SOA 数据集成学习指南

SOA 数据集成学习指南

在 Web 服务发展初期，人们的目光都集中在应用集成和工作流程上。标准如 SOAP、WSDL、UDDI 和 BPEL 创建和企业服务总线（ESB）成为一个能够进行这种类型集成的新的技术。

SOA 数据集成学习指南将为您解答如何在一个面向服务架构 SOA 中的数据集成。本专题涵盖：数据服务、数据管理、XML、REST、数据 mashups、商业智能（BI）、XQuery 查询、服务数据对象（SDO）和 ADO.NET。

数据服务

如果 REST 是一种风格，那么以资源为导向的架构就是这种风格的一种实现，就好像如果面向对象编程是一种风格，那么 Java 就是一种实现一样。资源导向型架构就是是应用到真实世界的 REST。

- SOA 数据服务 mashups 出现
- REST 数据服务 完美体现 SOA 风格
- JBoss 之 SOA 目标：解决数据服务难题

企业数据管理

在现今包围 SOA 的所有诉求和术语中，对团体而言，最寻常的仍是寻求如何将面向服务架构集成到他们的 IT 框架中，以避免他们设计中的数据整合、处理、管理等相关问

题。他们开始学着与 SOA 并存，然而，他们经常发现与其他系统的协同工作、解决方案引起了令人觉得好奇的问题。

- SOA 数据管理
- 建立与 SOA 相关的主数据管理
- 用流程 mashups 平衡可重用性和 situationality（一）
- 用流程 mashups 平衡可重用性和 situationality（二）

数据标准

XML 是一个众所周知的数据标准。数据服务也可使用其他的标准和技术，如 REST、XQuery 查询、SDO、ADO.NET、JavaScript 对象符号（简称 JSON）。

- REST：Web 服务设计的简易性
- 如果您使用 XQuery 是否需要 SDO?
- 用 ADO.NET 和 SDO 实现数据连续性

SOA 数据服务 mashups 出现

Kirstan Vandersluis 是 Xaware 公司的创建者和首席科学家，是开放源数据集成供应商，他为建立数据服务层提供工具，其工作主要是关于为面向服务架构(SOA)应用，创建数据服务混合，本月在 JavaOne 会议上，在同 SearchSOA.com 网站的访谈中，他解释到，数据服务 mashups 和 XML 一样也是一种标准。

你能解释一下什么是数据服务 mashups 吗？

把许多资源中的数据放到一个逻辑单位中，就是一个数据服务 mashups。所以数据的方式同样也是一种 mashups。这是完全不同的球类游戏，我们将虚拟世界的任何地方的数据放入逻辑单位。并且正试图获取一束末端数据系统，这些系统非常复杂，按照 XML 模式，该系统更加合理化，有些人将一些理念注入到设计中。

当你获得了所有的数据资源，你最终会将这些数据放入何种业务功能中呢？

最后，你将拥有一个和模式相匹配的 XML 数据集，你可以调用服务从根本上取回数据。

你能举一个类似旅游公司的例子么？

对于一个旅游公司来说，你可能对客户针对旅游史提出的不同观点很感兴趣。所以，网站需要信息。这就是我们没有涉及的 GUI 块。前端应用会提出要求：“获取用户意图。

这里有用户 ID”于是数据 mashups 就成了来自许多资源的数据组合。在这里有一个 CRM 系统，因此我们要在 CRM 系统中查看基本的客户文件。我们要进入旅游管理系统获取旅游历史。如果数据返回，可能会有一些粗格式，这也许是来自 CRM 或者 API 的直接文件，旅游管理系统可能是一个主框架，这些数据会依次返回。然后这些粗数据被自动的转化为 XML 数据集。这些数据集被连到了更大的用户意图结构上。并将其返回应用程序。

在数据服务 mashups 有那些比较么，我们可以对这些数据库和数据市场做些什么？

这是我们做事的另一种方式，一种物物交换。我们肯定不是在说我们正在取代数据库。你可以在一个小的应用中使用它。这个方法带来的一个好处就是所有一切都是实时的。实时的将一个子要求传递到任意一个系统。可以一定程度上将数据库技术融入到你的数据服务 mashups 中。你可以直接查询操作系统。这在一定程度上是由应用程序的要求所驱使的。如果你需要实时数据，我们就有。成为一个抽取层也很好，因为你可以依照模式建立自己的应用。这样你就可以随着时间的推移做出变化，而不会影响到程序。应用程序依赖 XML 模式规定的合同。所以如果你一开始你将所有的数据服务都指向你的数据库，过后你就会发现那并不是实时的，你可以改变正在混合起来的数据集，因此其中的一个可能干扰了一个操作数据源。

你会如何描述那些使用工具的人的工作？它是一个数据架构么？谁真正在做这项工作？

通常，它要么是一个数据设计师，要么是一个 Java 开发商，或者是一个熟悉数据的人。当然数据源是有比例关系的，任何数据设计师或者数据库管理员在这种情况下都是很舒服的，，也许那是一个 Java 开发商，一个数据设计师经常是典型的情况。

你是在典型的数据设计师的工具箱中找到这些技能组的么？

是的，我在那里找到了。一般来说，我在人们那里得到了良好的反馈。我们从数据设计师那里得到很好的反馈，因为它是视觉化的。当我们可以使用元数据时， 我们就将其曝光。这只不过是个拉住鼠标的拖放问题。所以我认为，这完全在他们的技能可以解决的范围内，这比我们以前做的要好的多。

你认为业务方的人们理解数据服务么？当你和他们交谈时，他们意识到数据服务的价值了么？

我认为二者是不可调和的。有些人做到了，有些人做不到，我必须要说大多数人没做到，我认为这更主要的是他们理解面向服务架构的问题，这是他们使用数据服务层的典型例子。在一定程度上，他们引进了面向服务架构，他们就理解了数据服务层。我要说他们大多数并不理解。有些经常和商务人士打交道的客户把大量的精力投入到将业务方参与到 SOA 战略中，但是他们的速度太慢，以致成果并不明显。

我们在 SOA 中的数据服务的采用程度如何？你有没有从中获利或者发现一些真实的应用程序？

数据服务的采用肯定在增长，我们已经有了 50,000 下载量。这仅仅是在我们有了 GA 产品之后过去两个月的下载量。当然人们不会在在两个月的时间里将理论付诸实践。从某种程度来说一切还为时太早，但是我相信数据设计师和开发商在 SOA 方面是理解的。数据服务层应用的另一更为广阔的领域就是丰富的网络应用方面。在这个领域，使用 Ajax 的屏蔽部分被服务所迁移。从数据的颗粒性角度来说，这确实是一个不同的领域。在该领域中你需要高度的粒状数据和许多服务，而不是最终会被业务和服务所驱动并且被业务所规定的 SOA 方面。这就导致你只有一些粗研磨的服务，但是我看到它在两个领域都有增长。

这就是我们要令其更加成熟的数据服务标准，有你需要的标准么？

我们的世界主要是以 XML 模式为中心。这是关键的标准，建立和 XML 模式相适应的数据类型。这还有一些关于服务调用的标准：WSDL 和 SOAP 标准以及 REST。

那些类似保险业 ACORD 为特定行业量身订做的标准如何，这些标准足够成熟么？

我认为 ACORD 发展状况良好。保险业比其它行业采用 XML 和 XML 模式都要多并且时间更长。在 XML 公布之后不久他们就开始研究 ACORD，由于行业的不同，成熟的程度也不相同，但是保险业发展尤为成熟。

考虑到你对 XML 的关注，XML 的数据十分整齐，在你做数据服务时会因为机构没有一直遵循标准而碰到问题么？

从标准的角度来说，它是十分整齐的。你会在用户想要应用标准的用户化过程中遇到问题。ACORD 是我们尤其需要考虑的一个标准。每一个主要信息组都有一个为用户设计的扩展区域。当然，你越将其用户化，你就增加了无法和其它公司配合操作的风险。从标准的角度来说，XML 非常整齐，但是在有配合操作性的地方肯定有用户化。这些都是人们在设计应用时需要考虑的因素。

