ESB是企业服务总线的缩写，而SOA的意思是面向服务架构。

## 真相

假设一下你通过银行前端的应用登入银行，会发生什么呢？

1. 会显示你的名字
2. 会显示你的账号余额
3. 会展示你的信用卡和借记卡
4. 会列出你的共同基金
5. 会列出一些你可能感兴趣的，预先被计算好的，有吸引力收益的借贷产品

现在，很可能所有这些模块都来自于不同的系统和应用，通过各种接口把数据展示出来（是HTTP?JSON? AMQP?XML? SOAP? FTP?CSV?都不重要）

1. 是来自在linux和Oracle的CRM系统
2. 是来自z/OS大型机的COBOL系统
3. 据说是来自大型机，但是他们的嘴巴很紧，不肯告诉你任何事情，只提供CSV文件。
4. 来自跑在Windows的混合着PHP和Ruby
5. 来自Postgresql，Python和Java，跑在Linux和Solaris的

现在的问题是，你怎么让前端的应用跟1到5交互呢？ 直接通信吗？ 不，你不会这样做的，你不会让他们直接对话的。

为了让在这样的复杂的环境中还能保证通过增加一些系统进行扩展，不能直接通信。这是基本原则。

在下图，每条不同宽度或者样式的线表示了app之间的调用

大家要注意到，我们还没展示高优先级的进程呢。（比如应用1调用应用2 ，然后根据应用6的返回成功与否来决定调用应用3或者应用5；如果应用1允许的话，应用4晚点还可以抓取从应用2生产的数据。 -\_-||| ）。大家还需要注意到，我们说的是服务，不是服务器。每个系统都可能跑在10台的物理机器上，那么至少有60个物理部件需要相互通信。

还有一些显而易见的问题：

怎么分离接口？你的计划怎么展示？如果每个应用由不同的组去管理，怎么去协调更新或者计划中的停机时间？如果生产商或者部门一半的开发者都已经离开不在现场了咋办？

如果你觉得，你解决6个应用没有问题，那么30个呢？

你可以处理400个吗？2000个怎么样？每个应用都有自己独特的生态系统，都需要用10个物理服务器或者设备跑在上面。所以，就好比有2万个移动的群体散落在大陆上，并且有着各自的技术的或者文化的边界。 所有这些群体彼此之间需要不断的，持续的交换信息，聊天，一刻也停不下来。

对这种情况，有个很好的名字，叫一团糟。

## 怎么去清理这一团糟的东西？

首先，你得诚实的承认，事情已经发展到不可收拾的地步了。这样子在找补救措施的时候，你会感觉到没有那么内疚。好吧，事情已经发生了，你不知道怎么办。但是，有个机会，这团糟东西可以被清理干净。

这意味着对IT系统要做一些结构上的变更。另外，我们还要回忆一下以前的经验，系统和应用很少用来被创建成到处推送数据，他们是为了支持业务，无论你的业务是什么东西：它可以是银行业务，声音记录，无线电设备或者其他任何东西。

