事件驱动架构 EDA

事件驱动架构 EDA

EDA 是事件驱动架构，在面向服务架构(SOA)领域，一个比较重要的概念就是事件驱动的体系结构 (EDA)，英文全称为 Event-driven Architecture。

EDA 允许您将创建或遇到事件的过程中的所有这些事件发布到一个中央事件处理主干上，从而使所有感兴趣的相关方可以从此处找到它们。产生事件的过程或服务本身无需考虑这些外部各方，否则，会给该特定过程的执行带来压力，系统之间交织过密，造成维护困难。

对于 IT 部门来说幸运的是，EDA 方法与 SOA 是互补的，随着具有前瞻思想的 IT 部门伴随 Web Services 向前进时，他们开始更多地采用此方法。采用基于 SOA 的方法，开发者通过将那些定义可重复用业务功能的“服务”或软件部件组装起来，就能构建应用程序。

什么是 EDA?

Gartner 在 2003 年引入了一个新术语事件驱动架构（Event Driven Architecture，EDA), 主要用于描述一种基于事件的范例。EDA 是一种用于进行设计和实现应用和系统的方法—在这些应用和系统里,事件所触发的消息可以在独立的、非耦合的组件和服务之间传递，这些模块彼此并不知晓对方。这些应用程序中的 EDA 极大地改进了企业或政府响应不同的、表面上毫无关联事件的能力。通过提供瞬时过滤、聚合和关联事件的能力，EDA 可以快速地检测出事件并判断它的类型，从而帮助组织机构快速、恰当地响应和处理这些事件。通常事件可以采用发布/订阅机制。

- 事件驱动架构（EDA）的最佳方法

EDA 应用

事件驱动架构(EDA)是分布式应用程序的普遍架构形式，非常典型的是:分布式应用程序都被设计成为模块化的、封装的、可共享事件服务的组件。能够通过应用程序、适配器以及无入侵性的代理操作来创建这些服务。由于 EDA 的特点，在金融贸易、能源贸易、电信以及欺诈检测这些行业中，一直都在采用事件驱动架构(EDA)技术。近期在我国政府的电子政务建设中，利用 EDA 分布式处理架构的优势构建共享交换平台，实现跨部门、跨平台、跨应用系统的政务信息资源的共享与交换，并对政府应急系统和跨委办局之间的业务协同办公提供支撑和保障。

- Web 服务通告
- 你需要多少架构？
- 事件流处理和业务灵活性

EDA 与 SOA 的关系

SOA（service-oriented architecture）是面向服务的一种体系架构，1996 年，Garnter 就预见到了服务构架的重要性，并提出了 SOA 概念。有些观点认为 EDA 的出现会逐渐取代 SOA，其实这并不正确，EDA 并不会完全取代 SOA，而会对 SOA 形成补充，有人称之为 "Event driven SOA"。虽然 SOA 通常更适合请求/响应交换环境，但 EDA 引入了一些长时间运行的异步进程功能。而且，EDA 节点可发布事件，且并不依赖于所发布的服务的可用性。它真正地实现了同其他节点的分离。

- SOA+EDA=FUD?
- 作为企业责任的 SOA

- SOA 和 EDA 连接桥梁—规范商业模型
- SOA 和 EDA：用事件跨越解耦合服务边界

事件驱动架构（EDA）的最佳方法

 Sri Nagabhirava 是 nleague 服务公司的创始人和总设计师，专注于面向服务的方向为中心的咨询和解决方案。

问：您能介绍一些事件驱动业务流程最佳做法和错误做法吗？

答：以下是一些实施事件驱动的业务流程最佳做法：

- 综合数据来源
- Alert 定义和通知
- 业务流程的定义
- 多种来源事件关联
- alerts 自动回复

上述前三个综合数据来源，alert 定义/通知和业务流程定义是至关重要的组成部分。在第四清单中，多种来源事件关联使系统关联可能出现在无关的系统级事件表面上，制作一个有关业务级别的活动。在上述清单的第五能力，alerts 自动回复。自动回复，很简单，因为记录错误或复杂援引一个自动的过程中对内部或外部的制度。这种技术，用户不仅要干预真正的特殊条件。还应该使用户定义的各种视觉的隐喻，让事件都最有意义的显示给用户。mash 最多的活动通知与 charts, maps (2D or 3D)使用 Enterprise 2.0 技术

在这种情况下将是非常有效的。代表事件的表格数据掩盖了真正发生的事情，破坏了事件驱动的业务流程的实现。

（作者：Srinath ’Sri’ Nagabhirava 来源：TechTarget 中国）

Web 服务通告

由于 Web 服务没有锁定某个特定的平台或者用户，所以在实现松耦合方面起了很大的作用。原则上来说，该项设计是诸多企业进行设置的最佳选择。下面我们要解决的一个重要问题，这个问题在松耦合架构方面需要一个特定服务：应用通告。

应用通告这个概念在 Java JMS,Microsoft MSMQ 和 TIBCO Rendezvous 这样的企业信息技术领域根深蒂固。迫于在不同的应用之间建立通讯渠道的压力，通信技术成为了 IT 企业领域的前沿科技。但是，在将多种应用混合和对比的过程当中也出现了许多问题，例如建立一个统一的通告机制解决设计中的固有问题。

Web 服务的首要目标就是运用相似的原则桥接应用的差异，结果会把 XML 这个同样通告问题带到某个 Web 服务情况当中。Web 服务中的重量问题不仅是因为在连接不同应用时所产生的不同的设计变量，还与应用和网络绑定的性质有关，这样就会产生如等待时间，网络故障，以及设计无效等问题。

在解决这些问题的过程中，改进过许多 Web 服务标准的 OASIS 公司创建了 Web 服务通告(WSN)。WSN 的目的是为建立基于主题的发布/预定 Web 服务提供一个事件驱动的方式。如果你了解以前提到的企业通信技术，这条术语对你来说也许陌生，它包括以下几个部分：

- 订户：一个将自己注册到特定事件/通告中以便得到更新的 Web 服务
- 发行人：一个将通告信息发送给订户的 Web 服务。
- 事件/通告：令发行人将通告信息传递给订户的真正报告表
- 通告信息：依照事件/通告，将自己发送到订户手中的有效负荷传

WSN 研究小组将原有的部分分解为更加具体的概念，这足以帮助我们理解 WSN 试图解决的 Web 服务模式。WSN 事实上必须遵守三个规范：WS-Base Notification,WS-Brokered Notification 和 WS-Topics，但是在认识到这三个规范的复杂性之后，这些研究小组应该最小限度的考虑到负责 WSN 实施的终端用户。

既然我们已经为大家介绍了 WSN 的核心理念，现在我们再介绍一个 WSN 实例。在其它的专栏中，我们将依据 Apache Software Foundation 生产的开放源软件进行深入的探索。Pubscribe 将为我们提供一个可以和 Apache 生产的 Web 服务栈一起使用的 WSN 实施。，例如主引擎使用的 Axis。Apache 的 WSRF 实施，用于 WS-Coordination 的 Apache Kandula 或者在其它项目/规范中用于 WS-ReliableMessaging 的 Apache Sandesha。

在开始工作之前，我们应该认识到 Apache 的 Pubscribe 是与其 WSRF（Web 服务资源框架）实施紧密地整合在一起的。由于 Web 服务通告流程暗含这代表发布人进行数据检索维护某种状态的意思，但是我们要注意的是这并不表示每一个 Web 服务栈要求 WSRF 使用 WSN.这是 WSN 设计师的独特选择。这种路由可能和其他的 WSN 实施不同。对此你可以阅读先前关于 WSRF 核心概念的专栏。我们的实施包含两种类型呼叫中心使用的 Web 服务。其中一个类型服务是用来传送外来信息的（发布人），其他的类型是用真正的有效负荷（通告信息）来处理请求的（订户）。该有效负荷包含了 XML 语言编码的片段。从业务层面来看，Web 服务发布人可能代表一个机构的总部，订户可以被看做是机构的分部或者是离岸系统，它可以在先到先得的原则之上接收发布人的请求。

