基于消息的SOA企业解决方案

第一章 SOA简介

本章内容

1. SOA的基本概念
2. SOA参考模型
3. SOA生命周期
4. SOA的实现技术

本章内容

1. SOA的基本概念
2. SOA参考模型
3. SOA生命周期
4. SOA的实现技术

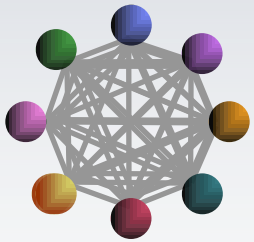
SOA产生的背景

- 企业对于IT系统的需求
  - 传统企业构建IT系统时聚焦于应用程序本身，即满足一个特定业务单元或部门的需求，产生了许多“一次性”的应用程序。
  - 经济全球化、一体化的发展以及IT技术的发展，企业对对IT系统提出新的需求：
    - 企业内部要求IT系统自动化
    - 企业要求与外部系统能够更加灵活地通信
    - 企业希望IT系统能够帮助提高业务系统的执行效率
  - 传统解决方案——点对点集成：
    - 加重复杂性和IT运营成本
    - 不具备通用性
- SOA成为解决企业对IT系统新需求的主流解决方案

IT构架的演变

传统架构

基于消息传递的模式



- 应用之间点对点的连接
- 实现简单、基本的信息交互和数据传递

过渡架构

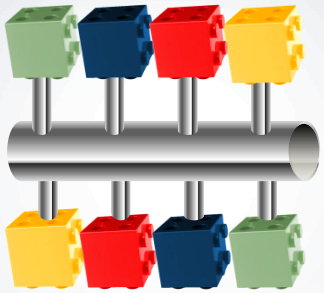
企业应用整合

- 通过HUB模式实现应用之间的整合
- 很容易管理大量的连接和系统

先进构架

面向服务体系架构

SOA



- 通过企业服务总线实现服务的整合集中和流程实现
- 借助标准的接口灵活地连接，实现真正的随需应变

Flexibility

SOA产生的背景

- IT的发展趋势
  - 电子制造业的摩尔定律同样适用于软件行业
  - 架构：C/S → B/S → 基于服务的架构
  - 语言：机器语言→ 汇编→ 面向过程(C…)→面向对象(C++/Java…)→ 面向组件(J2EE)→ 标准化的Web服务
  - IT编程语言的发展过程是逐步降低耦合性，即接口与接口实现之间逐渐分离。
  - 编程粒度逐渐变大，低层架构和技术能够处理的功能更加广泛、层次性的封装愈发普遍，程序员可以更加贴近业务逻辑进行开发，节约程序员的时间。

SOA产生的背景

- SOA的产生和普及
  - SOA的概念是1996年由Gartner公司提出来的。
  - SOA的初衷是：
    - 为企业提供一种构建IT系统的标准和方法
    - 通过建立可组合可重用的服务体系减少IT业务冗余，加快项目开发进程
    - 提高IT部门的效率
    - 实现IT技术和业务的整合
  - 由于当时的技术和市场环境方面的限制，SOA架构没有得到广泛关注
  - 近年来，由于SOAP、WSDL、UDDI等Web Service相关标准的出台，SOA真正进入了实施阶段

SOA的定义

- Service Oriented Architecture面向服务的架构
- SOA的定义分成两类
  - 架构风格：服务是最核心的抽象手段，业务被划分为业务服务和业务流程
  - 分布式软件系统构造方法和环境：涵盖服务的整个生命周期，包含运行环境、编程模型、架构风格及相关方法论等。

SOA架构风格

- 服务
  - 定义了一个与业务功能活业务数据相关的接口， 以及约束这个接口的契约。
- 业务
  - 业务服务：相对独立、自包含、可重用，由一个或多个分布的系统实现
  - 业务流程：企业中一系列创造价值的活动的组合，由服务组装而来

SOA的基本要素

- 松耦合
  - 服务之间的松耦合
  - 接口与实现之间的松耦合
  - 业务组件和传输协议之间的松耦合
- 粗粒度
  - 服务的接口应该比面向对象编程的API大一些
- 位置和传输协议透明
  - 目前的组件调用方式中，位置和传输协议都不透明
  - SOA的服务调用方式中，通过服务总线对组件的接口进行封装，保证服务位置和传输协议的透明。

容易混淆的基本问题

- SOA是架构风格，是方法，不是具体架构具体实现技术（如Web Service）、具体架构元素（如企业服务总线，Enterprise Service Bus， ESB）。
- SOA的首要目标是IT与业务对齐，支持业务的快速变化；其次是IT架构的灵活性和IT资产的重用
- 在工程上，SOA的重点是服务建模和基于SOA的设计原则进行架构决策和设计。

本章内容

1. SOA的基本概念
2. SOA参考模型
3. SOA生命周期
4. SOA的实现技术

SOA参考模型

- SOA参考模型是一个用以指导建立具体SOA的抽象框架，描述了SOA环境中各实体及实体间的关系。现有SOA参考模型可分成三类：
  - 抽象模型，如：OASIS的ROA-RM1.0、W3C的Web服务架构中也给出了面向服务体系结构模型
  - 层次模型，如：IBM的SOA参考模型
  - 基于具体应用平台的模型，如：IBM的SOA Foundation、普元软件EOS工作流、Oracle的融合体系结构、SAP的企业服务架构ESA