当我们跨行业使用这些程序时，例如我们需要从保险业、金融业和医疗行业获取数据时，会遇到问题吗？

不会。你可以创建和任何模式相匹配的服务。该服务可以是翻译成数据服务模式中任何一个一个模式。

在数据服务开发中能够产生问题的最坏的实践是？

不合适的 XML 设计就是个问题。有时人们甚至没有设计出一个模式。当他们开始设计 XML 实例时，并没有将其按照逻辑组织。这就使得他们不得不在整个设计周期不断的做出

改变。如果你的机构设计不是很合理，就会遇到再处理后端系统的麻烦。在你的逻辑视图和真正的物理结构之间，总会有阻抗失配。所以，你可以在保证自己的逻辑结构是合理的，你就能够避免这些麻烦。

(作者：Michael Meehan 译者：杨君 来源：TechTarget 中国)

REST 数据服务 完美体现 SOA 风格

因为 REST 风格不同于面向对象的编程风格，它是通过统一资源标志符(URIs)对所有事物命名，因此它们可以被检索，而且它们非常适合创制数据服务的应用程序，他说到。他设计了 Burton 的案例，在此案例中，他将 REST 的使用作为 Webcast 的补充。这个分析公司上个星期在 Webcast 中发表了 SOA 原则如何应用到数据存取和管理。

Lacey 一开始就解释了 REST 是什么以及 REST 如何融入到数据服务的世界。“它是一种风格，”他说到：“它不是一个技术，它的分布式计算环境的风格，作为其关键的抽象概念，促进了资源的发展。于是一个资源就变成了简单的东西，它可以被命名。可以被命名意味着它能够通过 URI 的方式被提供。”

从业务数据的角度，已经被命名了的资源可以是任何显而易见的事物，比如一个雇员，或者某种股票或者项目文件，他说到。

“但是它也能够公平地表现一些不明显的事物，比如说今天购买的第三项东西，或者说 2007 年所有的销售量，或者仅仅第二季度，或者仅仅是昨天，甚至仅仅是昨天 3 点的等等。”Lacey 解释到说。“有一种无限供给的资源定位符，你可以使用资源定位符来命名无数的资源。总之，资源是简单的数据，并且当你谈论数据服务的时候资源变得很有趣。”

他说原先最早描述 REST 的文章指出，“一个系统架构的数据元素的性质和状态是 REST 中一个关键的环节”，不像一个分布式对象系统。

“换而言之，在 REST 里面，数据是第一级的公民，”Lacey 说到，“不像其他的系统，比如 SOAP 或者 DCOM 或者 CORBA，重点放在过程上，在 REST 里面，数据才是焦点。”

这就是使得 REST 非常独特地适合于数据服务之处，Burton 的顾问说到。

为 ROA 做好准备

虽然用行话来说，承认所有人都“升级到这里”，然而 Lacey 又提供了了另外一个资源导向型架构(ROA)，此资源导向型架构是为数据服务使用 REST。

“如果 REST 是一种风格，那么以资源为导向的架构就是这种风格的一种实现，就好像如果面向对象编程是一种风格，那么 Java 就是一种实现一样。”Lacey 说到，“资源导向型架构就是是应用到真实世界的 REST。”

在这个世界上，他解释到说，所有的每条信息都拥有自己的 URI。HTTP 是用来在网络上提供数据和服务的可用性的。资源可以通过统一的接口得到。并且所有对这些资源的操作可以归结为四个非常常见的命令，获得，插入，删除，提交。

“在一个资源导向型架构中的最后一个关键的组成部分是，一切的资源都应该包含连到其它资源的链接。”Lacey 说到，“换句话说，我们提供新的发现。我们创造了一个 Web，因此当我得到一个雇员的时候，它应该提供此链接给他的经理人。它应该能够提供链接到他的部门。因此我可以通过我的资源搜索到并且得到我想要的信息。”

他说到，使用 REST 作为数据服务的概念目前正在各种平台上审定中，这些平台包括 Astoria，微软实验室早期的 beta，以及一个开源的叫做 SnapLogic 的平台。他说到，开发者可以选择使用成熟的开发框架，容易得到的有 Restlet——他认为 Restlet 是继 Ruby on Rails 之后最为成熟的。对于 Python 程序员来说有 Django 以及 CherryPy 可供选择。

Beta 开发项目也会有帮助，Lacey 说到，此项目是 IBM 公司的 Zero 项目和微软公司的.NET 3.5，这些项目中都将包括更多的 REST。在 Java 的世界里面，存在 JAX-RS，它将为 Java 提供 REST 的 API，他谈到。

“所有这些现在都可以使用了，”Lacey 说到。“现在成熟的产品已经能够用来方便地构建数据服务了。”

（作者：Rich Seeley 来源：TechTarget 中国）

JBoss 之 SOA 目标：解决数据服务难题

在奥兰多 JBoss 世界（JBoss World Orlando）会议上，红帽公司中间件事业部副总裁 Craig Muzilla 接受了 TechTarget 的采访，谈到了数据服务、MetaMatrix 和 SOA 治理等问题，深入探讨了一些常见的数据服务问题以及关于开源治理的未来目标。

您从哪看出数据问题给用户带来了具体的难题？您所看到的那个问题或者说那些问题的本质是什么？

哦，这是个大问题。有各种各样的问题。其中一个问题就是人们试图在组织中提出规范的数据格式或者标准。我给你举个例子。在一个组织机构中，对一名客户的定义是什么呢？可能会有二十个系统记录了同一名客户的部分记录。在银行，你可能拥有一个主系统，用以处理信用卡相关的信息。你可能用另一个系统来处理账户余额、支票账户、存款账户，然后是贷款数据库。这些数据库中的每一个可能对客户的定义都各不相同。因此，如果你看看数据库模式，一个数据库可能会说“用户 ID”，而另外的可能会说“用户名”，所以甚至你如何从数据中识别客户也不尽相同。

MetaMatrix 能帮助解决的一个主要问题是，帮助公司明确数据应该是什么样的，以及如何更加容易地在 SOA 环境中使用。它填补了关系数据与其如何在 XML 中或作为一项 Web 服务表征之间的鸿沟。我们看到的一个问题是在应用程序基础架构中创建更多的灵活性。一般说来，无论你是否采用 JBoss 应用程序服务器编写应用程序，你都需要在应用程序服务器和数据源之间设置不易被修改的代码。因此，也就存在一个数据库，可能是 Oracle，也可能是 IBM 的 DB2，但是基本上，你需要在很多方面将逻辑与数据库匹配。目前存在的

数据问题之一就是如何才能解决那个问题，并获得更多的灵活性。所以，如果数据库发生变化，并不会破坏我的应用程序，也不会破坏我的服务。因此，将应用程序逻辑从数据源中分离，你可以在不破坏应用程序的前提下交换数据库和数据存储。这就是我们在数据服务中看到的一些问题。

您认为这些问题有多常见？

相当常见。尤其是上面提到的问题。特别在面向服务的环境中，普遍存在一个架构问题，因为面向服务架构的整体思想是我不需要事先了解我的服务如何构成，它只是给我提供一些东西。

这就是您为什么看重开源吗？这样，连接性的问题就会变少了？

嗯，开源未必是架构构建，更多的是传送构建，即我怎么才能使软件可用。那么，我怎么创建呢？以开源方式创建，我就不会束手束脚。我可以看到代码，可以控制代码，也可以自己编写代码。无论什么时候我需要迁移或者做一些不同的事，我都不必局限于任何供应商。这不仅仅是商业构建和授权构建，更是技术构建。

架构问题属于您所看到的很盛行的问题吗？

我想其中一些问题可能已经存在好多年了。向 SOA 发展加剧了这些问题。因此，如果你对你的数据没有一个很好的理解，你就无法向面向服务的架构转变。如果客户信息不属于某项总体服务的客户，你就没有规范的格式获取客户信息，这对你没有好处。因此，在实施 SOA 之前使数据合理化非常有帮助。