我们的应用中包含两种类型的 Web 服务，其中一种用于传送外来请求（发布人）另一种用有效负荷（通告信息）来处理这些请求（订户），该有效负荷中包含 XML 编码的片断，在业务层面，网络服务发布人代表机构的总部，而订户则代表分部或者离岸系统，并按照先到先得的原则来接收外来请求。

在这种状态下，我们要先看一看总部将要发布的 WSDL 合同，这项合同需要经过每个订户的处理，表 1.1 展示了这个合同。

```
<?xml version="1.0"?>

<definitions name="SupportSystemResourceDefinition"
   targetNamespace="http://ws.apache.org/resource/example/filesystem"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.xsd"
         xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
ResourceProperties-1.2-draft-01.wsdl"
          xmlns:wsntw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-
BaseNotification-1.2-draft-01.wsdl">

<!-- WSDL <import>, <types> and <message>
     sections omitted for brevity -->

                              <portType      name="SupportSystemPortType"
wsrp:ResourceProperties="tns:SupportSystemProperties">

    <!-- Resource operations -->
     <operation name="GetResourceProperty">
                                 <input   name="GetResourcePropertyRequest"
message="wsrpw:GetResourcePropertyRequest"/>
                               <output   name="GetResourcePropertyResponse"
message="wsrpw:GetResourcePropertyResponse"/>
                                     <fault    name="ResourceUnknownFault"
message="wsrpw:ResourceUnknownFault"/>
```

```
                        <fault   name="InvalidResourcePropertyQNameFault"
message="wsrpw:InvalidResourcePropertyQNameFault"/>
            </operation>

            <operation name="SetResourceProperties">
                        <input   name="SetResourcePropertiesRequest"
message="wsrpw:SetResourcePropertiesRequest"/>
                        <output   name="SetResourcePropertiesResponse"
message="wsrpw:SetResourcePropertiesResponse"/>
                        <fault    name="ResourceUnknownFault"
message="wsrpw:ResourceUnknownFault"/>
                        <fault   name="InvalidResourcePropertyQNameFault"
message="wsrpw:InvalidResourcePropertyQNameFault"/>
            </operation>

          <!-- Other Resource operations omitted for brevity -->

          <!-- Notification operations -->

           <operation name="Subscribe">
              <input message="wsntw:SubscribeRequest" />
              <output message="wsntw:SubscribeResponse" />
                        <fault    name="ResourceUnknownFault"
message="wsntw:ResourceUnknownFault" />
                        <fault   name="SubscribeCreationFailedFault"
message="wsntw:SubscribeCreationFailedFault" />
                        <fault   name="TopicPathDialectUnknownFault"
message="wsntw:TopicPathDialectUnknownFault" />
```

```
            </operation>

            <operation name="GetCurrentMessage">
               <input message="wsntw:GetCurrentMessageRequest" />
               <output message="wsntw:GetCurrentMessageResponse" />
                                        <fault    name="ResourceUnknownFault"
message="wsntw:ResourceUnknownFault" />
                                      <fault   name="InvalidTopicExpressionFault"
message="wsntw:InvalidTopicExpressionFault" />
                                         <fault    name="TopicNotSupportedFault"
message="wsntw:TopicNotSupportedFault" />
                                      <fault   name="NoCurrentMessageOnTopicFault"
message="wsntw:NoCurrentMessageOnTopicFault" />
            </operation>

          <!-- Other Resource operations omitted for brevity -->

        </portType>

                              <binding     name="SupportSystemSoapHttpBinding"
type="tns:SupportSystemPortType">

                                          <soap:binding     style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

            <!-- Binding for notification operation(s) -->

            <operation name="Subscribe">
```

```
                <soap:operation style="document"/>
                <input>
                   <soap:body use="literal"/>
                </input>
                <output>
                   <soap:body use="literal"/>
                </output>
                <fault name="ResourceUnknownFault">
                   <soap:fault name="ResourceUnknownFault" use="literal"/>
                </fault>
                <fault name="SubscribeCreationFailedFault">
                            <soap:fault  name="SubscribeCreationFailedFault"
use="literal"/>
                </fault>
                <fault name="TopicPathDialectUnknownFault">
                            <soap:fault  name="TopicPathDialectUnknownFault"
use="literal"/>
                </fault>
             </operation>

              <!-- Other bindings for operations omitted from brevity -->

          </binding>

          <!-- Service name declaration -->
          <service name="SupportSystemService">
                                                  <port      name="filesystem"
binding="tns:SupportSystemSoapHttpBinding">
```

```
                                                    <soap:address
location="http://www.hqsupportsystem.com/pubscribe/services/pcsupport"/>
                </port>
            </service>

        </definitions>
```

在表的最顶端，你会看到 Web 服务通告的命名空间。接下来你会看到两个和通告事件相关的操作：预定和获取现有信息，用户 Web 服务呼叫这两种方法以便通告发布人它想要了解事件，并且获得最新信息。在 WSDL 合同的各处你会找到 WSRF 语句以及同 WSDL 合同相关的典型操作和捆绑。，但是我们只能描述和通告中主题相关的片段。

由于文章篇幅的限制，我们就不再展示 WSDL 背后的 Web 服务，发布人以及相应的订户 Web 服务了。但是必须要确保与 Web 服务相对应的生命周期重点关注上述在 WSDL 提到的两个方法，并保证 Pubscribe 提供的 API 建立的这两种方法的逻辑完全建立起来。

现在我们可以对 WSN 做总结了。 现在你应该能够同 Web 服务一样并入到相同的松耦合架构之中了。有了应用通告的支持，业务就需要更长时间的需要基于事件通告的存活过程。

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

你需要多少架构？

我们发现事件驱动架构(EDA)和面向服务架构(SOA)之间的区别后，很快便认识到另一个结构 TLA（三个字母的首字母缩略词）：流程定向架构（POA）。POA 的支持者认为将流程作为架构的基本元素。POA 利用 SOA 技术，但是却将 SOA 技术作用归为只能将 IT 功能展示为服务。

SOA, POA, EDA，更不用说阶梯层架构——你要问的是究竟需要多少这样的架构？信息技术似乎最终会弄懂什么是软件架构，目前，人人都想建立一个新的架构。这种趋势非常危险：IT 内不同的营地排列在一起，这时我们可能面临着将筒仓架构添加到筒仓操作系统和筒仓应用的风险。最终我们需要的还是更多的筒仓。人们通常的反映是认为“我的架构比你的好”，你要知道，作为 ZapThink 的成员我们都会大力宣扬 SOA 技术。但是这不仅仅是“我们 vs.他们”之间的问题。问题的关键在于人们对这些术语的不同定义。只要 IT 世界能消除对这些定义的误解，我们就能避免架构的破碎作用，向着单一、能为众人所接受的结构方法而迈进。

不同的定义

由于人们对架构这个词的理解各不相同，架构可以指一整套完美的实践或者规范??——换种说法，就是你手头正在做的事情。架构的另一个意思就是一个特定的模型或者系统设计——你所创建的事物。把这些定义结合在一起，架构实践（意义 1）包含架构的创建（意义 2）。

第二个我们必须要澄清的问题就是架构和结构观点的区别。一个结构观点实质上是系统架构的一个方面。要想了解结构观点，一个好的方法就是思考盲人摸象这个故事。你可能还记得，四个盲人被要求描述一头大象，每个盲人分别触摸大象的不同部位，摸到大象

