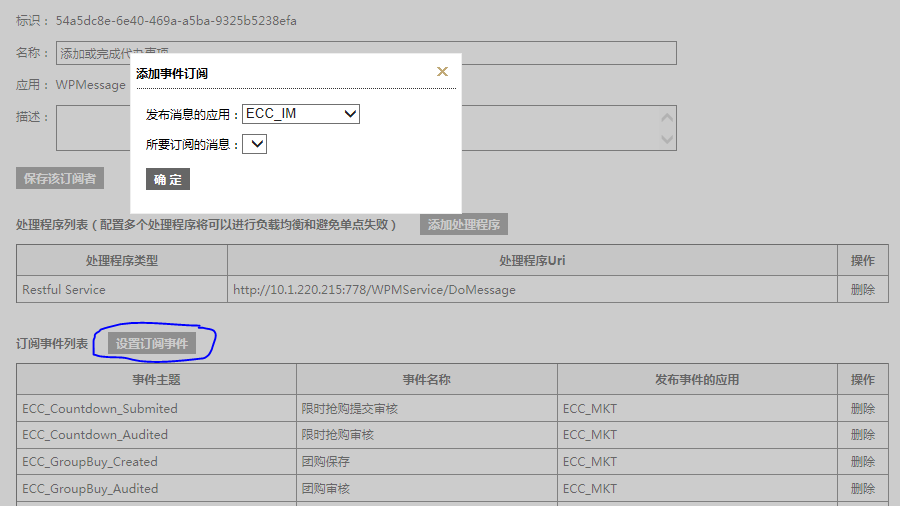




1. 配置订阅关系

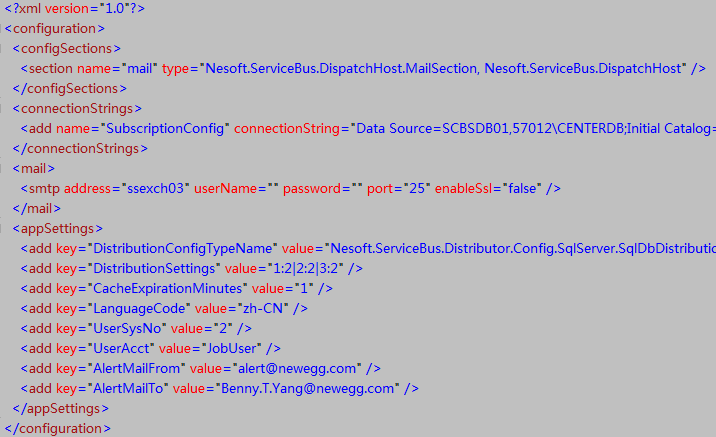
可以通过事件，来给事件添加订阅者；也可以通过订阅者，来添加所订阅的事件；两种操作的效果和结果是完全一致的，只是2个不同的维度视角而已；





1. **部署分发事件消息的DispatchHost（Windows Service）**
2. 修改配置文件

* Subscription所在的数据库的连接字符串：
* <connectionStrings>节点下add节点，name必须为SubscriptionConfig，connectionString则根据实际环境进行配置；
* 配置邮件服务器、提醒警告邮件的发件人、收件人：
* <mail>节点下可以配置1个或多个邮件服务器信息（配多个的原因是用于防止单点出错）；
* <appSettings>下的<add key="AlertMailFrom">和<add key="AlertMailTo">用来配置警告提醒邮件的发件人和收件人；收件人可以配置多个邮件地址，使用英文逗号或分号隔开；
* Distributor的相关配置（配置的模式、监控的Queue及线程、配置缓存的过期时间）：
* <appSettings>下的<add key="DistributionConfigTypeName">用来配置所使用的加载配置数据的实现类，现在只有SQL Server的实现，所以其value固定为Nesoft.ServiceBus.Distributor.Config.SqlServer.SqlDbDistributionConfig, Nesoft.ServiceBus.Distributor；
* <add key="DistributionSettings">用来配置该Distributor所监控的Queue的ID，以及该Queue的消息分发调用订阅者服务所使用的线程数，其格式为{QueueID}:{线程数}|{QueueID}:{线程数}|{QueueID}:{线程数}；其中QueueID为在SubscriptionConfig的配置数据中用来唯一标识一个Queue的ID；
* <add key="CacheExpirationMinutes">用来配置缓存的SubscriptionConfig数据的过期时间；出于性能考虑，所以Distributor会把消息路由分发的配置数据缓存到内存里，这里设置的时间也就是这些缓存数据的过期时间；
* Restful的Client的配置（语言、上下文中的用户信息）
* <add key="LanguageCode">用来配置RestClient的调用请求的上下文里的语言；
* <add key="UserSysNo">用来配置RestClient的调用请求的上下文里的用户唯一编号；
* <add key="UserAcct">用来配置RestClient的调用请求的上下文里的用户登录账号；