一旦你有了这两个清晰的理解， 你可以开始考虑建设或者重新设计你的系统服务。

服务是有趣的，能重用的，并且原子性的，它能提供为其他有需要的应用提供用途，但是它不会用点对点的方式直接暴露出来。我们尽可能给它最短的有意义的定义。

如果一个给定的功能系统要做成服务，需要满足以下这三个条件：

* 有趣 I nteresting
* 可重用 R eusable
* 原子性 A tomic

因此，如果满足这三个条件，他可以也应该以服务展示给其他系统，而不是直接相连。

我们通过一些例子来讨论IRA方法

| **Variable** | **Notes** |
| --- | --- |
| 变量 | 备注 |
| 环境 | 电力公司CRM系统 |
| 功能 | 在自己的门户中返回在2012年Q3所有客户信息 |
| 有趣吗？ | 是的，能够生成各种有用的报表和统计数据 |
| 能重用吗？ | 不，虽然它可以统计整年的数据，但是很显然，2018年是用不上这玩意的 |
| 原子吗？ | 非常像，如果每个季度都有类似的模块，那么就可以有一整年的视图。 |
| 怎么让它IRA？ | * 让它接收任意的起始和截止时间，而不是限定于某个季度 * 让它接收任意应用的访问，而不仅仅是门户，让具体的某个应用作为参数输入，系统不能只为了门户硬编码 |
| **Variable** | **Notes** |
| 变量 | 备注 |
| 环境 | 电子商务网站 |
| 功能 | 返回某个给定客户收集过的每个商品信息 |
| 有趣吗？ | 是的。当你进入网站，你总是能够找到你想要的东西 |
| 能重用吗？ | 足够有趣，但不是精确的能重用。因为基本上没有其他应用关注里面的数据 |
| 原子吗？ | 一点也不，这个庞然大物的功能在逻辑上是有几十个小部分组成的 |
| 怎么让它IRA？ | * 拆分成细小的模块，考虑怎么去描述客户（他们有地址，电话，喜欢的东西，他们喜欢的联系方式等等），任何这些东西应该做出一个独立的服务 * 使用ESB来组合这些原子的服务 |

| **Variable** | **Notes** |
| --- | --- |
| 变量 | 备注 |
| 环境 | 任何地方的任意CRM |
| 功能 | 在创建账户以后，更新表C\_NAZ\_AJ 的CUST\_AR\_ZN 字段 |
| 有趣吗？ | 当然不是，这是CRM的内部函数，没有人想要用这么低层次的功能。 |
| 能重用吗？ | 可能。通过多渠道创建账户的时候，有些东西是可以重用的 |
| 原子吗？ | 看起来是，只是一个简单的更新表的行为 |
| 怎么让它IRA？ | 不要尝试去把他变成一个服务，他并不有趣。没人愿意去考虑特定的列和表，这是CRM错综复杂的细节。所以即使它是一个可重用的，原子性的，你也不应该用它来做服务。这是你（CRM）的责任去考虑，不要让别人去背这个责任。 |
| **Variable** | **Notes** |
| 变量 | 备注 |
| 环境 | 移动电话运营商 |
| 功能 | 给预付费卡充值 |
| 有趣吗？ | 当然，每个系统都想用它。通过文本信息，语音信息，即时信息，门户，礼物卡等 |
| 能重用吗？ | 是的，他可以提供给任何其他高级的服务。 |
| 原子吗？ | 是的，从调用方看，他提供充值成功或者充值失败的结果。账务系统会把调用结果作为一些步骤其中的一步。从商业透视的角度看， 他是不可见的服务，提供接口给账务系统调用。 |
| 怎么让它IRA？ | 已经IRA了 |

如果你在过去的50年或更长的时间里面，做过程序开发，你会很明显的理解，提供了一个服务，就类似于在代码中提供一个API接口。

不同的是，你不是在处理单系统的子模块，而是在处理整个分布式系统的某个层级。

## 使用SOA架构，用ESB提供服务

当你理解系统并不直接交换信息，理解什么是服务，那么现在你可以开始使用ESB了。

ESB的工作就是提供和调用集成系统的服务。使用了ESB，在大多情况下，每个系统和ESB之间，只需要定义一个访问方法，一个接口。如果这样，像上面的表一样，你有8个系统，就会有16个接口（1个方向1个）需要被创建、维护、管理和关注。

如果没有ESB，你就需要56个接口需要去思考和处理。（假设每个系统都需要跟其他系统对话）少了40个接口意味着少花时间和金钱。这就是你周末不用那么神经紧张的原因之一。

基于这个事实，你应该迫切地考虑需要引进ESB。

如果一个系统需要重写，改变所有者，生产商或者部门分拆，这个是ESB的工作去处理这个变更。其他的系统都甚至不需要周知，因为他们与ESB的接口不变。

当你开始每天使用IRA服务，你可以考虑组合他们。

还记得上面的“把客户所有信息给我”的服务吗？你最好不要去创建这样的服务。虽然有时候你不得不处理客户端应用需要集成和汇总这样的信息。

这将是ESB的责任，去选择原子的服务组合成混合服务，提供给某些客户端系统需要的特定的数据组合。

随着时间的推移，大家开始明白，不再有数据库表、文件、批处理、功能、程序或记录。系统是架设在有趣的、可重用的、原子性的，由ESB提供的服务应用的架构之上。人们不会再考虑应用和系统之间直接传递东西。他们只会考虑ESB作为统一的接入网关，提供可以使用的有趣服务。他们甚至不在了理会谁能够提供什么，他们的系统只会关注ESB，处理与ESB相关的事情。

