基于消息中间件的数据交换平台传输框架设计①

梁彦杰[1,2]，廉东本[2]

[1](中科院研究生学院，北京 100039)
[2](中国科学院沈阳计算技术研究所，沈阳 110168)

摘 要：在为辽河流域的不同应用系统搭建数据交换平台的过程中，采用 Web Services 在节点之间进行数据传输存在着传输不可靠、不能很好地支持复杂数据、仅支持同步传输等问题。首先对 Web Services 下可靠传输的方案进行了探讨，在此基础上提出了基于消息中间件的数据交换平台的传输框架，并对该传输框架下的数据传输流程和具体组件设计做了详细介绍。

关键词：数据交换平台；Web Services；消息中间件；可靠性；可扩展

Design of Architecture on Transporting Data for Data Exchange Platform Based on MOM

LIANG Yan-Jie[1,2], LIAN Dong-Ben[2]

[1](Graduate University, Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)

Abstract: During the process of constructing data exchange platform for different application systems which belong to LiaoHe Basin, We found that there existed several problems by using web services to transport data between two application systems such as low reliability, the difficulty of supporting the complex data from different data source, the only support of synchronous process. In this article, firstly, we did a research on the solution of reliability data transmission based on web services. Secondly we had the idea of designing the Architecture on Transporting Data based on MOM. At last, we described its working process and the details of its components.

Key words: data exchange platform;web services; message-oriented middleware; reliability; extendable

数据交换平台能够有效地对异构的数据源进行集成和共享，一方面满足了人们宏观上掌握数据的需求，另一方面促使交换各方及时、安全、高效地传递数据，避免了信息孤岛的存在。在辽河流域水污染治理过程中，数据交换平台在协调不同部门之间数据通信中扮演了重要角色，是辽河流域环境风险评估与预警平台课题的重要组成部分。其中，数据传输是数据交换的前提和重要保障，本文旨在设计一个高效、可靠、准确和可扩展的数据传输框架，提高数据交换平台的性能。

1 存在的问题

现有的数据交换平台在开发过程中不同节点之间数据传输是采用 Web Services 实现的，中心节点（数据交换平台的交换引擎所在，负责对数据进行清洗、转换、校验等处理）和客户端（辽河流域下各个异构应用系统）都要部署 Web Services。由于 Web Services 采用无状态、不可靠的 HTTP 协议作为底层的传输协议[1]，在测试的过程中，发现数据传输不可靠。图 1 代表数据交换平台进行数据交换的一个任务流程，节点 A 发送数据到中心节点，然后由中心节点将处理之后的数据发送给 B 节点。

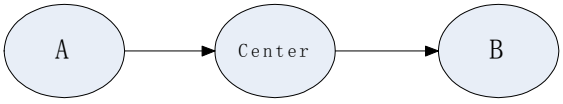

图 1 数据交换平台的一个任务流程示意图

① 收稿时间:2011-07-17;收到修改稿时间:2011-08-22

数据交换的过程中的不可靠性具体表现：首先，中心节点和 B 节点任何一个 Web 服务停止就会导致任务失败；其次，当平台运行在一个复杂的网络环境下或者节点的处理速度过慢，发送节点在 HTTP 的超时重传的机制下就会多次发送数据；第三，由于中心节点下有很多客户端，同时运行多个任务时，中心节点不能灵活地实现负载均衡。

此外，采用 Web Services 还存在着其他几个问题：

1) 不能很好地支持复杂数据

由于水污染治理过程不同的应用系统要处理各种复杂的数据，针对每一类型的复杂数据，必须提供相应的 Web 服务并且开发不同的客户端，降低了数据交换平台的可扩展性。

2) 数据传输与数据交换引擎过耦合

平台将数据的接收、发送以及解析动作分别作为资源放在了数据交换引擎中一起被调度系统进行处理，这种数据传输和数据处理的耦合，即增加了平台监控的复杂性，又使平台异常处理和事务回滚变得困难。