我给你举一些实际的使用例子。人们会尝试着做一些事情，如创建报告程序，而当他们涉及到商业智能时，他们试着从两个或者三个不同的源中获取数据。他们需要创建常见

的格式，使用相同的数据，而不必复制数据或者创建新的数据库。这时，数据服务和 MetaMatrix 能帮你办到这一点。它能提供基础层，帮你获取数据。

根据您的经验，谁需要负责解决这些问题？

从数据角度讲，这是个很有趣的问题，因为在你讨论 ESB 和基于 Java 应用程序的服务器时，那是整个公司和开发团队的高级架构。在你谈论数据时，它又不一样了。当然，也有一些具有架构团队的先进公司比较关注这些数据问题。但是，通常还有一个独立的团队，称为数据管理团队。他们需要负责数据仓库、商业智能工具，很模糊，就是一个数据领域。一些在公司里管理数据库的数据库管理员也有别于开发人员。因此，数据领域真的跨越很大，到底谁要负责任也不明确。

在您看来，应该由谁负责？

从架构角度讲，通常是高级架构副总需要关注你如何编写应用程序、如何整合程序。同样，还要关注获取数据的最佳方法是什么？如何构建数据？你需要几个数据库？数据库应该是什么样的？这个人最终应该负责任。

只是一个人吗，而不是开发团队吗？

当然每个人都有作用，但是如果你看待一个具有架构基础的领域，通常是一个团队负责，这个团队需要跨越应用程序开发世界以及数据库和数据管理世界。

ETL 软件在哪些方面不能解决这些问题？

ETL 是一项数据迁移技术。从一个数据库中抽取数据，迁移到另外一个数据库中，从而实现转化——从 A 取出，使其看起来像 B。这并不抽象，这是从多数据库中实时利用数据，但并没有真正地实时利用。而是载入数据仓库的批处理过程。所以，像数据服务技

术、MetaMatrix 等技术实时提供数据。不是迁移数据创建数据仓库，而是帮你不必再创建额外的数据标志、复制数据，你可以得到原始的数据源，从这些数据源中获取数据。因为我有不同的应用程序，不同的应用程序需要如此，ETL 帮你建立二十个新的数据库，但是现在你有二十个不同的数据库管理额外的硬件。

数据服务技术所做容许你忘记复制数据，在原始数据上覆盖抽象层，使其看起来需要寻找利用数据的应用程序，而不必创建另外一个版本。这仅仅是一个数据源。

MetaMatrix 代码公布的最后期限是什么？

我们今天发表了一个声明。这是声明的一部分。MetaMatrix 有三个主要的组成部分。运行时间能帮助你获取数据，帮助实现转化，帮助运行分散式查询。IDE 或者开发工具能帮助你模拟数据，因此你能在运行时间内实现转化。第三部分是元数据管理系统。元数据管理系统就是你可以存储模型、存储数据的地方。元数据看起来像什么？我们在讨论的 DNA 项目就是元数据管理系统。有了 MetaMatrix，元数据管理系统就只是在数据模型中使用。我们所做的就是携带元数据管理系统，在任何与 SOA 相关的存储库中加以采用。其技术基础令人难以置信，因为元数据管理系统采用该项技术远远超过 MetaMatrix。所以，采用开源的第一部分就是元数据管理系统。我们不久后将会就产品线路图的其他部分发表声明，那时这些都已成为开源模式，在未来的 10 至 12 个月，这些也会变成开源。

治理的想法存在已经有一段时日了，您认为企业将要多久才能理解开源治理的必要性？

将来几个月，我们会让其他人下一个定义。

其他供应商吗？

是的，我们已经与一些供应商谈过，包括我们的合作伙伴及其他对此感兴趣的供应商。我们将与其他供应商和终端用户对话，明确 SOA 治理的定义。有哪些任务呢？从那时起，一些软件解决方案就会紧随其后。要让每个人都理解这一点可能需要好几个月的时间。DNA 项目已经启动，我们将与其保持一致。那将是如何定义治理的一部分，然后就会出现众多项目。我们需要一个注册项目、一个政策管理项目，我不知道会是五个项目还是十个项目，因为这要根据定义而定。

所以，这个焦点会持续多久？

可能到明年或者更久。我们可能会基于这些项目引进一些产品，新项目出现了，我们就会扩展产品。我想可能需要 2 至 3 年的努力，但是随着存储库、或者注册之类的明确概念出现，定义可能还会拓展。

（作者：Brein Nally 译者：周姝嫣 来源：TechTarget 中国）

SOA 数据管理

在现今包围 SOA 的所有诉求和术语中，对团体而言，最寻常的仍是寻求如何将面向服务架构集成到他们的 IT 框架中，以避免他们设计中的数据整合、处理、管理等相关问题。他们开始学着与 SOA 并存，然而，他们经常发现与其他系统的协同工作、解决方案引起了令人觉得好奇的问题。实际上，这些问题使人觉得系统和 SOA 组件之间的交互作用很烦扰，并造成了时间消耗。

关键是识别无论存放在何处，也就是不管是在 SOA 的保护之内还是之外的团体的数据值，找到使它们能够截取数据的方式，以及将制造商和用户之间数据传输过程中的混乱和繁忙降低到最小程度的方式。这引发了 Gartner 称为 SOA 的面向数据服务视图，该公司预言此视图必将成为 SOA 即将发展的重要组件，同时现有视图的一次重要升级的失败使此视图得到应有的重视。

用 XML 来生成元数据和数据表示法，然后构造 XSLT 应用将其输入输出到 SOA 组件，在此过程中开发者获益良多。

他们创建了保存键数据元素的方法，相互作用和语义不仅使得 SOA 组件间或组件中的数据移动简易化，还为他们所用的数据(和他们所需要的元数据)的关键理解和假定提供文件资料。

这很大程度的提升了与系统相关的每个人的数据透明度，使数据路径和数据关联变得清晰许多。还指出了哪儿的问题与数据一致性、数据重复有关，或开始出现标准化和缩减的需求。

当新的商业需求浮现、新的制造商和客户这些数据加到图中来时，组件之间和组件中清楚的数据流抽象视图、数据流类型和规模也给他们提供了重新调整方向的机会。

XML 和如 SOAP 的通讯协议确实使数据提取和数据移动变得容易的多。但是它们也提高了以下问题的重要性：哪儿存放数据，怎么获得(或保留)适当的上下文，怎么联合它所比拟的现实世界信息来检验特定的语法、语义和正确性。这引起了一些专家称为“以 XML 为中心的混合数据服务”，以处理原始的使 SOA 环境中的数据有效需求如数据存取、集成、转换(联合、提取、过滤等)、验证和确认、特性控制和管理。此方法使得组织可以以 XML 为中心的、数据驱动的工作流设计工具，这些工具将用以 XML 为核心的工作流引擎来部署和管理。

对于团体而言，重要的事除了使用正确的工具、商务规则之外，还需认识到基于 XML 的描述只有在设计它们的人使其运转时才能运转。这意味着能干的、有效率的数据架构师和数据管理专家必需投入到创造正确的表示法和截取、验证、维护和分配数据的步骤中。除 XML 描述、面向 XML 数据库、XQuery,和一个形式数据服务方法来处理 SOA 框架中的数据之外，应有一个专门的数据专家和管理员团队来确保所用与所需一致，并能随着时间流逝保持数据仓库的干净整洁、不断刷新和正确性。

(作者：Ed Tittel 来源：TechTarget 中国)

建立与 SOA 相关的主数据管理

自 20 世纪 90 年代以来，内部业务应用程序开始从两层的架构模型，转向更为分散的 N 层架构模型。目的很简单：提取出表现层中复杂的数据层，实现其灵活性。这样做的结果是，不需要对整个程序进行大规模的重新设计，数据模型就能够支持新特性和新业务功能。看到这一优点，设计师们很快意识到他们可以将相同的提取工作应用于业务逻辑，并已经开始将越来越多的逻辑从表现层转移到应用层（图 1） 。