躯干的盲人认为大象是一只长的像马的动物，另一个摸到象鼻的盲人觉得它像是一条蛇，摸到象腿的盲人认为它是一棵树，第四个摸到大象身体的盲人觉得大象是一堵墙。

架构就和那头大象一样，大多数人如同那四个盲人，只看到了架构的某一方面。那些关注计算机、网络、应用的人以指导计算机、网络、应用的机构和原则的角度来看待架构。但是，侧重编写软件的人们却以应用和其它软件组件的机构的角度来看待架构。同样，如果你关注业务流程，你可能把架构看做是在整个机构内发生的设计和流程机构。

就像对待那些盲人一样，所有这些观点都有它真实的一面，但是这些观点都不全面。一个拥有远见卓识的人会看到大象的各个部分的功能和外形会像马、树、和蛇，但当他把大象看做一个整体时，大象却和这四个事物毫无相似之处。

这就是人们产生困惑的根源：POA 实际上只是一种观点。有些时候人们把 SOA 看做是单一的一种观点，但是 ZapThink 却认为 SOA 是一种更为宽泛，包含多重理念的事物。如果你仅仅从狭义上将 SOA 定义为一个叫做 POA 的自下而上的方法。但是 ZapThink 的定义更为宽泛，包含了 POA 部分。对于我们来说，SOA 是以流程为中心的，也是自上而下或者自下而上解决架构问题的最好方法。但是，仅靠其中一种方法是远远不够的——你必须同时兼有 SOA 和 POA。

参考架构的作用

幸运的是，致力于帮助大家扫除疑惑的公司并非只有 ZapThink 一家，像 SOA Blueprints、 OASIS SOA 参考模型技术委员会也在潜心研究参考架构和参考标准——尤其是那些设计师开始处理结构问题所使用的其它工具。但是，我们到底是需要一个包含所有观点的参考模型还是需要包含多种观点的多个模型？这始终是个悬而未决的问题。那么你是不是需要一个包含所有观点（或者至少能够协调观点）的拱形的参考模型呢？广义的参考模型可能从本质上来说就是服务定向？

ZapThink 认为广义的服务定向原则会导致一个以服务为导向，流程定位，事件驱动的包含参考模型的诞生。现在问题的关键是为什么要称之为包含参考模型。是把 SOA 限定为

狭义的技术模型还是将 SOA 延伸为更为广义的模型。你称其为 Fred——我们所关心的是这个术语不要阻碍我们在架构和实施方面获得成功。有时人们把大部分精力都浪费在了对这个术语的争论上，而忽略了实践。

还有一个问题，狭义的 SOA, EDA 和 POA 是否应该被区分为不同的模型——这种区分是否能够真正解决问题，还是将原有的问题复杂化了？如果 SOA 应该以事件为驱动，以流程为导向，EDA 和 POA 是否应该是以服务为导向的——现在你应该明白——将三者区分为不同的模型是否真的有用呢？问题的答案是取决于模型所要展示的是什么。如果这些狭义的模型真的在更为广义协调模型的背景下代表了不同的观点，那么这些模型就能充分发挥作用。最后我们要做的是将架构的实践部分分散成几个 TLA 筒仓。

ZapThink 采取的措施

多年来，ZapThink 一直在讨论结构观点，但是结构观点不是由我们最先提出的。实际上，Philippe Kruchten 早在 1995 年的文章"架构 4+1 视图模型 ."就描述过这个观点。ZapThink 借鉴了 Kruchten 原有的架构 4+1 视图模型，并且针对 SOA 把该观点进行了修改。Kruchten 的模型和我们的模型的最大的不同之处就是这个额外的"+1"视图是用例视图，但是在 SOA 中架构并不解决单个的用例视图，而是用于回应模糊的业务需求。因此，用例视图更为重要，就像以服务为定向的设计师必须协调各种观点以保证能够灵活的从架构中受益。

很显然，我们这里谈到的 SOA，指的是广义上涵盖所有个别观点的 SOA。但是我们谈到的 SOA 更多的是指这个概念的实践而非其形成过程。如果你认为 SOA 的形成是一层不变的，你就无法将灵活性注入架构。因此在面临无法预测的业务变化时，SOA 必须提供灵活性。所以，解释 SOA 的最好方法不是试图解释它到底是什么，而是真正的付诸实践。

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

事件流处理和业务灵活性

多年来对业务灵活性的探索使人们广泛采用面向服务架构（SOA）,现今的 IT 集成架构已经有很大的改观。过去的技术管道开始用企业服务总线进行连接，并为网络、通信和协调以及用于支持 SOA 的服务容器管理提供架构。每一个集成软件供应商都在自己的产品中提供某种形式的 ESB，ESB 目前已成为服务定向应用的集成标准。但是，IT 集成组织下一步的发展方向如何呢？

IT 集成组织下一步的发展方向应该是一类名为事件流处理的软件（ESP）。ESP 被软件供应商和分析人士誉为是“下一个大事件”，因为 ESP 能够帮助 SOA 集成组织更加智能化、及时做出响应。ESP 令业务以迥然不同的方式思考其操作和 IT 基础设施，因为 ESP 能够理解业务现在的状态而不是业务过去的状态。

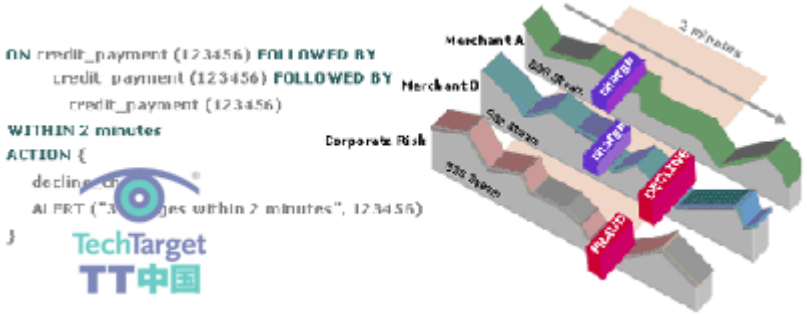
ESB 不是已经发行事件了吗？不是所有的 SOA 基础设施要素都能够发行并运送事件吗？答案是“是的”现今的 ESB 可以处理事件。但是，他们不会规定自己服务发行事件的用途。这也是 ESP 价值的关键所在。ESP 能够使事件驱动的 SOA 破解事件模式（如果是按照先后顺序），弄清事件时间性（4 秒钟内）和空间性（10 英尺内）之间的关系。——并且在实时过程中也能做到上述两点。这就使业务能够持续在实时过程中分析关键绩效指标，并能验证实时风险和机遇，及时采取应对措施。这些功能需要一种新型的数据处理方法——流计算——它可以在事件驱动的 SOA 和实时业务之间传送缺失的链路。

ESP——找出 ESB 事件模式使用的因果关系和时态约束

为了展示 ESB 和 ESP 是如何一同工作的，我们先分析一个特例：信用卡欺诈监测。监测的目的是为了监督系统内部的购买活动并捕获可以分析的授权请求，以便监测欺诈行为。该项操作展示了事件处理范例的三个阶段 1）监测，2）分析，3)行动

首先，我们需要以电子方式对 ESB 上的事件进行存取。以“商人 A 和商人 B”形式出现的源事件流代表通过信息传送给事件引擎的购买活动事件。

其次，我们需要管理这些事件的规则，既然 ESP 引擎可以异步处理事件，事件可以来自任何地方，可以是任何类型，能够以任何顺序被接收。事件处理语言能够将事件特征，事件发生时间，以及经过推断的事件间因果关系作为事件处理语言的基本元素，而不是把结构化数据和 SQL 的关系代数作为其基本元素。EPL 流程对“局部性”提出了质疑—例如，当 A 和 B 为真，那么如果 C 在 N 秒钟之内发生，就需要采取行动。实时检测事件模式可以帮助应用识别业务瞬间发生的机遇和威胁，如同实时购买和抛售股票，将制造车间自动化或者监测信用卡欺诈行为一样。以下是一个基于事件原则的实例。