2 可靠性方案及传输框架的提出

数据传输的可靠性是 Web Services 的一个缺陷，学术界以及商业领域针对 Web Services 的可靠性提出了很多解决方案。这些解决方案大致可以分为两类：一类是通过扩展消息头部来保证可靠传输，如结构化信息标准促进组织（OASIS）提出的 WS-Reliability 标准。这个标准的本质是在 SOAP 消息头部中加入确认和重传机制的标签内容[2]，以此来保证消息的一次送达和按序送达。这种方式虽然解决了可靠性问题，但却增加了 Web Services 应用层处理逻辑的复杂性，应用层不仅要处理自己本身的业务逻辑同时还要处理 Web Services 的可靠性逻辑；第二类的解决思路是用可靠的传输协议替代 Web Services 底层的不可靠的 HTTP 协议，如使用 JMS 来充当底层的传输协议，Apache CXF 服务框架通过配置可以搭建这样的 Web Services。第二类方案的演变是采用底层可靠的数据传输协议构造一个消息中间件（MOM, Message-Oriented Middleware）来保证数据可靠传输[3]，这样 Web Services 框架被分成了两部分，上层提供服务，下层进行可靠传输。国防科技大学 StarWebAdapter 系统[4]以及 Piyush Maheshwari 等人设计的 WSMQ 系统[5]都是采用这种思想来实现的。

尽管这种解决思路保证了数据传输的可靠性，但是由于其仍然属于 Web Services 的框架，解决了可靠性却不能解决上文提到的其他问题。而且数据传输的效率也严重降低，因为发送节点先要将原始数据封装成 SOAP 消息，然后通过消息转换组件（SOAP 消息与 MOM 消息互相转换）封装处理，接收节点进行相反的动作进行处理，在整个过程中加了很多 SOAP 标签，使传输的数据量增大。

基于以上研究，这里提出基于消息中间件的数据传输框架，即数据是通过消息中间件进行传输的，Web Services 不是用来进行数据传输而是对数据进行处理，工作在消息中间件的上层。进行数据传输时发送节点首先调用与发送节点同在的 Web Services 进行数据处理，然后将处理之后的数据通过消息中间件传递给接收节点。

3 传输框架的设计

客户端与中心节点拥有相同的传输框架结构，不同的是中心节点的传输框架上层是数据交换平台的交换引擎，客户端上层是辽河流域下的各应用系统。传输框架的结构图如下所示：

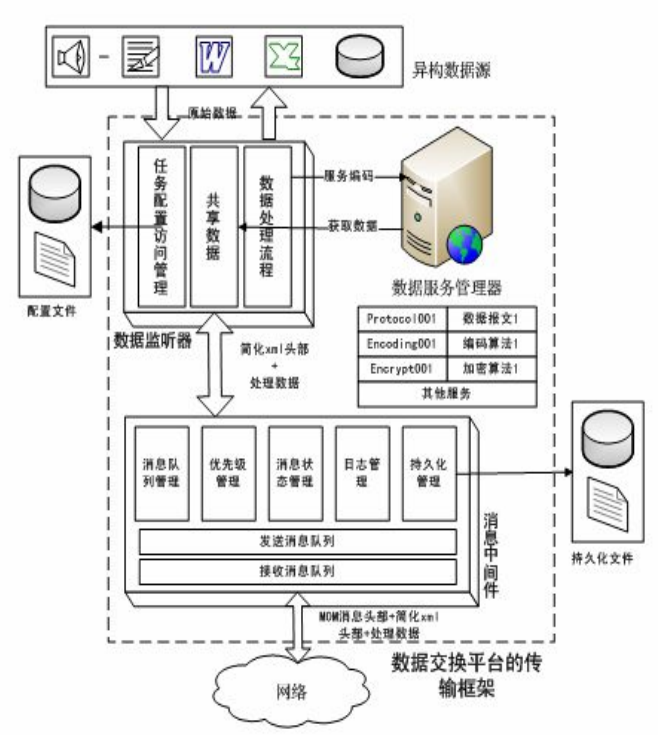

图 2 传输框架的结构图

其中，数据服务管理器中将各种数据处理的方法发布成服务；数据监听器是数据处理的管道，它连接上层应用系统和底层消息中间件，根据任务配置的服务编码调用数据服务管理器中的服务，完成数据的加工处理的流程；消息中间件是传输框架的核心，在各种可靠性机制下负责将经过数据监听器处理的数据可靠地在节点间传递。数据监听器和数据服务管理器的具体设计实现见下文中描述，消息中间件采用实现高级消息队列协议（AMQP）的 RabbitMQ，这个消息中间件有着完善的可靠性机制并且使用方便。以节点 A 传向平台为例，传输框架下的数据传输过程为：