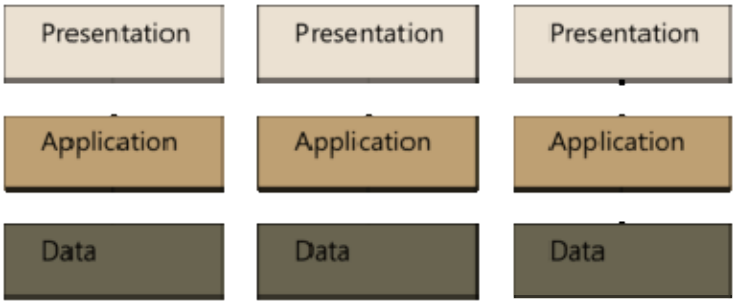


图 1：三种不同的分布式解决方案，每一种都建有自己的程序孤岛

随后，垂直的业务应用程序孤岛打乱了 IT 现状。这些程序孤岛通常就是复写业务逻辑和复制数据，从而导致大量冗余事件的普遍出现，给 IT 企业造成了不必要的负担。

为了减轻这种负担，当前的分布式技术（DCOM，RMI-IIOP）被视为一种使复用成为可能的手段。虽然这种技术有时在设计相似的系统下运作良好，但它却无法成为真正适合企业的解决方案。看到因在异构企业之间共享业务逻辑而带来的技术挑战，技术人员对 EAI 解决方案做出改进，跨越技术所有权和程序所有权限制实现了数据的抽取。

大约在 20 世纪 90 年代末，开发团队发现互联网标准协议具有互操作性，并开始利用这些协议处理企业解决方案中的问题。许多项目最初采用简单原始的 XML（POX）Web 服务，随后又采用 SOAP Web 服务。当时，数据已经遍布于企业内部，单一逻辑的业务实体属性存储于多个系统中。

从最初一种发展优质服务的范式，SOA 迅速转变成一种从复杂的整合/持久层提取出业务流程层、或从技术模式提取出商业模式的方法。现在，我们看到 Web 服务所具有的作用，看到后端复杂性的消除，看到服务生命周期管理（SLM）的采用，就能够很好的理解企业 SOA 现在所具有的涵义。不过，由于开发团队是在整个企业内部创建抽象层，所以他们经常会遇到与数据有关的难题。

虽然开发团队通过迅速采用 Web 服务节约了时间，但是他们却没有时间确定读取或写入的某种属性是否来自正确的记录系统。即使团队清楚的知道从哪里能获得可以采信的数据，但他们还是经常会遇到一些问题，包括存储的整合数据使用的关键字不同、多余的属性彼此不一致。企业组织应当要认识到 SOA 基础设施的开发是件困难的事情。效益的实现要依赖那些利用该基础设施的后续项目。不过，面临的难题却往往很难描述，而且最终可能会损害 SOA 项目。但所幸的是，这些难题的解决方案都可以在称为主数据管理（MDM）中找到。

MDM 是将分布式企业数据规范化、合理化，并且表面上将其集中化。MDM 这个名词并不新鲜。自从开发出关系型数据库管理系统（RDBMS），企业组织就曾试图建立一个中央数据储存库。中央数据储存库通过修改现有程序和开发新程序以利用共享数据库，而 MDM 则允许数据孤岛的存在，而且任何试图彻底消除数据孤岛的尝试都将失败，因为这项工作太复杂，或数据孤岛消除后可能不够灵活。

举一个典型的例子

假设一个机构中有：

• 储存在 ERP 系统中的用户账单信息

• 储存在自定义 LOB 程序中的用户订购数据

• 储存在 CRM 系统的呼叫中心中的用户交互数据

图 2：三种与用户有关的数据，分别存储在三种环境中。

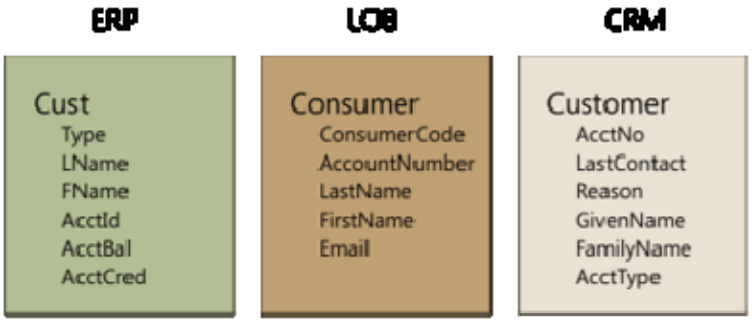


图 2 显示重复的数据具有附加属性，支持各系统的特定功能，而最终存储于每一个系统中。而因为相同的数据往往会被不同的名称、或包含不同数据的相同逻辑属性所引用，所以这一情况变得更为复杂。举例来说，ERP Cust.Type 领域面向商业用户的代码中可能含有“BUS”，而 CRM Customer.AcctType 领域面向商业用户的代码中与之相对应的则可能是“Business”。因此，在三个系统中要想找出相同的实体很困难，而且如果主键不同的话还需要人为的干预。在大型系统中，由于拥有许多类似的记录以及很多不同质量的数据，这项任务将变得更加困难。让我们看看你找到的这些记录：

• John Smith

• Jon Smith

- Jonathan Smith

- Jon Smyth

你可能不仅无法识别相同的实体，而且更有可能在识别相同的实体时出现错误。当一个系统更新时，EAI 或 ETL 进程频繁的同步更新其他系统。在大型 IT 企业中，这种同步状态可以发展成一种复杂而灵敏的控制体系，以至于业务实体的属性发生的任何改变，对于业务实体本身来讲都将成为一项艰巨的工作。

新系统将不断试图寻求用户资料读取的高效率。如果一个交互语音应答（IVR）解决方案正在开发中，比如说，为了查看订单状态和帐户余额，那么它就需要访问储存在 ERP 和 LOB 程序中的数据。但是，呼叫中心的顾问却想知道是否正是因为来电者一直在使用 IVR，而导致无法完成期望的开发工作。

这一系统的开发人员会问这些问题：

- 哪一个系统才是记录系统？

- 如果没有一个明确的记录系统，我们是否应该立即更新所有系统？

- 如果立即更新所有系统，那么在分布式事务处理中，我们应当怎样做？

- 这将如何影响系统的可用性？

- 这将如何影响系统的性能？

在 LOB 系统中，在哪里可以找到用户数据？

服务层的作用就是为服务面向的用户消除这种复杂性。然而，对于服务供应商来说，企业数据层的复杂性需要进一步得到重视。不仅服务供应商需要排除整合过程中的障碍，而且企业也必须要相信执行的逻辑是正确的，返回的数据也是正确的。

因此，要提供可信赖的服务就需要采用新的技术，可以立即应用现有的概念、技术及进程。举例来说，在用户数据中，在系统之间会重复使用帐户余额，这很常见。然而，在大多数情况下，公司只相信一个系统中的帐户余额数据。因此，如果开发人员已准备进入到另一个系统中，服务面向的用户间则可能会显示不一致的数据。当这种情况发生在项目中时，整合开发的效率无论怎么提高，都会被企业主要求的数据映射过程抵消掉。

而当 SOA 继续与创建复杂、异构、分布式 IT 环境的技术决策者们交流思想时，有效管理数据的能力才是提供可靠服务的根本。

主数据

主数据的建立是首要目标。其目的是建立一个高质量的数据存储，储存已汇总、映射、规范化和标准化的企业信息。通常，我们将主数据建立在一个新的数据库中，因为现有的系统状态中命名约定并不理想，而且数据质量可能不高。在某些情况下，系统可能需要具有足够的质量而发挥主数据的作用，而且可能需要添加只有在其他系统中才能找到的属性。

数据整合

企业信息有机地散布在整个组织结构中。虽然不同的系统拥有相同的核心属性，但是通常都会有独特的属性，支持系统的特定功能。要想真正从企业的角度认识核心业务实体，比如说用户，那么就必须识别出独特的属性列表，并将其储存，如图 3 所示。