这个 EPL 编码的开头是一个事件过滤器“开启信用卡支付（用户）”。这条语句指导引擎监测 ESB 上代表信用卡指令的事件。当事件流经 ESB 时，事件模式就会得到满足。下一语句"FOLLOWED-BY"指导 ESP 引擎监测随后针对相同帐户“用户“的信用卡支付事件。如果这三个指令在两分钟内发生，编码就识别了一个潜在的欺诈活动。尽管我们在前面的文章中没有提过，但是包含"空间"逻辑的例子可能标志着超出购买模式范围的任何一个指令，如果购买位置表示其中的一个指令是非法的，那么该例子可能标志着一个随后的指令。这些例子都诠释了 ESP 第一个核心概念：推断因果关系。ESP 从指令的相互关系中可以推断出其中的一个或者多个指令是由欺诈行为所引起的。

"WITHIN"这条语句向我们展示了 ESP 的又一个重要概念：时间。在这个例子中，如果第三个信用支付事件没有在两分钟内监测到第一个指令，该行为就不会被标记为潜在的欺

诈行为，脚本也就结束了。如同在实时灵活的企业当中，单个事件在流计算当中的重要性也会随着业务的重要性的降低而迅速降低。针对事件采取行动的机遇是非常短暂的。除非事件处理架构能够迅速的感知其重要性并做出回应，否则开发这种状态的机遇转瞬即逝，这种状态也会因随后的事件或者其它的因素而发生改变。

最后，我们要看一看 ESP 的第三个重要组成部分：行动。同信用监测应用一样，一旦监测到一种模式，自动系统就会采取事件驱动的行动。在这个例子中，指令要求遭到拒绝，通过传送 ESB 中的一个派生事件，帐户被标记为欺诈管理活动。

结论

企业服务总线一直都是企业 IT 集成基础设施的主干，并且提供了一系列的事件流使实时构想成为现实。流计算，和 ESP 工具使这一切成为可能，并且能够监测时间，原因和 ESB 事件中基于空间的模式。通过将 ESB 原则和 ESB 规范化的集成组织结合在一起，企业会真正变得更加灵活。

(作者：Mark Palmer 译者：杨君 来源：TechTarget 中国)

SOA+EDA=FUD?

现今人人都在谈论面向服务架构(SOA)，有些供应商和分析人士开始提倡另外一种方法事件驱动架构(EDA)。有些人甚至认为 SOA 和 EDA 二者互为竞争，是两个互补的概念，企业必须从两者之间做出选择，这就造成了 IT 架构委员会在决策时产生了许多恐惧、不确定和怀疑的情绪。如果你也本身也有 FUD 这样的情绪，不必过于紧张，ZapThink 认为从实践的角度出发，EDA 并不是一个单独的结构方法，但是 EDA 却是公司妥善实施 SOA 的核心，是一个 SOA 的附属品。

什么是事件驱动架构？

EDA 的支持者认为，异步信息在独立的软件部件之间进行传递时，这些信息并不知道其它信息的存在——换句话说，是去耦、独立的目标。事件资源如同总线一样在一些中间件集成机制中传送信息，中间件将信息传给那些预定过信息的人。事件本身封装了一个活动，即对于一个特定操作的完全描述。

另一方面，SOA 是一种软件在松耦合状态下的结构方法，是位置相对独立的服务。当 SOA 不是网络服务的必需品时，基于网络服务上的 SOA 就成了 ZapThink 关注的方法，同时 ZapThink 认为 SOA 涵盖了 EDA 的核心理念。特别是，ZapThink 认为 SOA 应该是异步的，因为异步对于服务生产商和用户之间的松耦合非常重要，同时异步也是业务流程基本特征的核心。

要想理解为什么 EDA 是 SOA 的附属品,你需要更多了解潜在标准的细节。首先从理解 SOAP 开始。SOAP 支持四个交互操作模式：请求——响应模式（客户到服务器然后返回），通知——响应（服务器到客户然后返回），单方向从客户到服务器或者从服务器到

客户。在这种模式下，无论是在同步或者异步状态下，SOAP 本身就可以支持远程过程调用和文档形式的互操作。作为 EDA 核心的事件通告从开始就成为了 SOAP 的一部分。

SOAP 本身无法提供事件驱动发布/预定环境所有需求，事实上，许多供应商团体和标准机构正在运用许多方法来完成这些需求：一部分人在致力研究 Web Services Notification (WS-Notification), Web Services Base Notification (WS-BaseNotification), Web Services Brokered Notification (WS-BrokeredNotification) 以及 Web Services Topics (WS-Topics)，而另一些人正在研究 Web Services Metadata Exchange (WS-MetadataExchange), 和 Web Services Business Activity Framework (WS-BusinessActivity).很明显我们在把长长的规范项目单添加到单一的现代化的标准当中还有许多工作要做。但是一旦尘埃落定，基于 Web 服务的 SOA 就能提供一个完全而又坚固的事件驱动机制。

更重要的是，ZapThink 正在研究将其应用逻辑和功能粗粒度化。他们看起来“更具商业魅力”而不是“更像 API”。研究的最高水平就是让服务用户无法看到公司如何操作（其流程和服务）。调用这些服务就成了传送正确信息启动流程的问题，这样就会为下面的服务处理生成事件。

所有的 EDS 都是以服务为导向的么？

如果你之前一直在仔细阅读这篇文章，你就会注意到 EDA 和 SOA 之间的微妙差异：SOA 是松耦合的，EDA 是去耦合的。换句话说，除了服务合同的内容之外（例如服务的 WSDL 文件），Web 服务过程和用户不需要了解其它工作程序，但是在 EDA 中，不需要这样的合同。事件生产者和消费者之间的联系就是他们发布和预定的活动。

去耦合 EDA 面临的最大问题是，它们不是为了某些特定的应用与应用之间的通信而量身订做的。这种去耦合是为完全无结构信息所服务的，例如，人类使用的电文信息。当一个应用发布了另一个应用所需要的数据时，在缺少服务合同的情况下，这些数据必须是细

粒度的。换句话说，一个完全去耦合事件可能在警告人们一个流程完毕或者是某种肯定回答，但是如果没有服务合同，很难从如此复杂，结构化的事件中理出预定应用的头绪。

那么这里的底线就是，EDA 可以不以服务为导向吗？从实用的角度考虑，EDA 中的事件驱动交互操作应该是以服务为导向的。两种方法只是在技术上略有差异，这种差异对业务的影响不大。因此，我们现在所讨论的并不是一个叫做 EDA 的独立概念，而是将两个概念融合之后的一个“事件驱动 SOA”理念。

为了理解区分 SOA 和 EDA 的重要性，我们需要看看人们的观点，无论是好是坏，SOA 却被过度炒作，受到了其本身应有的关注，人们很自然会产生疑问，并在寻求其它的方法。好让过于膨胀的 SOA 放放气。ZapThink 正在驾驶这 SOA 这架花车，我们忠实的读者可以相信，我们一定会给这些过度的炒作降降温，，因此，在有关 SOA/EDA 争论的背后也有我们所支持的积极的一面。

一些业内分析人士一直都怀有 FUD 情绪，很自然每个分析者都需要一个自己能够捍卫的独特的市场范畴和争论点，但是已经有太多的分析人士在研究 SOA，人少有人会另辟蹊径。不幸的是，对于那些向分析师寻求指引的 IT 用户来说，分析师这种只关注 SOA 愚蠢的行为毫无益处。