1. 安装windows服务

安装windows服务的服务器需要能够：

* 访问Queue服务器（SQL Server or MongoDB的服务器）；
* 访问Subscription的配置所在服务器（SQL Server服务器）；
* 访问订阅者的Service（通过http方式，因为当期只支持restful和soap方式）；

到此我们的Distributor就已经配置、部署好了。

1. **搭建事件消息订阅方的Service（Restful Service or Webservice）**
2. Restful Service

对订阅方的RestfulService实现有如下3个限制要求：

1. 服务方法必须要能够接受Post方式
2. 服务方法接受的请求的格式必须按Json格式，即要求RequestFormat=WebMessageFormat.Json
3. 服务方法只能有一个入参，参数名称可以随意，该参数的类型的名称也可以随意定义，但该参数类型里必须要有4个public的property：

public string MessageID { get; set; }

public string Subject { get; set; }

public string SubscriberID { get; set; }

public string Content { get; set; }

通过该参数的MessageID属性就可以得到该次调用的消息的唯一标示，通过该属性数据可以让订阅方Service方法实现幂等性，即同一个消息的多次调用该方法不会导致数据错误（最简单的实现方式就是把执行过的MessageID保存下来，然后每一次的调用都要判断传来的MessageID是否已经存在了，以判断该消息是否已经执行过了，如果执行过是重复执行那么就可以忽略这次的重复调用）；

Subject属性用来指明了本次所订阅的消息的Subject；订阅方可根据自身需求选择性使用；

SubscriberID是当前订阅方的唯一标志；

Content是消息数据的内容，在Publisher方被序列化为字符串了，这里需要订阅方Service自行根据Publisher那边的序列化类型来进行反序列化，当前支持三种序列化方式：Xml、Json、二进制Base64；如果使用SQL Server的DBQueue，则只能使用Xml方式；

1. SOAP 1.1的Webservice

对订阅方的Webservice实现有如下2个限制要求：

1. 所暴露的wsdl需要是flat的，即在一个url地址里返回所有的wsdl信息（WCF的BasicHttpBinding方式的service是标准的Soap 1.1，但是其返回的wsdl是被分拆到多个文件里的，这种就无法被成功调用，需要使用一个FlatWsdlBefavior的EndpointBefavior来保证其返回的wsdl是flat的；传统的asmx的webservice就是flat的wsdl，就没有问题）；
2. 订阅方Webservice的方法的参数必须为4个string类型的参数（参数名称可以随意）：

public void XXXXX(string id, string subject, string subscriberID, string content)

其中第一个参数为该次调用的消息的唯一标示；

第二个参数为本次所订阅的消息的Subject；订阅方可根据自身需求选择性使用；

第三个参数为本次订阅者的唯一标示；

第四个参数为消息数据的内容，在Publisher方被序列化为字符串了，这里需要订阅方Service自行根据Publisher那边的序列化类型来进行反序列化，当前支持三种序列化方式：Xml、Json、二进制Base64；如果使用SQL Server的DBQueue，则只能使用Xml方式；

1. **部署事件消息发起方（第三方的任何应用系统）**
2. 引用程序集

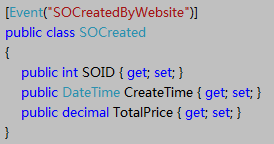
在消息发起方的应用系统中引用程序集Nesoft.ServiceBus.Consumer，这是一个.net开发的dll文件，直接在项目工程里本地引用即可；

1. 修改配置文件

修改消息发起方的应用系统的app.config或web.config文件，在<appSettings><appSettings />节点下添加：  
<add key="ServiceRegistryConfigUrl" value="http://10.1.220.206:6636" />  
<add key="DefaultQueue" value="1" />  
其中ServiceRegistryConfigUrl的value表示配置文件所在的路径，可以是远程url地址，也可以是本地硬盘上的物理路径（支持绝对路径，也支持相对路径）；这里的url可以有2种配置方法：  
*(1). 这个url可以直接是http://[DistributorHost的IP]:6636，注意需要把服务器上防火墙的6636端口打开，这种方式的好处是配置直接来源于数据库，通过管理Portal那边调整了Queue的配置，那么这边会自动得到更新后的配置信息，但要求就是消息发布方的应用需要能够访问到DistributorHost的6636端口了；  
(2). 另外如果消息发布方的应用无法访问到DistributorHost的6636端口（比如网络隔开的），那么可以在本地配置一个xml文件（可以直接访问http://[DistributorHost的IP]:6636然后另存下来，但需要注意保存下来的文件编码方式，建议复制xml内容到txt里然后再保存以免出现文件编码问题），然后将ServiceRegistryConfigUrl的值配置为这个本地文件的绝对或相对路径都可；*  
而DefaultQueue的value则是表示默认使用的Queue的ID（这个ID是对应在ServiceRegistryConfigUrl所指定的配置文件中的Queue的ID）；