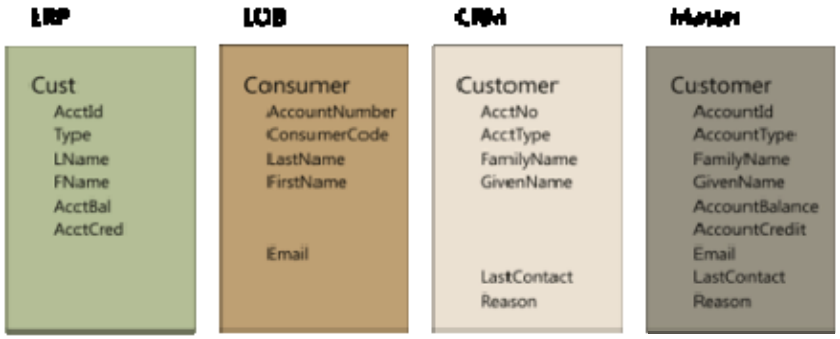


图 3：一份扩充的用户属性名单

在这种情况下，用户主数据要包括 ERP、LOB 和 CRM 系统中的共同属性，以及每个系统中各自独特的属性，如帐户余额、帐户信用额度上限、电子邮件、上次交易日期、交易原因等。

数据映射

人们经常发现，如果不主动对数据进行管理，数据就会独立的发生变化，导致表名称和列名称不同，以及命名惯例异类。数据映射的任务之一是识别出哪些列代表所有系统中相同的实体属性。一个原始的例子是 surname, last name 和 family name 它们都指代相同的信息。

数据规范化

不同的系统必然具有不同的数据质量，彼此互不同步。主数据应代表 the single point of truth 单点的真相，必须可以信赖。数据的规范化使得电话号码、社会安全号码及邮政编码具有一致的格式。同时也对地址进行有效性验证，并按照邮政标准统一其格式。

数据标准化

经过适当的映射和规范化过程后，数据元素在整个系统环境中仍然可能未保持一致，状态代码的管理方式也可能不同。举例来说，交易日期可以在系统中完成映射，并且格式一致，但代表的是不同的时区。（日期和时间应该根据格林威治标准时间标准化，同样，状态代码也需要标准化）。

主数据应该能提供一个关于业务实体的聚合视图。而且要注意，有一些属性只针对某一特定系统，这些属性并不一定包含于主数据中，因此也不需要将其纳入数据清理和数据移动过程。

这不仅会产生用户数据的聚合视图，而且也能通过规范电话号码格式、验证地址有效性、按格林威治标准时间规范时间格式等提高数据的质量。既然主数据已经确定，下一个目标就是确定拓扑结构和合并策略。

确定拓扑结构

主数据及组成系统的可用性和性能特点，将在很大程度上影响拓扑结构。有一点很重要：参与的系统可以直接使用主数据表；或者主数据表可能只是作为企业的数据中心，而因此应该是透明的，可为现有的系统所调用。中央拓扑模型是最简单的结构，因为它只需要主数据的一个复本。

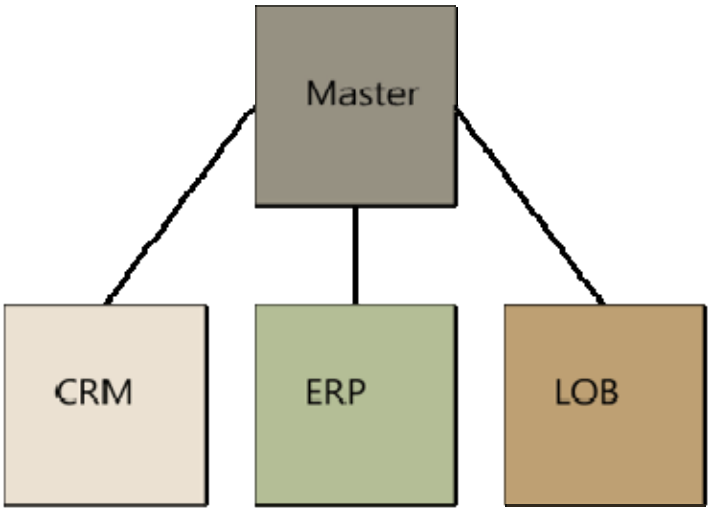


图 4：集中式拓扑模型

分布式拓扑模型能够将主数据的本地副本部署到每一个组成系统中。这种方法能够分散主数据承担的负荷，而修改后可直接获取主数据的系统，由于要存储本地数据其负荷则可能会增加。

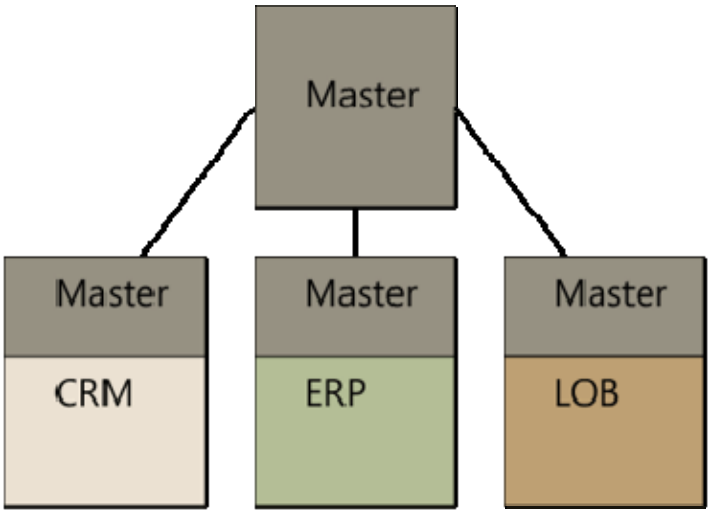


图 5：分布式（分散）拓扑模型

对于分散模型，人们普遍关心的是主数据的多个副本所需的存储量。为了解决这个问题，可以将每个系统设计成只订阅主数据的某一个子集，从而仅涉及与该子集有关的程序。

确定合并模型

在主数据管理解决方案的实施过程中，对于其长远的成功和维护工作来讲，一个有效（或无效）的合并模型所造成的影响至为重要。

确定合并模型时易犯的两个常见错误：

1．认为所选择的拓扑结构应该要能推动合并模型。

2．认为编写工作应该要始终面向中心主数据存储。

举例来说，部署分布式主数据并不一定意味着每个系统都能更新各自的本地数据副本。但是，在某些情况中，要求对主数据进行本地更新，并要求将该更新传递给所有参与的系统。

所有这一切都需要从识别可更新的节点开始。可更新的节点可以只有一个，也可以包括所有参与的节点。如果只有一个可更新的节点，则该节点可以是参与节点中的任何一个。

让我们来看一看节点的选择：

可更新的单节点——中心主数据

只要有可能，服务就应该直接面向主数据进行读取和编写，但这不可能总是可行。如果选择这一种节点，参与的系统可来去自由，而只受到很少的干扰或根本不受到干扰。而因为数据的良好结构，服务的开发效率也能得到提高。

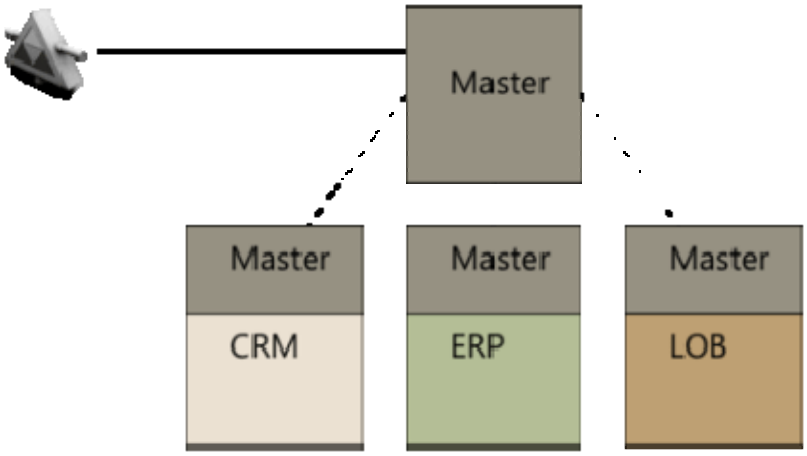


图 6：可更新的单节点/中央主数据模型

可更新的单节点——本地主数据

这种方法常用于那些经过修正而能使用本地主数据的关键程序。一个典型的例子是呼叫中心代理所使用的用于进行调整的结算系统。利用本地主数据可以提高性能，而且企业可能只会信任程序中的业务逻辑来完成更新。