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

作为企业责任的 SOA

在这个重视企业责任和一致性的年代里，我们需要考虑 IT 基础设施以及其它公司资产的效能。对于大多数公司来说，因为它们无法及时适应业务变化，企业架构成为制约业务经济发展最为重要的瓶颈。许多 IT 人士认清了这个事实，并且我们即将不可避免的迈入充斥着这个事实的年代。事实上，一些颇具创新意识的公司正在学习如何利用 SOA 将 IT 技术从目前静止、脆弱的形式下解放出来。而那些创新意识较弱的公司很快发现自己的市场份额输给了那些灵活应用 SOA 理念的竞争者。

其次，那些采用低效架构形式的公司会从股价中发现自己的商业价值受到了影响，甚至可能像真正的企业责任一样，当企业架构的形势发生改变时，成为股东起诉的对象。如果管理层的投资者对其中的误差和误销售做出了解释，那么由于 IT 的低效率而造成的损失远远超出了他们给予解释的范围。因此，拥有高效、灵活的企业架构是一个企业责任，并且那些没有对 IT 架构给予足够关注的人，在短期内是不会发现这些损失的，但是时间长了，这些损失足以导致企业破产。

大多数人都承认，现有企业架构的低效能问题正处于关键性阶段，许多个人依赖那些关注公司效能的人所忽略的问题。事实上，企业架构技术性很强，即使是技术人员都很难理解，更不要说是非专业人士了。到目前为止，那些一直重点关注可疑计算和公司管理不当的公司也反映了公司服务中 IT 的不可见性，并且给予 IT 行业致命一击。如果 IT 行业确实是我们通常认为的由业务驱动的资源，那么，情况无论如何也不会是这样的。现今，那些公司评估者和投资人需要纵观企业的方方面面，尤其是那些效能偏低的领域，这其中也包括 IT 领域。换句话说，功能失调的企业架构最终会降低整个公司的价值，令业务面临风险。

仅仅通过在系统间建立连接和采用新技术，现有的企业架构是无法固着的。架构是一种规范，需要大量的策划，需要分析师制定策略，令企业架构能够实现业务的灵活性和高效性。因此，完成一个脆弱的，不可变更的/或者供应商驱动的架构需要耗费几年的时间，所以现在是时候建立一个能够满足业务需要的架构了。

情况是如何变坏的？

IT 系统，许多人愿意将其看作是一个企业架构，像考古学一样——本质上建立在层层的系统，应用，数据库，连接之上用以解决战略问题的工具。许多公司说自己公司企业架构的长期发展方针。事实上，对于大多数公司来说，多年来战略需要早已超出了战略方针。层层的技术就是最好的说明，一个不可变更、静态、脆弱的架构是很难同业务需求一起改变。这些是常态而不是偶然发生的例外。

为了应对这种困境，企业创建了企业设计师。在机构里，这些企业设计师是单个的或者成组的定位，并且负责为企业架构策略向前发展提供动力。但是这些都是理论，事实上这些企业设计师在公司或者政府内部没有行政权利和财政资金支配权。许多实例表明，企业设计师被派去编写那些无人问津的报告，并且这些报告中包含的方针和指导性意见很容易被人们所忽略。因此如果没有结构管理和持续的公司管理压力重新定向 IT 项目资源，企业架构会一直地保持复杂、静止、和易碎。几年前，这些还只是公司遇到的小麻烦，但是现在却成了制约业务创建股东价值的瓶颈。公司不会轻易转向投入新兴市场，收购公司，它需要很长时间才能调整主要的业务流程。在有些情况下，公司完全无法在策略上进行改变。换句话说，情况变得越来越糟了。

SOA 的作用

面向服务架构不是治疗不良架构的灵丹妙药。但是，对于那些想要令自己现有企业架构更为有效更有价值的公司来说却是一个正确的选择。那些在长期战略结构计划中采用 SOA 和实用结构方法的公司能够顺利的完成结构更新，并处于同行业的领先地位。

SOA 有两大贡献，第一个贡献是能够降低冗余，通过将资源看成是可复合、松耦合、可重用资产，增加资产的名义价值。第二个贡献是能够从数据到功能到流程乃至原则上的大范围内改变 IT 以便在面对个别市场变异时满足不断变化的业务需求。因此，灵活性是 SOA 和企业架构的最终价值主张。不幸的是，这些我们都多次提到过，许多想要采用 SOA 理念的人正在试图使用一个 SOA-in-a-box 类型的解决方案????可能是一个企业服务总线（ESB）一个业务流程引擎，或者是一个治理工具，或者是所有这些。不幸的是，“购买并采用”技术很少能解决问题，并且可能令情况变得更糟。ZapThink 曾多次表示，SOA 是你要做的技术，而不是要买的技术。但是这些在 SOA 策划阶段的购买模式很大程度上是受“炒作驱动”并且由“舆论管理”方案的影响，当人们意识到这项技术并没有炒作中的那么好时，会使许多人对 SOA 产生反感。因此，这里没有什么捷径可以走，必须脚踏实地的工作。

事实上，执行 SOA 是一项非常复杂的任务，你需要许多学习才能通过新兴的方法、技术实现高效性。那些成功制定 SOA 计划和完成 SOA 设计的公司早在开发和实施之前就开始工作了。只有那些理解 SOA 工作重要性的人才能真正走上战略 SOA 和有价值的 SOA 实施之路。另外它们有公司赞助、预算拨款以及恰当的监控和培训。

如同一个新产品或者一项一样，SOA 的发展在公司和机构内部具有重要的战略意义。事实上，考虑到长期的投资回报率，计划周密、充分实施 SOA 相比之下更有价值。从本质上来说。SOA 应该具有纵观全局的能力。

有时，架构似乎应该有一个不良人力资源机构。对于那些分析企业架构在业务上限定的真正成本的人来说，这种益处尤为清楚。但是 IT 对业务限定的消极影响被人们广为接受，并被认为是无法解决的问题。但是，如果公司管理者缺乏动力自己解决结构问题的动力，那些消除多年来公司红利的股东诉讼和错失的战略机遇就会成为他们新的动力。试想你是公司的股东，并且最近蒙受严重的业务损害，损失，债务，以及由 IT 的不变性引起的灵活性问题。（例如 US Airways, Jet Blue, TD Ameritrade, TJX Corporation，这几

家公司等等）我们想在 rants@zapthink.com 上，听一听你的想法。让我们一起开始行动把！

ZapThink 采取的措施

无效率问题现在已经对大多数企业架构的有限业务产生了全局性的影响。由于层层的战略 IT 项目叠加在已有的功能失调的架构之上，情况变得更糟了。因此，无效率问题不再单纯的是解决结构问题。除非你想把业务置于危险而不顾，解决无效率问题是我们必须要做的工作。现在是时侯为解决架构问题进行投资了，但是因为问题复杂，并且意义深远，这需要耗费机构很长时间。为企业架构的再生制定计划和预算，使用好的方法和技术，SOA 无疑是最好的选择。

但是，跳过其它步骤首先采用 SOA 解决结构问题也不值得提倡。首先，企业要认识到这个问题和持续的不变性和变化力有关，并愿意长期部署资源解决问题。其次，机构需要参与其中帮助他们解决问题，例如对 SOA 进行指导，对员工进行培训，并提供咨询服务。第三，业务需要制定一个长期规划，确保考虑到所有相关业务驱动程序和现有问题。这些会包含许多项目，并朝着长期战略目标迈进。最后，要想实现高效执行，必须要确保将 SOA 纳入逻辑结论，并且要尽量避免由于业务需要而重新定向资源。