1. 定义消息

在消息发起方的应用系统的代码中，定义代表事件消息的实体class有2种方式：

* 1. 需要在class上添加EventAttribute，用来指定其subject，该subject需要等于第一步中在消息服务总线配置中心所配置的事件的主题（subject）；例如：  
     

该EventAttribute还有重载方法，除了可以用来指定该消息的subject外，还可以指定消息的优先级和消息序列化方式。默认情况下，如果没有指定subject，那么将默认使用class的类型的FullName（带命名空间的，但不带程序集名字）来作为subject；如果没有指定优先级，那么默认的消息优先级等于1024；如果没有指定序列化方式，那么默认的使用xml序列化方式。

* 1. 使用EventMapping.AddEvent<T>(...)的静态方法，如下：

EventMapping.AddEvent<SOCreated>("SOCreatedByWebsite");

这样就把SOCreated这个类型的实体映射到了"SOCreatedByWebsite"这个消息主题上了，等同于上面a)中的使用EventAttribute的效果；同样的AddEvent<T>方法有重载方法，除了可以用来指定该消息的subject外，还可以指定消息的优先级和消息序列化方式。如果没有明确指定subject或优先级或序列化方式，那么其默认情况和a)里使用EventAttribute方式时的默认情况一致。

此时Service端所拿到的Content的string则为该实体对象按指定序列化方式进行序列化后的字符串。

1. 代码中调用EventPublisher来发送消息
   1. 将消息发送到指定的Queue中：



这里EventPublisher.Create方法的入参就对应于app.config或web.config中ServiceRegistryConfigUrl所指定的配置文件中的Queue的ID；

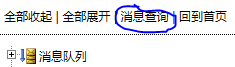
* 1. 将消息发送到默认的Queue中：



这里所使用的Queue，将会是app.config或web.config中DefaultQueue所配置的Queue；

**附1、消息队列中消息处理情况的实时管理**

可以使用左上角的消息查询功能：

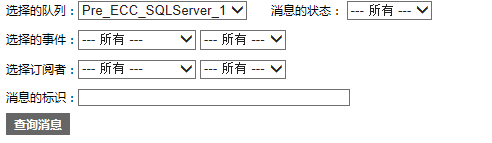


1. 统计查看整个Queue中的消息情况

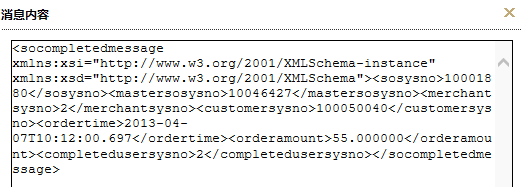


1. 按条件查询消息分发任务

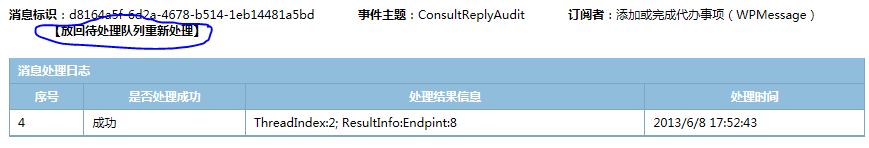
可以按照不同的Queue来根据消息状态（待处理、处理成功、处理失败）、消息的事件类型、订阅者等条件来查询消息；（后续还会加上时间段条件，可以按照消息进入队列的时间来查）



1. 查看消息序列化后的字符串内容



1. 查看消息的详细处理日志（包括成功、失败的情况）



1. 人工将以处理的消息重新放回待处理列表中，以再次进行处理

见前一幅截图中蓝色圈其的按钮；

**附2、使用统一服务代理**

1. 完成前面第三点的1和2的部署
2. 部署好服务提供方

当前支持标准Soap 1.1的webservice、标准Restful Service和ECC Restful Service（和标准Restful Service相比，会多出对业务上下文的自动构建和传输，这个是ECC系统要求的），将来可以根据需要再进行扩展支持更多的类型；

1. 在Portal中配置服务提供者

首先选择该服务提供者所属的应用系统（如果该应用系统不存在，可以在Portal中先创建一个）,添加一个服务，需要指定该服务的标题、标识（Service Key）和服务类型；其中标识（Service Key）需要保证全局的唯一性，消费端将根据该标识（Service Key）来找到和调用该服务；不同的服务类型则会决定使用不同的配置参数；标题则是纯粹给人看额文本信息；