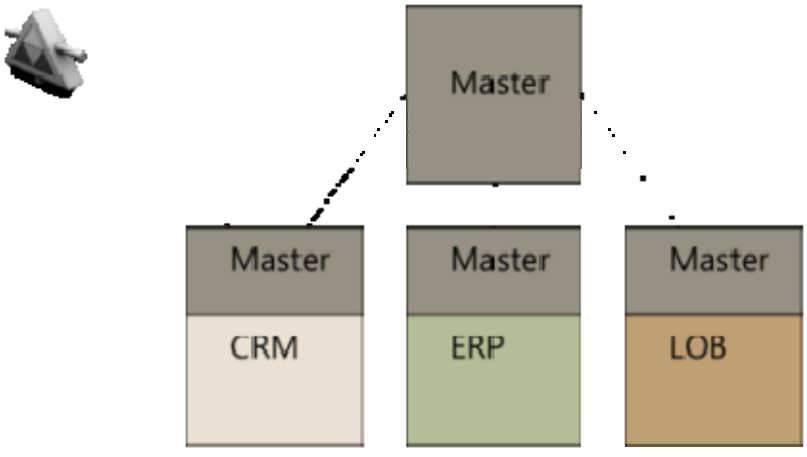


图 7：可更新的单节点/本地主数据模型

可更新的单节点——本地数据

这一方法常用于那些没有经过修正的程序，以及通过 ETL 过程与主数据明显参与的程序。它通常与使用本地主数据作为可更新的单节点的情况类似。

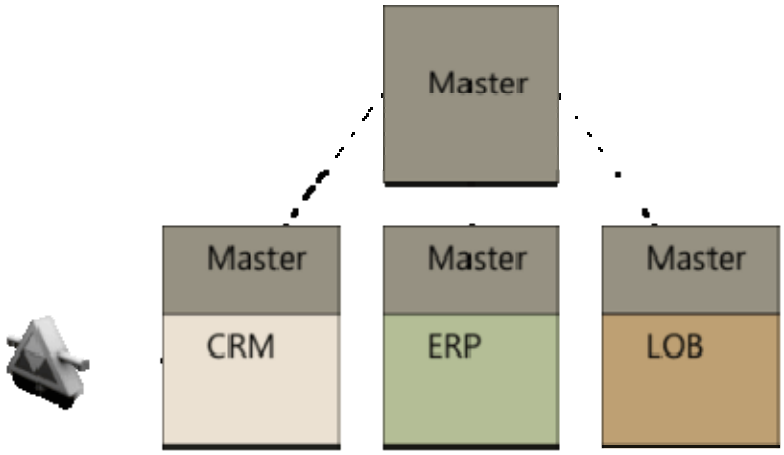


图 8：可更新的单节点/本地数据模型

可更新的多节点

当存在潜在的合并冲突，而该冲突必须以手动方式解决时，一般在合并过程中使用这种方法进行描述。虽然合并冲突可能会发生，但是此方法只需要对参与系统做出最少程度的改进。

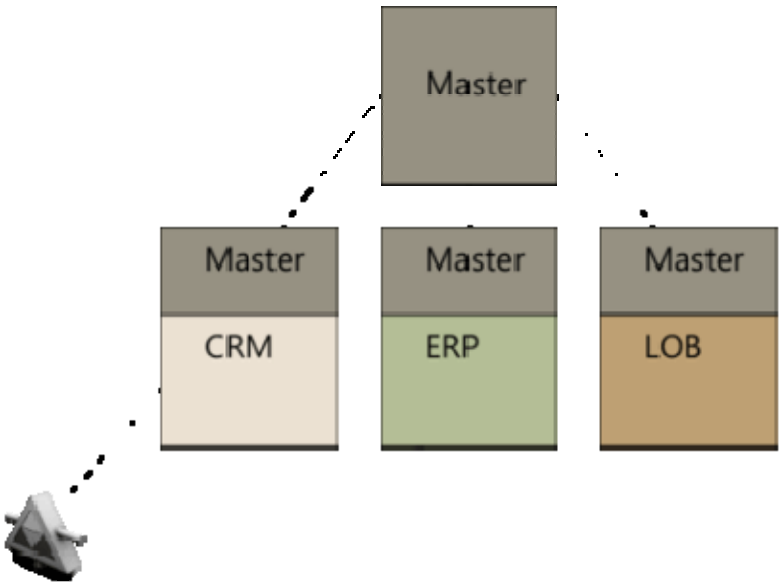


图 9：可更新的多节点模型

在与本地主数据（或不是单个或多个更新节点）有关的任何情况下，主数据作为一个枢纽，保持各参与系统的同步状态，并储存整合后的、规范化的、标准化的数据。这些系统虽然可使用相同的技术移动数据，但是没有这个必要。CRM 系统可以利用商务智能（BI）中的技术，并使用 ETL 工具。ERP 系统可利用行之有效的方法转移 FTP 文件。较新的 LOB 程序可能会收到更新通知，就像在 BizTalk 等 EAI 工具中发生的一样。与大多数架构的确定情况相似，选择的方法将取决于独特的程序、可用的工具及队伍的技术能力。

EDW 的考虑

强大的主数据也将简化企业级数据仓库（EDW）的工作，因为 EDW 从主数据中加载的数据，可以自身实现数据汇总、数据规范化和标准化。虽然 EDW 和 MDM 解决方案提供了一个业务实体的企业视图，利用了许多相同的概念和技术，但是，与 MDM 解决方案相关的模型却是事务型的而不是空间型。

结论

面向服务因为其能消除可操作界面、LOB 程序及业务伙伴的后端复杂性而得以迅速的应用。应用 Service Fa?ade 模式可以实现运行和维护过程所需的后端优化。MDM 是该优化过程中的一个辅助技术。通过后端优化，能够轻松实现对现有服务的维护，并使其运行更为出色；同时能够轻松实现对新服务的开发，并可避免费时长久的数据映射过程，使其成为业务可信赖的服务。

（作者：chris Madrid 译者：Eric 来源：TechTarget 中国）

用流程 mashups 平衡可重用性和 situationality（一）

面向服务架构（SOA）的核心是面向网络架构（WOA）以及以业务为中心的 Web 2.0，我们称其为 Enterprise 2.0，这是对企业 mashups 的称呼。它被定义为以 Web 为基础的用户接口环境中被治理以及被管理服务部分。尽管这样的应用在企业中范围相对有限，但企业 mashups 仍然可以成为 SOA 的主要驱动力量。

实际上，是 mashups 限制了这些程序的应用范围：使用一个 mashups 需要建立一个 mashups，换句话说，就是让业务用户管理 mashups 的功能即准许用户在使用 mashups 的过程中创建应用。这个想法听起来不错，但是 IT 真正希望程序使用者建立并修改这些程序的时候有多少呢？

像这种“使用即创建”的应用，我们称其为情景应用。情景应用是为了满足一部分人群短期需求而专门设计的应用。短期内在某个特定的情景下可能会很有用，但是不会产生长期的效益。许多情景应用都是数据 mashups，通过把不同的数据源信息混合在一起，用户在 mashups 中创建应用以便满足特定的业务需要。

情景应用和数据 mashups 的结合会影响企业 mashups 的全部力量，在 SOA 环境内编排服务的关键就是要支持灵活的业务流程。数据 mashups 主要是一个将数据混合在一起的流程：要保证其是一个有用的流程，不过它只是企业想要自动化的众多业务流程中一个有限的实例。我们可能把这样的流程看做是一个“流程 mashups”，而不是一个数据 mashups。那么什么是流程 mashups，你为什么需要流程 mashups，在企业环境下你如何实施流程 mashups？

一个业务 mashups 实例

ZapThink 在 Licensed ZapThink Architect 课程上一直在谈论流程 mashups。我们在课上使用的是呼叫中心应用这个例子。由于“swivel-chair 集成”问题，呼叫中心也是面向服务业务流程（SOBAs）诞生的地方，它是用于实施业务流程的的一整套服务组合。传统呼叫中心的代表需要访问多层系统才能完成工作，这些系统主要包括主机绿光屏，用于客户关系应用的客户/服务器接口，用于公共门户或者网站的网络接口，以及为手机系统量身订做的接口。为了满足呼叫客户的需求，CSR 必须在两个屏幕之间不停旋转，这种方法很费时间，而且容易出错。