这需要时间，耐心和合作，但是这个是可行的。

## 但是要注意

毁掉SOA概念的最好的方法就是，推出ESB，就期待所有的事情自己顺顺利利。虽然ESB是一个了不起的想法，但很不幸，只是简单的安装ESB不会解决你太多问题。最好的方法，你还是要打扫一下你的毛毯（整理一下你的架构）。

像下图一样，如果不打扫，即使用了ESB也得不到任何好处。

（如果如上图构建你的系统），那么你的it维护人员就会厌恶这个系统（ESB），管理层虽然一开始会容忍ESB，认为他作为一个新系统，总会出点小问题，但是后来它就会成为一个笑柄（系统难以维护）。“什么？这就是你的新杀手锏？哈哈！”。如果ESB不是作为你IT大计划中的重要组成部分来推动事情发展的，那么这种结果不可避免。

## ESB是只为银行或者类似的应用服务的吗？那么？

完全不是。只要是需要多数据源和多访问方法合作去获得一个有趣结果的情况，都是一个好的选择。

例如，从热传感器读取信息，并且发布到几个通道去，比如email警告、iPhone应用。 这听起来就是一个很好的集成应用平台场景。周期性的查询和监控是否某个应用的所有实例都起来了没有，跑一个预配置的脚本去发送文字信息给管理员的应用场景听起来也不错。所有需要集成在一个干净的，定义好的环境，可能都是一个ESB服务的应用场景。但通常，判断某个东西是否真的匹配成一个服务则需要相关人员的经验。当然了， Zato support 后面的团队可以帮助到你。

## 但是我听说SOA全部都是关于XML，SOAP和Web服务的

是的，这是某些人希望你这样相信的。

如果你与某些使用BASE64编码的CSV文件，并且用SAML2保护的SOAP信息来发送的人或者生产商一起工作，其实挺能理解为什么你会有这样的想法。

XML，SOAP和web服务有他们各自的用途，但是就像其他东西一样，他们也可能被滥用。

SOA是一个关于干净的，可管理的架构。具体的某个服务用或者不用SOAP，其实是与SOA无关的。作为一个架构方法，即使你完全没有SOAP服务在上面，SOA依然是有效的。比如一个建筑师在设计一栋漂亮的建筑，他们真的跟油漆工人采用什么颜色的油漆来处理内饰没太大关系。

所以不是这样的。 SOA不是关于XML，SOAP和web服务的，这些都能在SOA里面使用，但不是SOA的主体。

我们鼓励你友好地去指引你被误导了的同事去阅读本文，使他们理解SOA的真正含义

## 更多的内容

这篇文章主要覆盖非常基础的知识，但是能够让你明白ESB和SOA应该是怎么样的，以及如果要获得成功需要做什么。

这篇文章还不包含（不限于）以下的内容：

* 怎么为ESB获得管理层的支持
* 怎么组织SOA架构师和分析团队
* 在组织中引入规范数据模型
* 关键业绩指标-你现在系统中有统一和通用的方法来提供服务，你应该监控和评估你的系统你收到了什么
* 业务流程管理–什么时候以及怎么样引入业务流程管理平台去协调各个服务（别太快，先熟悉怎么搭建优秀的，优雅的服务）
* 如果没有API的系统，应该怎么做？比如ESB应该直接访问他们的数据库吗？（看情况，没有黄金法则）

## 所以，Zato确切来说是什么东西？

Zato 是ESB，并且服务都是用Python写的。Zato它可以用来构建中间件和后端系统。它是开源软件，有商业公司和社区支持。并且 Zato使用了Python，这个因为他的易用性和生产效率高而闻名的编程语言。使用Python和Zato意味着你有更高的生产效率，少花一些时间在麻烦事上。

Zato 是实用主义者写给实用主义者的软件。 他并不是那些在炒作ESB/SOA的生产商快速胡拼乱凑出来的系统。事实上，Zato来自于对上述一团糟的系统的救火经验。Zato的作者花了太多的时间去处理这些灾难性的的环境，才使他们不至于陷入困境。

因此，Zato就是这样被设计出来的，它是应时而生的解决方案。这也是为什么跟Zato类似的解决方案能提供高效的生产力和无与伦比的易用性的原因。