（作者：David Linthicum 译者：杨君 来源：TechTarget 中国）

SOA 和 EDA 连接桥梁—规范商业模型

虽然 SOA 做出了很多承诺，但是消费者和供应商之间缺乏理解的共同基础将是实现价值和有意义的重用之路的障碍。想像一下，假如没有了保证代表们能够理解的同声翻译，联合国会员大会成什么样。演讲者的观点将无法被目标听众接受。事实上，听众将无法知道是否他们关心演讲的内容。

由于大部分 IT 系统中都存在着异构性，系统之间的通讯就像巴别塔(又名通天塔)一样，无法识别共同的主题。每一个打包的应用、自产的解决方案和软件服务都有自己的一套“语言”，由带有“地方特色”或者上下文相关的词汇和语法所组成。

规范的商业模型

答案就是规范商业模型，被定义为表示内生的商业概念、和个人使用或者软硬件实现无关的信息模型。

这个共同的模型作为通用的翻译，类似于在各种科幻小说中被地球居民和外星人雇佣来理解对方的翻译，便于在各种系统语义之间进行理解。

理解是行动的前提。一旦完成了调查，很多事情就成为可能——1)更好的服务重用，2)多个自动商业流程订购者对已发布消息的重用和 3)跨信息流的事件和消息的关联。

更好的服务重用

在 SOA 实现中服务经常被描述为具有位置独立性和实现封装的特征。奇怪的是，这些定义、输入和输出和原有的底层实现联系都太紧密。这造成的影响是服务并没有它本应该的那样被广泛的重用，由于这些描述可能看起来太“外行”而无法使用。如果服务的接口是用通用商业语言来定义的，那么就会有很多的重用机会。

多个订购者对已发布的事件的重用

商业事件表示了企业内部的重要活动。为了避免点到点事件驱动架构的陷阱，这些事件和伴随的信息必须被所有潜在的订购者都理解。编排的商业流程(Orchestrated business processe)在处理通用对象的时候变得更加灵活和动态。

跨信息流的事件和消息的关联

为了获得全局的视图并且准确的理解实时业务中的状态，跨领域中事件和信息流必须“说同样的语言”才能从复杂的信息的推导出情报。否则，就像是没有首先统一分母的情况下计算分数运算一样。在共同的基础上，表示威胁和机会的时事件和信息的模式就可以被识别出来。

创建商业模型

存在着多种选择。一些行业中有很多被广泛接受并使用的、被良好制定的信息模型。在这种情况下，采用这些模型是最容易的。虽然常常很烦琐，有时还过于复杂，好处是它们可以被立即使用。这些行业标准相关的模型在 B2B(business-to-business)的交互中非常有用。

另一种选择是新建一个企业内部使用的商业模型。这可以增量来完成。企业没有必要实施一个庞大的企业建模项目，它往往无法得到投资，只要在完善的建模技术，比如 UML 领域建模，和扩展技术，比如 XML Schema，被使用。架构师应该使用现有的信息存储和模

型来推导出关键的业务领域对象和属性来快速的创建第一次迭代。在建模过程中，首先识别出在很多业务活动中引用到的核心通用商业对象是很重要的。特别的商业流程信息可以通过结合通用的对象和引入上下文相关的属性来实现。

使用模型

异构性是当今 IT 中的事实。因此，不可能存在一种通用的模型在参与商业流程的所有系统中实现。为了从企业神经系统中的信息和事件流中获得最大的价值，就使用集成技术来让在规范商业模型和终端系统之间语义数据的翻译尽可能接近。这种方法创建了一个全面的信息生态系统，所有的参与者都处于相同的水平上。

概要是没有通用的商业模型，SOA、EDA 和 CEP(复杂事件处理)是不可能实现的。只有在很多环境中被理解和使用的条件下，企业信息流才会传达真正的价值。服务将有更好的前景被重用。所有感兴趣的订购者将消费相同的已发布商业事件，而不再通过扩展端到端的时间驱动集成。使用通用的模型，灵活的编排商业流程可以在跨领域内被创建。通过关联跨领域的事件和信息，威胁和机会都可以被识别出来。

(作者：Maja Tibbling 来源：TechTarget 中国)

SOA 和 EDA：用事件跨越解耦合服务边界

各种机构趋向于频繁的改变他们自身的结构。对面向服务和全球化的日益重视加强了这种趋势。结果，世界上大部分的组织都准备好迎接以独立、自治的服务供应商和服务消费者为特点的、以网络为中心的商务结构。现有的部分商业流程将被外包给外部的合作伙伴，同时组织的各个部分和业务单位被转型为服务供应商。这些服务供应商不再仅仅只关注机构的内部，而且同样会在外部市场中寻求一席之地。

所有的一切都朝着随需应变(on-demand)的商业模式发展，在这种模式下，服务供应商对来自于外部环境的刺激——事件——做出反应。为了在一个充满竞争的市场中胜出，他们需要一种高层次的自治性，包括能够自由的选择合适的支持系统。分离程度的扩大能够产生出对于敏捷性的需求，当组织的构成不断变化时候，在服务之间的松耦合能够支持业务的连续的、不受阻碍的发展。

为了达到“真正的”敏捷性，支撑的应用系统必须做到对于机构的各种改变不可知，这些变化包括变化的责任和角色，外包或者内包，部门的拆分或者加强等等此类的重组。此外，业务流程适应机构变化的敏捷性必须不会受到支持它们的 IT 系统的限制。所有这些要求都可以用松耦合来满足。降低耦合的服务更容易在不干扰现有 IT 支持系统连续性的基础上，改造组织的结构。这就是所谓的机构敏捷性的基础。

应用搭建上的灵活性可以通过在定义良好的功能边界内使用共享的服务来完成。基于标准的技术对于敏捷性做出贡献，主要是能够解决对应用快速交付市场的担心(只要这些标准已经成熟并且采用了合适的工具)。敏捷性紧紧依赖于为了重新设计商务流程而对应用进行重构增加敏捷性的能力。最终的结果是降低的 IT 成本和更快的项目交付周期。

EDA 和 SOA 都承诺能够增加整个机构层面上的敏捷性，但是两者采用不同的方式来做到。

EDA 和 SOA 的关系

虽然 EDA 和 SOA 拥有共同的目标，我们如何知道何时使用事件驱动或面向服务的方法呢?那么让我们仔细想想在什么情况下，这两者种构架方式是最合适的。

和传统的分布式架构不同，EDA 并不提倡同步的、命令-控制的消息交换模式。相反，它采用了基于异步的、发布-订阅模式，在这种方式下，发布者完全不知道订阅者的存在，反之亦然。服务的松耦合就意味他们只共享消息的语义。

假如你试图做到商业流程中各个步骤之间的独立性，那么 EDA 可以提供巨大的好处，尤其在联合的、自治的处理环境中。EDA 被公认合适处理以下的情况：

- 流程链中各个层之间的水平通讯
- 需要跨越公认的功能单位边界的流程(包括内部和外部的边界)
- 需要跨越物理单位界限的流程

然而，当试图利用商务流程中的内聚性的情况下，SOA 则可以提供巨大的好处。它可以适用于以下的一些情况：

- 功能分解之后上下等级之间的垂直交互
- 功能上属于请求-应答类型的流程，比如人机对话的时候，用户等待应答
- 具有交易特点的、具有提交及回退性质的流程

类似于 EDA，SOA 被经常通过消息框架来实现，该消息框架同样是利用异步消息模式来实现的请求-应答通讯。通过将请求的发行者和应答的消费者分离，同样将请求的消费

者和应答的发行者分离，让系统的松耦合。但是，由于存在着一种普遍的误解，认为采用 SOA 就意味着使用 Web Service 所采用的那种(紧耦合的)RPC 类型的架构，因此很多 SOA 实现采用了传统的、根生蒂固的同步交换模式。