当然，使用 SOA 效果更佳，通过将应用抽取为服务，呼叫中心可以为 CSR 建造一个流线型的灵活接口，这样 CSR 就可以解决用 swivel-chair 集成问题了。实施客户服务 SOBA 支持这样的接口非常简单，现在许多呼叫中心都实施了 SOA。

实施 SOBA 只是其中的一部分，因为 CSR 只是这个阶段的一个部件。如果我们看一下其它组件，就会发现一个更为复杂的应用布局图，正如如下的图表所示。

运转中的流程 mashups

上面的图表向我们展示了单独的几个人是如何以不同的方式和客户服务 SOBA 发生交互作用的，这主要取决于它们的作用：

• 执行程序（1）会对呼叫中心的关键效绩指标感兴趣，即呼叫时间（越短越好）客户满意度（越高越好）。这样关键效绩指标就可以通过管理仪表板和 SOBA 发生交互作用，通过管理仪表板，人们可以看到这些关键效绩指标。

• 呼叫中心管理员（2）负责处理呼叫中心执行的业务流程。换句话说，呼叫中心管理员负责 CSRs 所执行的脚本，并且要注意 SOBA 是如何支持脚本描述流程的。管理员同时

也要负责 KPI 仪表板，以及 SOBA 实施的流程流只读窗口。基于老板的意愿，管理员还要适时地调整呼叫中心流程，并将这些变动上报给业务分析师。

• 在这个实例中，业务分析师可能是唯一有能力改变 SOBA 性能的人。换句话说，业务分析师是呼叫中心拥有 mashups 接口的人。按照管理员的指令，它会直接在 SOBA mashups 的接口中改变 SOBA 的性能。

• CSR（4）有可能是这个程序最高级的用户，因为在运行不同呼叫中心脚本的同时，它要求 SOBA 的全部功能都要满足客户的需要。应该不会改变应用的功能。

• 但是，用户会和 SOBA 发生交互作用，这是因为它支持共有网站（5）或者是因为他们直接通过手机呼入了呼叫中心（6）。用户在网站上的阅读和写入权利是有限的，但是这仍然不能改变应用性能。

（作者：Jason Bloomberg 译者：杨君 来源：TechTarget 中国）

用流程 mashups 平衡可重用性和 situationality（二）

运行中的业务授权

这个实例告诉我们企业 mashups 的意义是什么？以及流程 mashups 的特性。首先治理在这里发挥了重大的作用。实际上业务授权需要治理。因为 IT 不会为流程提供强大的工具，除非你可以灵活的管理这些工具的使用情况。

其次，业务分析师的企业接口是针对某个特殊用途而设计的，作用有限，只有一小部分人拥有 mashups 接口，他们可以通过这些接口做出改变。在这个实例中，业务分析师被纳入了呼叫中心小组，而在实际过程中，这个认为可能是 IT 的，尤其是 mashups 工具技术性极强时。理论上讲，mashups 是为业务为中心的知识工作者而服务的，但是其可用性取决于这些工具。

第三，需要注意的是，这个程序是一个流程 mashups 而不是数据 mashups，因为该程序的目标是支持呼叫中心流程。从技术角度来说，业务分析师修改程序所采取的步骤本身就是一个流程，这个流程不是程序的关键，流程和数据 mashups 的关系才是关键。换句话说，这个 SOBA 的 situationality 要比典型的数据 mashups 小，每次用户和工具发生交互作用时，这个 mashups 流程都会发生变化。

因此，mashups 最重要的不是 situationality，而是可重用性。毕竟我们没必要多次运行一个流程，所以我们不用最先将流程自动化，否则自动化就不划算了。

situationality 和可重用性是一个领域的两个极端，从我们的观点来看，这个流程则处于中间位置，这个流程具有一定的可视性需要测量实施这些流程的程序situationality。就是这些流程计算这些流程 mashups（总的来说就是 SOA）。

ZapThink 采取的措施

同流程 mashups 相比，支持数据 mashups 所需的工具和基础设施是不尽相同的。因此，二者明显在技术角度划清了界限。也许，业务人员对二者的区别不是很清楚。毕竟，从业务角度来说，流程通常会涉及信息。如果不能使用 mashups，很难从信息中获取价值，当业务运转时，这个活动就可以构成一个业务流程。对于业务用户来说，这个授权的意义远比数据和流程之间的差别要大得多。

不管是数据 mashups，还是流程 mashups，企业 mashups 在全局中的意义重大，正处于这场完美风暴的风眼，而现在的这场风暴就是业务授权，业务灵活性以及业务流程。对于机构来说，有这么多技术细节，实在很难保证不亏欠。但是，为了了解 IT 是如何在 SOA 启用的企业 2.0 环境下提交业务值，我们必须要对企业 mashups 的多样性和功效有所了解。

（作者：Jason Bloomberg 译者：杨君 来源：TechTarget 中国）

REST：Web 服务设计的简易性

Web 服务的请求者和提供者之间的连接给双方都带来了大量的工作，其中包括像同意实现业务功能、认同技术合同细节这样的事情，当然还包括从服务到应用程序的宏伟计划的整合。但是，如果过于经常使用标准化的 SOAP/WSDL 方法，这只会使这种情况变得极度复杂。

还有一种办法，就是部署被称为 REST 的 Web 服务兼容架构，它代表“表述性状态转移”（Representational State Transfer）。REST 是由 Roy Fielding 于 2000 年在其博士论文中所命名的一种技术。这种技术与行业中的主要规范对立竞争，虽然他是否有创造这一技术的先见之明还值得商榷，但是自从那时起，REST 的简称在网络服务设计的舞台上，已经扮演了主要角色。

是什么使 REST 不同于标准化的 SOAP/WSDL？是 REST 的简易性。它抛弃了后者必需的一些冗繁的要求，而这些要求在实现 Web 服务的基本功能时，在许多情况下都被证明是不必要的。

为了更多的了解 REST 的理念，让我们选取 SOAP/WSDL 设计中一个典型的过程：

1 - 提供者在现有的应用程序上创建并实现 Web 服务接口。

2 - 提供者创建 WSDL 契约，从而将 Web 服务细节提供给潜在的用户。

3 - 用户获取 WSDL 契约以便利用该契约。

4 - 用户在特定的平台——Java，C#，PHP，Perl——上执行 WSDL，并且创建通讯程序。

这个过程中，最大的困难在用户身上。他不仅需要按计划提取 Web 服务的有效载荷——例如 SOAP 消息——利用其内嵌于程序做一些有用的事情，而且也需要按计划通过 WSDL 发出起始服务请求，以取得有效载荷。如果你像许多人一样，在不同的情况下都经历了这个过程，也许你会认为这是处理 Web 服务时已存在的一个事实。但是，这个过程往往能证明是不必要经历的，所以我们不如退一步，探寻 Web 服务的实质。

许多 IT 部门的待议事项上都有 Web 服务，其主要原因很简单：为了在不同系统之间实现互操作性，例如一个与 Java 应用程序通信的 SAP ERP，或一个连接到 a.NET 新闻服务的 PHP 网站。了解了这一点，让我们分析一个非常普遍的服务——Atom 和 RSS 新闻提要，它完全符合这一解释，而且其核心也是 REST。

新闻摘要，最初只是在博客中的一个构想，而现在几乎在每一个主流网站上都有应用。它允许提供者发布网站更新，而且允许用户按需获取这些更新。在后台，提供者的站点可以建在 Java、PHP 或 Perl 中，同时提供一个 RSS 格式或 Atom 格式的 XML 有效载荷。随后，这些更新就可以被任一种语言写成的程序所使用。