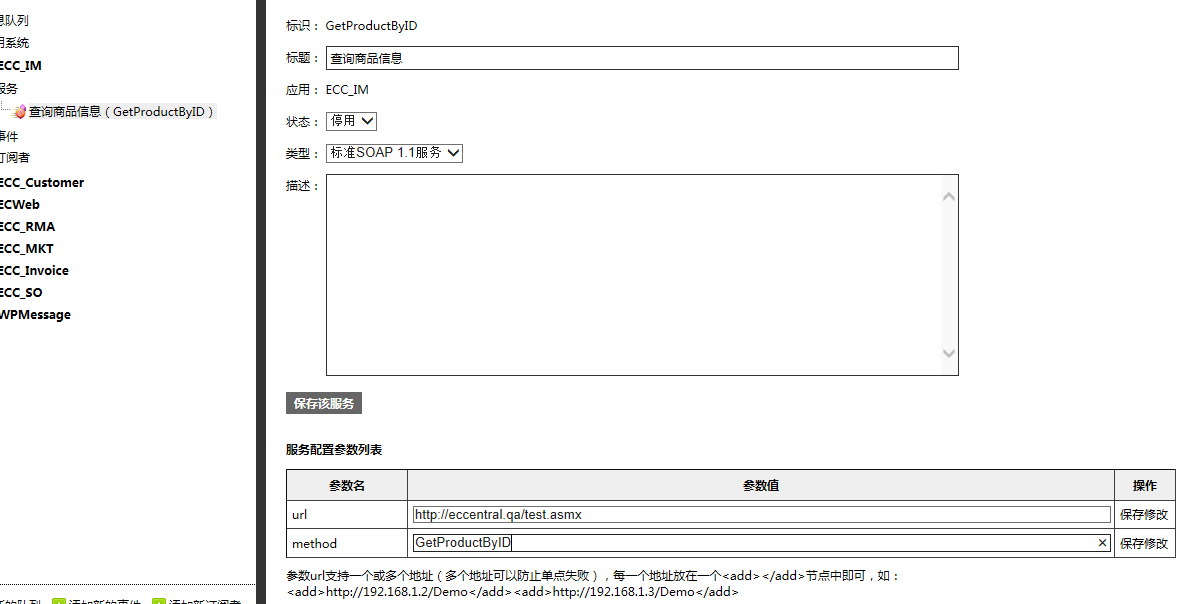
添加好后，在左边的树上就可看到，点中就可以进行进一步的配置：



当服务类型为Restful service或ECC Restful Service时，需要为该服务配置url、http method和message format，以保证consumer在调用该服务时使用正确的相关配置；其中url又可以通过一个<add></add>的xml结构来配置多个url，以实现load balance和防止单点失败。

另外当Restful service或ECC Restful Service使用Get的http method时，会根据consumer的调用方传入的调用参数然后来构建出不同的url来，所以此时是通过在url字符串里放入标准站位符“{数字}”方式，在运行时来被方法入参替换的。

而如果服务类型为标准的soap 1.1的webservice的话，其配置界面如下：



这里需要为服务配置的就是url和method（webservice的方法名）了，同样这个url可以通过一个<add></add>的xml结构来配置多个url，以实现load balance和防止单点失败。

1. 在服务消费方代码中使用ServiceProxy
2. 在消费方系统中引用程序集Nesoft.ServiceBus.Consumer，这是一个.net开发的dll文件，直接在项目工程里本地引用即可；
3. 配置消费方系统的app.config或web.config文件，在<appSettings><appSettings />节点下添加：  
   <add key="ServiceRegistryConfigUrl" value="http://10.1.220.206:6636" />  
   ServiceRegistryConfigUrl的value表示配置文件所在的路径，可以是远程url地址，也可以是本地硬盘上的物理路径（支持绝对路径，也支持相对路径）；
4. 代码中用ServiceProxy来调用服务；
   1. 同步调用，无返回值

ServiceProxy.Find("SOOutstock").Invoke("参数1", "参数2", …, "参数n");

* 1. 同步调用，有返回值

ProductInfo info = ServiceProxy.Find("GetProductByID").Invoke<ProductInfo>("参数1");

* 1. 异步调用，无返回值

ServiceProxy.Find("SOOutstock").InvokeAsync(() => { }, ex => { }, "参数1", "参数2", …, "参数n");

其中第一个参数的Action，表示异步调用线程服务调用成功后的回调；第二参数Action<Exception>表示异步调用线程中发生异常后的异常处理方法；

* 1. 异步调用，有返回值

ServiceProxy.Find("SOOutstock").InvokeAsync<ProductInfo> (info => { }, ex => { }, "参数1");

和c)的唯一差异就在于第一个参数为Action<ProductInfo>，成功回调方法有入参，该入参就为调用服务的返回值；