为了建立一个 SOA 和 EDA 和谐共处的环境，有必要进行一些前期的分析。一个具有建设意义的准备步骤如下：

1. 为商业需求建立一定粒度级别的功能模型，该粒度级别能够满足预期的自治性
2. 勾画出该应用的地形图以便识别出受到影响的系统
3. 将应用的地形图映射到对应的商业功能模型
4. 鉴别出那些跨越功能边界的应用，它们是潜在的“敏捷性瓶颈”(对于那些需要跨越机构的外部边界的应用请给予较高的优先级)

其中第 4 步是在下面的章节中将会进一步讨论的解耦合服务边界(decoupled service boundaries)的基础。当我们寻找系统中的某些位置，在这些位置上 EDA 的发布-订阅模式可以作为连接这些边界、同时允许各个边界保持自己解耦合的状态的一种方式的时候，这些跨越边界的通讯正是我们感兴趣。

完成这样一个简单的流程，可以为引入一定程度的松耦合奠定基础，同时确定一个好的开始点，将这张技术地形图改进成兼俱 SOA 和 EDA 优点的、现代的、灵活的、可适应的环境。

松耦合和“解耦合点”

松耦合意味着独立。简单的说，松耦合的服务对彼此的依赖要比紧耦合的服务小。这个规则同样适合于功能和数据。当某个服务相关的数据被仅仅局限在该服务的功能边界内的时候，服务之间的耦合度就会增加(拉紧)。这种设计方法会增加服务之间为了访问被孤立的数据而形成对彼此依赖的概率。当以数据复制、避免保证数据传输和语义一致性的共

享层之外的其他层共享等机制实现的数据冗余被允许的情况下，服务之间的耦合程度会降低(松开)。

保证跨越解耦合边界的数据冗余让松耦合更加强健。EDA 适合这种情况，由于它支持在冗余环境中的数据的自动一致性。

当使用 SOA 和 EDA 的时候，找到“解耦合点”通常被证明很多帮助。这是通过寻找某些经常同时出现的商业流程部分，这些部分你可以确认它们会一起出现在任何一个组织单位中(具有强烈的内聚性和原子类型的业务功能)。它们就是“解耦合点”。这样，商业流程的功能构成会变得清楚。在各个业务功能之间的边界就是解耦合点。如果原子交易跨越了解耦合边界，那么回滚交易就需要在这些解耦合点实现。

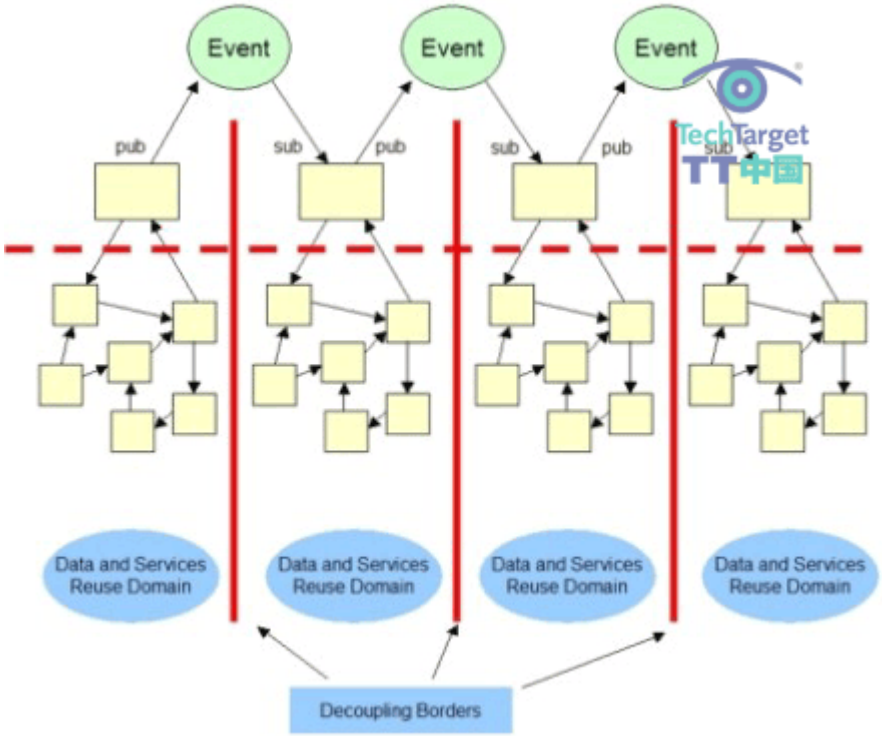


图 1：如何安排事件来跨越解耦合服务的边界

图 1 阐明了 SOA 和 EDA 之间一种可能的关系。在顶上的圆圈表示连接单个松耦合系统的解耦合点(事件)。无论在这些解耦合点上的服务是连接还是断开，都不会影响被连接的系统。不同域之间的所有的数据交换都发生在这些点，而不是在底下的层次。

在一个可重用的域(在图 1 位于底层)中，EDA 的粒度可以进一步精细。EDA 的粒度越精细，系统的灵活性也就好，尽管可重用的域也就越小。

假如在解耦合点使用基于 SOAP 的 Web Service 的技术，同时结合通用的基础架构(例如企业服务总线，enterprise service bus)，那么和其他异构系统系统的连接将会变得很容易，包括 SOAP 包装的遗留系统，商业现货供应软件(COTS)，ERP 系统和外部系统的网关等(如图 2 所示)。

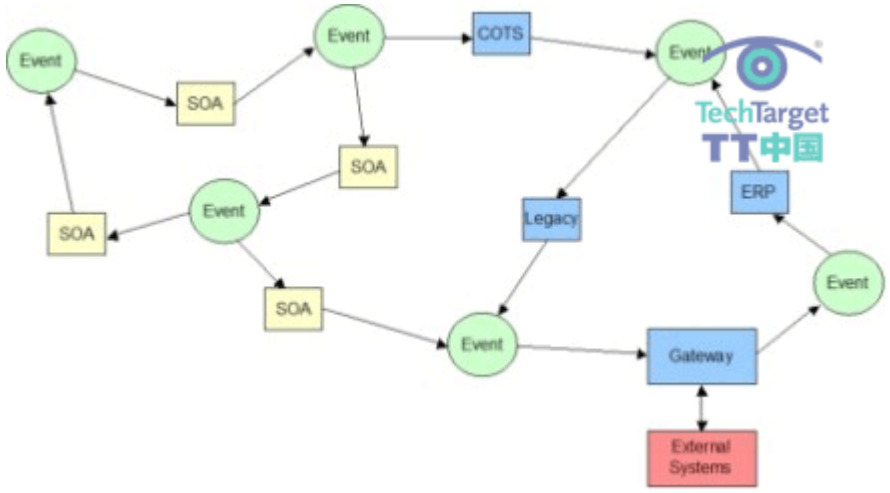


图 2：服务不再被直接耦合，而是通过由事件表示的解耦合点来连接

当谈到 EDA 和 SOA，为了松耦合而奋斗——也可以说是为了灵活性和敏捷性——从所有的粒度级别来讲，都不失为一个好主意。当然，松耦合必须在现实的条件下慢慢实现，尤其是考虑到运行效率的时候。

ESB 的注意事项和全局数据空间

如今使用 Web Service 技术实现 EDA 的时候，需要一个额外的、能够处理 SOAP 的消息队列基础框架。之所以需要这个消息队列并不是由于 SOA 实现是通过 Web Service 的原因，考虑到最本质的 SOA 只需要在已有的网络基础(比如 HTTP 层)上通过 Web Service 就可以实现。

当前的企业服务总线(ESB)构架提供了一种可以将消息队列和 Web Service 技术结合起来的途径。这就是为什么 ESB 非常合适于各种基于 EDA 和 SOA 的解决方案的原因。