如果你在网上迅速一瞥，就可以看到新闻摘要充斥在每个角落。有许多应用程序（用户）都从网上收集 RSS 数据源，从而提供丰富的内容，而这些数据源都来自第三方服务。

这个过程很简单，而且相比于 RSS/Atom 服务仅仅提供阅读功能来说，它实际上更为灵活。由于 REST 过程是在 HTTP 环境中执行，其所有的参与人员——get、post、put 及 delete 的操作人员——都即将实现需要实现的任何商业功能。

如果你还是对 REST 理念的含义有点半信半疑，认为它更学术化，或者认为 RSS 和 Atom 新闻信息是一个孤立的情况，那么让我们再看看一些使用 REST 的生产型应用程序。

REST API 允许开发人员录入众多的信息来源，从而将相同的数据整合入第三方应用程序。雅虎的 Web 服务产品也是基于 REST 建立，以至于他们目前甚至不提供 SOAP 支持。雅虎只使用 REST。

看了上述几个例子，你应该能感觉到 REST 开发的 Web 服务背后有强大的社会支持，但是别急着下结论。就像所有的技术一样，REST 也有其缺点。如果在 REST Web 服务中有一个共同论题的话，那就是它们处理的数据都具有的非必要性。

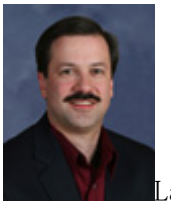
通过应用程序传输较为敏感的数据，如财务数据或采购信息，需要使用更先进的特性，比如业务交易，故障恢复功能和服务质量。而考虑到 REST 的简易性，这项工作相对来说就比较复杂。但这并不是说这一情况不能通过 REST 实现，而是说根据这类企业的需求，SOAP/WSDL 方法更为有利，更适合他们。

SOAP，通过其一系列的配套规范，如 WS-政策，WS-安全性，WS-可靠性和 WS-业务交易管理等，清晰详细的解释了如何从问题刚出现时就解决这些问题；而在 REST 中，这些问题被整合进程序中，变得更为复杂。

虽然 REST 不是适用于每一种情况，但是它可以使 Web 服务的创建和交流工作简单很多。鉴于其在主要门户网站中广泛的应用，以及其简洁实用的方法，在探索未来 Web 服务设计的事业中，这是一个不得不采取的方法。

(作者：Daniel Rubio 译者：Eric 来源：TechTarget 中国)

如果您使用 XQuery 是否需要 SDO?

 Larry Fulton 作为 Forrester Research 分析师是研究企业服务架构的专家。他精通面向服务的架构（SOA），SOA 治理，enterprisearchitecture 和企业中间件。Larry 主要研究领域是 SOA 治理，SOA repositories，和企业服务总线（ESB）。他是 SearchSOA.com 的数据服务专家。

问：我使用的 XQuery 是否需要研究服务数据对象（SDO）？如果是，为什么呢？

答：在一般情况下，假如您所使用的技术能够满足您的需要，这是在没有直接的或可预知的危险，我会说，“为什么不继续使用它”？尽管，这两种技术提供基本相同的功能。如果您是追求面向服务架构（SOA）和更具体的服务组件架构（SCA），这是一个标准的创造集合的多重服务，您可能会需要了解一下 SDO，特别是通过 SCA 达到你与第三方的合作。

在这一点上，XQuery 是非常的主流，SCA 和 SDO 的支持，真正的形成。另外一个需要考虑的是，无论是否对任何开放原代码产品起作用，尤其是 SDO。不过，我不能说是否属于或没有这种情况。

(作者：Larry Fulton 译者：娜娜 来源：TechTarget 中国)

用 ADO.NET 和 SDO 实现数据连续性

面向服务架构从设计到实施再到测试，在这个过程中一些系统和服务之间不可避免的会产生代沟。可能是因为机构数据间旧基础设施元素二者分离时间太久，也可能是因为设计变化企图在系统和服务之间进行数据交换，或者参与数据流。

通常在这种情况下，我们会用传统的方法创建一个客户程序。这个客户程序理解数据表述，交易语法，并且能够在双方缺少联系的要素间提供通信功能。这样就可以把那些先前分离的要素联系在一起了。这个方法很管用。但是，许多人都认为他们很了解交易双方，实际上，他们只了解其中一方。因此程序的功能也受到了影响。此外，还有系统维护和升级这样更为令人头痛的事宜：当程序和服务必须维持互连变化时，二者之间联系的密码也要随之发生改变。IT 部门越想弥补二者之间的差距，就越难建立这些程序。我认为“将来总会出台一个更好的办法来解决该问题”。但愿，读者的想法和我一样。

事实上，确实有许多更好的方法，但是几乎所有的这些方案都涉及到了 XML。这是因为，XML 能够出色地捕获数据表述，转换内容，并处理复杂句法，使这些方案能够在应用和服务之间建立起沟通的桥梁。有时在 SOA 维护 polite fiction 的同时，所有操作者的连通性也会得到维护，但是这种情况并不多见。

例如，考虑一下业内的移动销售和服务力量，它们能够愉快地接受命令，处理现场活动，业务原则必须确保所有的数据同步，但是通信只有在业内一些员工使用时才会发挥作用。另外还有一个普遍的情况就是，统一性是一个只有当多个供应商的系统和数据库相互发生作用时才产生的 polite fiction。在这种情况下，有时需要将数据渗透到基础设施的各个角落，这样原先设想中的单个、统一数据套装，现在终于得以真正实现了。

微软公司的 ADO.NET 可以帮助我们在分离的基础设施组件之间建立一个传递数据的结构。它也可以帮助我们解决与版本相关的问题（是否当前版本是最新版本？）以及一致性问题（跟踪读取的用户，更重要的是编写数据，防止不完整或者错误的升级）。当应用访问数据时，Snapshot 机制也会发挥作用，在反对数据以前，它们首先要对数据进行快照：应用完成工作之后，我们会拿原始快照和完成后的主集进行比较。如果二者仍旧是相同的，升级就完成了，如果二者不同，则需要新的快照，并重新启动升级过程。

ADO.NET 使用 XML 在 SOA 各要素之间进行通信，以便传递数据。这意味着只要寄件人和接受者都能意识到 XML，就可以保证所有的数据都精确的得以描述，并且可以在二者明确、清晰理解 XML 的基础上交换数据。二者可以共享 Schema， DTD 或者原语言描述。因为 ADO.NET 是在这种方式下工作的，寄件人和接受者在交换数据时不一定必须实施 ADO.NET。只要双方都能接受 XML 形式的输入，并创建一个 XML 形式的输出，数据交换就可以继续进行。

服务数据目标或者 SDO 为 SOA 抽取数据访问提供了一种方式。当数据库作为数据源和汇时，ADO.NET 运行十分顺利，但是 SDO 不只限于特定的几个类型的数据源。SDO 定义了一个松耦合架构，在该架构中，用户在操作源内容时，不必为数据源维护活动连接。SDO 常用于为服务组件架构定义一个标准机制，一个服务组件集成模型。

他们的方法就是用一系列工作中的数据建立一个相互关联的数据目标图表，并跟踪图表发生的变化，以及图表中所有节点的原始状态。这叫做变化概要。其本质就是要审核所有操作，以及这些操作发生的相应的变化。该图表方法可以使用户将通信限定在工作所需的范围内，并提供了一个用于决策的一致性机制。

同 ADO.NET 一样， SDO 也依赖 XML 描述通信的本质。这个描述数据的图表机制，在整个结构中进行导航，进行价值比较，处理升级，以及数据变化，所有的这一切都是在明确、清晰的 XML 基础之上完成的。因此，任何能够以 XML 形式输入并产生 XML 的服务，流程或者应用都可以使用 SDO。

该方法回避了许多在操作特定编程语言时固有的问题。例如 Java 或 C#，在所有的通信方，它们依靠一个共享的运行时间环境。ADO.NET 和 SDO 的 XML 基础令开发商将注意力集中到数据交换上，帮助他们解决间歇式 SOA 或者连接式 SOA 基础设施中固有的问题。要想确保数据在需要时可以及时的被使用，并要保持数据的现实性和准确性。没有哪种方法是一成不变的。但是它确实提供了一种令广泛分散的系统和组件在 SOA 相互联系的一种方式。

(作者：Ed Tittel 译者：杨君 来源：TechTarget 中国)