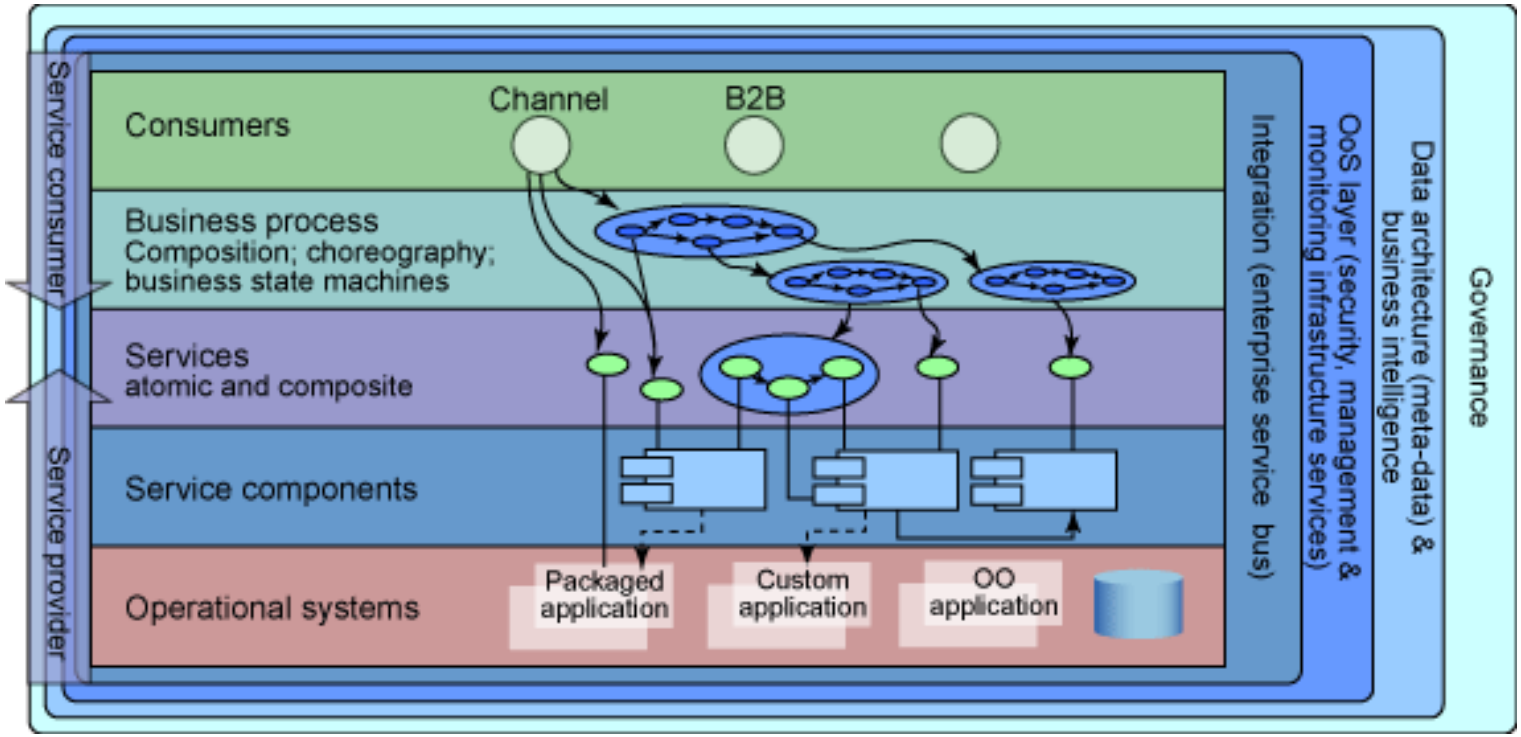
抽象模型

- SOA抽象模型是独立于具体技术、标准、执行规范和实现平台的模型，对SOA提供整体框架，提炼SOA的核心内涵。
- 模型过于抽象，它能够在企业的SOA规划中提供一个具有全局性的整体框架，但在实际执行时却不能提供太多具体可行的意见。
- 举例：
  - SOA-RM1.0定义了7个核心元素，即：服务、服务描述、服务提供者和服务消费者间的可见性、服务执行上下文、现实世界作用、契约和政策、服务交互，提出了通用于各种执行的公共语义。从这7个抽象实体出发，分别研究它们之间的关系。

层次模型

- 层次模型指在传统的3层体系结构基础上，依据流程关系建立的分层模型。
- 层次模型不依赖于实现所使用的技术，但已接近了具体实现的架构，具有可操作性，较抽象模型易于理解。
- 举例：
  - IBM的SOA参考模型描述了SOA解决方案的运行时概念视图，介绍了各个层次和概念（如业务流程、服务或服务组件）及其相互间的关系。