第一，节点 A 数据监听器监听到应用层数据的变化，查询变化数据对应的任务配置信息（数据类型、目的节点的信息、任务优先级、加解密信息、解压缩信息等），生成唯一的事件编号，将事件编号和任务的配置信息写成 XML 字符串作为自定义报文的头部。

第二，按照报文头部定义的数据封装流程，调用数据服务管理器的数据服务对数据进行加工，如加密、压缩等，处理之后的数据与报文头部组装成报文。

第三，节点 A 数据监听器根据报文中的目的信息查询路由表获取接收节点的信息，并将这些信息和自定义的报文传递给消息中间件的消息发送队列中，同时客户端数据监听器中记录任务日志。

第四，报文由消息中间件发送到中心节点的消息队列。

第五，中心节点的数据监听器检测到消息队列中的报文，取出解析其头部，抽取出报文数据的服务编码，调用中心节点的数据服务对报文数据进行解析，获得原始的数据。

第六，中心节点的数据监听器将原始数据交付给上层进行处理。经过平台交换引擎处理之后的数据再按照前三步从中心节点发送给最终的数据接收节点。

3.1 数据监听器的详细设计

如上所述，数据监听器完成了数据接收、数据加工处理、数据投递的管道处理流程，在传输框架中充当应用层与服务管理器以及服务管理器与消息中间件之间的接口。数据监听器主要是围绕自定义报文展开的，最终形成的自定义报文是消息中间件中要发送的消息，报文头部的长度是固定的，报文数据是对原始数据按照可选的规则来组织数据的，如用分隔符来对简单类型的数据拼接等。数据监听器的核心部件如图 3 所示，各部分组件的实现的具体功能为：

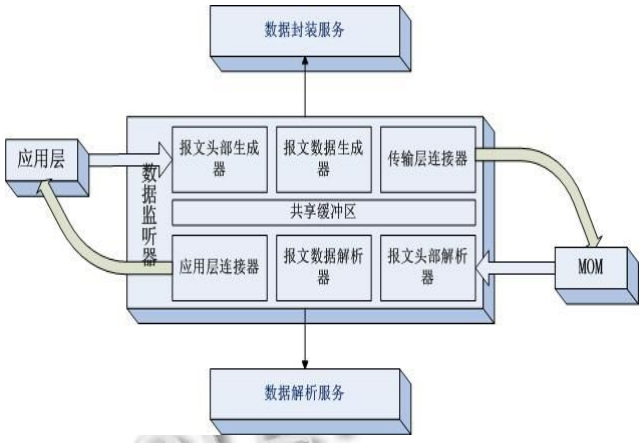

图 3　数据监听器的模块图

报文头部生成器：数据监听器通过定期扫描要交换的数据源，如数据库中的数据或者某个路径下的文件，如果有数据，查询该数据源关联的任务编号，根据任务编号查询任务配置信息，将配置信息作为自定义报文的头部存入共享缓冲区。一个可能的报文头部如下：

<Head>
<EventID>2011062623431123Task001</EventID>
<TaskID>Task001</TaskID>
<Priority>5</Priority>
<AuthorizationUser>LYJ86Client</AuthorizationUser>
<AuthorizationPassword>client86LYJ</AuthorizationPassword>
<RegisterDomain>SYDomainCenter</RegisterDomain>
<RegisterDomainIP>192.168.131.73</RegisterDomainIP>
<RegisterDomainPort>8080</RegisterDomainPort>
<DestinationDomain>NMGDomain</DestinationDomain>
<DestinationDomainIP>192.168.139.185</DestinationDomainIP>
<DestinationDomainPort>8080</DestinationDomainPort>
<BeforeService>
<DataProtocol>Protocol001< /DataProtocol>
<DataFactory>Encoding001</DataFactory>
<Encryption>Encryption001</Encryption>
<Compression>Compression001</Compression>

```
</BeforeService>
<AfterService>
<Decompression>Decompression001</Decompression>
<Decryption>Decryption001</Decryption>
<DataFactory>Decoding001</DataFactory>
<DataProtocol>DeProtocol001</DataProtocol>
</AfterService>
</Head>
```