另外还需要关注目前出现的几个和 EDA 和 SOA 都相关的关键的 Web Service 标准。其中最有希望被通过的标准包括 WS-Eventing, WS-Notification, WS-MetadataExchange, WS-ReliableMessaging, WS-Security 和 WS-CDL。一旦它们得到广泛的支持，并且和其他刚刚出现的能够处理 SOAP 的基础组件相结合，那么自然，它们会提供很多 ESB 的功能，目前来说这些功能只能由 ESB 的供应商的产品提供。

从 EDA 的角度来说，ESB 非常适合用作容纳已发布的商业事件的容器，因为它允许让这些事件被广泛的用于订阅。结果，ESB 可以被放入到整个企业的全局数据空间中，让所有的应用都能够一致的访问到事件信息，无论它们的位置，时间和后端使用技术(图 3 演示了这一情况)。

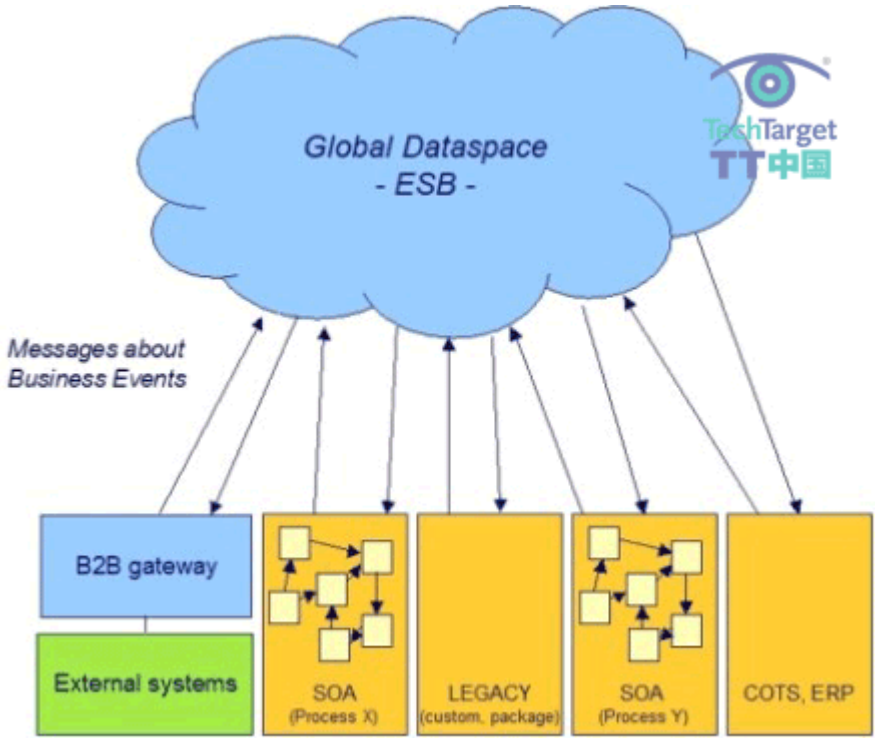


图 3：利用 ESB 实现的全局数据空间

ESB 不必依赖于某家公司的产品。在一个联合公司当中，多条服务总线产品可以结合起来提供 ESB 的服务，这就允许区域的自治。图 4 就演示了一个企业范围内的服务总线是如何通过运行多个域而成为整个组织的全局数据空间。使用 Web Service 的技术能够让跨越多个供应商产品的消息传播变得容易实现。

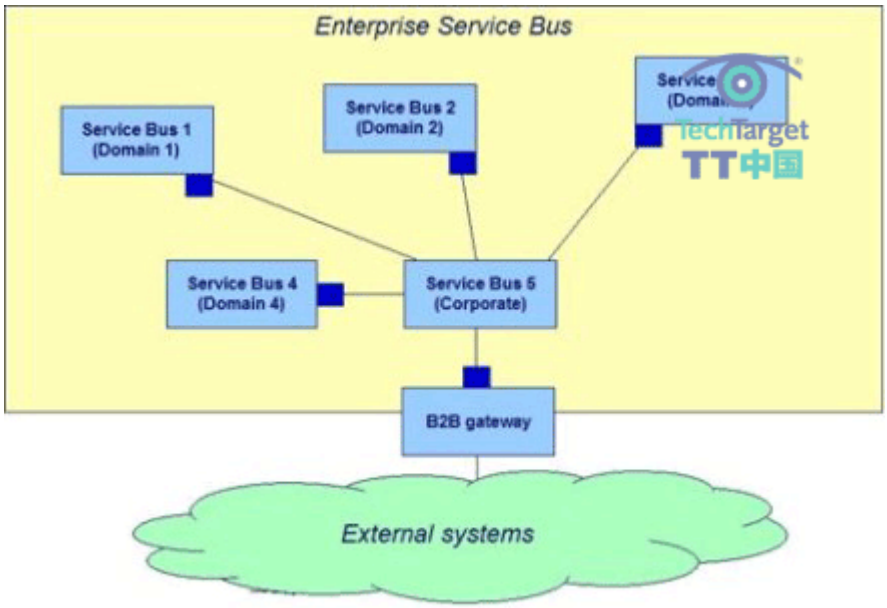


图 4：通过组合 ESB 形成的联合 ESB 结构

BPEL 的注意事项

同样，研究如何在商务流程管理(BPM)和商务流程处理语言(BPEL)的环境中实现 SOA 和 EDA 也是非常重要的。目前阶段下，BPEL 的实现都是集中在命令-控制模型上——基本上服务的控制和组合都是通过类似面向过程的语法来定义的，这需要一个 BPEL 的执行引擎。

BPEL 的使用不会对 SOA 造成任何的问题，因为它可以构建出一个服务组合的父层次，这个父层次完全抽象了商务流程逻辑。当然，任何一个商务流程并不限于这个层，因为 BPEL 允许一个流程可以使用服务来进一步封装，或者由其他的服务流程组合而成。

但是，BPEL 并不是为 EDA 量身而造的。从本质来说，更加适合 EDA 的模型应该是声明性的，而非过程性的。这个模型可以让设计者通过点击的机制就可以将事件和发布者和订

阅者联系起来。执行环境的实现应该独立于控制引擎，同时基于上面谈到的 Web Service 标准。即使一些平台正在朝着这个目标发展，但是目前我们还需要依赖于 JMS 的 SOAP 或者其他由 ESB 产品所提供的基于 SOAP 替代方案来实现。

结合 EDA 和 SOA

将 SOA 和 EDA 结合过程中最大的挑战在于改变方案设计者们的思想。一些和 EDA 相关的新的理念，比如有目的的数据冗余和显式的使用事件，只有在被充分的理解之后，才可能被恰当的使用。

比如，发布-订阅模式背后的动态性以及它与请求-应答模式之间的对立性，都需要被服务设计者和软件构架师理解清楚。只有这样，他们才能够在整体架构中寻找到机会，将 EDA 和 SOA 结合到复杂的消息交换模式中。这可能没有说起来那么容易，因为对于决大部分项目组来说，异步消息并不是一个被广泛接受的设计方式。

虽然发布-订阅模式并不是新创造的，但是对于那些只通过调用堆栈的方式思考问题的人来说，它还是相当陌生的。让我们据个例子，将记录写到批处理文件或数据库中，类似于发布消息。从文件中读取记录则类似于消费了订阅的主题。这些都是异步设计的实例。

结论

商业的事件机制提供了一种强大的方式连接起过去彼此孤立的环境，并且驱使服务的交互更加互动。如果 EDA 所代表的消息模式能够得到被清楚的认识的话，EDA 的所提供的功能可以被更加有效的利用。如果事件驱动的设计方法被得到合理的使用，它将使得服务之间的通讯更加畅通，并且加强服务的质量。

(作者：Jack van Hoof 来源：TechTarget 中国)