报文数据生成器：从报文头部中提取数据处理特征，如上述 XML 文件中 BeforeService 标签里面的内容，根据这些数据服务的服务编码，有序地调用位于本地的 Web 服务，完成对不同类型的数据源的数据的抽取、协议封装、转码以及数据传输过程中需要的不同方式地加密、压缩等处理。

传输层连接器：获取报文头部的目的节点信息（目的端授权的用户名和密码、目的 IP、目的端口、消息类型等），然后将报文数据，根据这些信息查询传输层中的路由表，获取接收节点消息队列的信息，将目的消息队列信息和报文一起传给消息中间件。如果是客户端节点，由于客户端只能属于一个平台而且任意两个客户端之间不能直接进行数据传输，因此路由表中只存有中心节点的消息队列信息；对于中心节点，路由表中记录该平台下所有客户端的接收队列信息以及其他信任平台的队列信息。

报文头部解析器：消息中间件接收到数据之后写入共享缓冲区，取出固定长度的报文转换成 XML 字符串，即为报文头部的内容，将报文头部放到共享缓冲区内。

报文数据解析器：根据报文头部中的数据处理特征，上述 XML 文件中 AfterService 标签里面的内容，有序地按照服务编码调用中心节点的 Web 服务，最终获得发送节点的原始数据。

应用层连接器：将原始数据传递给上层的应用系统或者交换引擎的调度系统。

3.2 数据服务管理器的详细设计

数据服务管理器与数据监听器紧密合作，将各种处理算法发布成服务供数据监听器调用。每个节点的服务管理器可以根据自身的需要定制服务，利用 Web Services 这种松散耦合的特性，算法可以很容易修改和扩展。

数据服务管理器中 Web 服务的总类大概分为：数据源抽取服务、数据源数据封装、类型转换、数据加解密、压缩与解压缩等。数据源的多样性决定了 Web 服务的多样性，对每一个 Web 服务提供一个服务编码进行管理。对调用者而言，提供了一个统一的服务入口，通过不同的服务编码调用即可。

数据服务管理器中的 Web 服务只能被服务所在节点上的数据监听器调用，亦即本地调用。采用本地调用这种方式，一方面避免了网络传输带来的不可靠性，另一方面本地调用的 Web Services 框架内部传输机制可以做优化，使传输效率更高，如 J2EE 开发环境下开源框架 XFire 本地调用时可以使用 In-JVM 传输机制。

4 结语

J2EE 下利用 RabbitMQ 消息中间件进行实验测试。当网络出现故障的时候，任务没有抛异常，网络恢复之后任务完成；当中心节点处理数据时间过长的时候，数据没有被多次发送。可见，该框架下数据能够被可靠地传输。对采用不同核心技术的传输框架比较如下表所示：

表 1 采用不同核心技术传输框架的比较

核心技术	可靠性	不足
MOM	可靠	对 MOM 依赖
Web Services	不可靠	不能满足数据交换平台的需求
SOAP/JMS	可靠	实现较复杂且不满足平台的需求

尽管本文提出的传输框架对消息中间件有一定的依赖性，但是其在可靠性和异构数据源上提高了数据交换平台的可用性和通用性，而且消息中间件的“发布——订阅”模式使平台能够制定更复杂的任务，增强了平台并发处理的能力。

参考文献

1 企业级 SOA 之路——在 Web Service 中使用 HTTP 和 JMS.2007–08–23.http://blog.csdn.net/steelren/article/details/1756683.

2 刘影,何克清,梁鹏,冯在文.Web 服务中可靠性消息规范的比较研究.计算机应用研究,2006,23(12):101–107.

3 曾一,陈传超.基于 MOM 的异步可靠 Web 服务模型的设计与实现.计算机工程与设计,2008,29(24):6224–6230.

4 王燕.基于可靠 WEB 服务的数据交换平台研究与实现.长沙:国防科技大学,2006.

5 Maheshwari P, Tang H, Liang R. Enhancing Web services with message-oriented middleware. IEEE International Conference on Web Services (ICWS04). 2004.