IBM的SOA参考模型



IBM的SOA参考模型

- 操作系统层：表示现有IT资产，应在SOA加以利用
- 服务组件层：实现服务，可能通过使用操作系统层中的一个或多个应用程序来进行。使用者和业务流程并不能直接访问组件，而仅能访问服务。
- 服务层：表示已部署到环境中的服务，由可发现实体进行治理。
- 业务流程层：表示将业务流程作为服务编排实现的操作构件。
- 使用者层：表示用于访问业务流程、服务和应用程序的通道。

IBM的SOA参考模型

- 集成层：集成层是一个横切关注点，支持和提供调节能力，包括变换、路由和协议转换，从服务发起者向正确服务提供者传输服务请求。这里出现的集成主要是服务组件、服务和流程层的集成
- 服务质量层：服务质量层也是一个横切关注点，支持 SOA 相关关注点的非功能性需求。

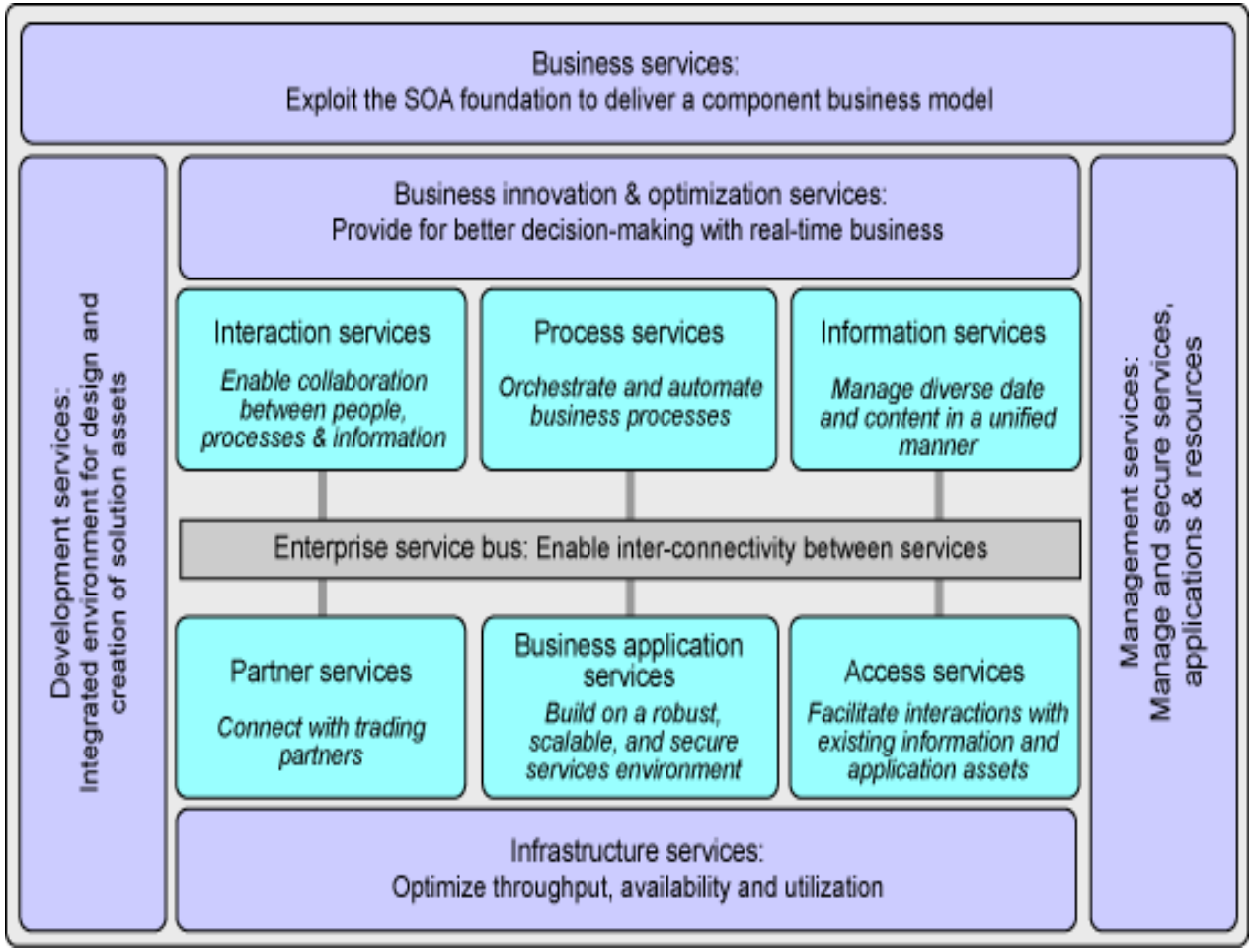
IBM的SOA参考模型

- 数据架构及商业智能层：该层也是一个横切关注点，负责以统一的表示形式呈现一个组织其各方面信息，保证业务需求和流程与业务词汇（词汇表和术语）保持一致。
- 治理层：该层也是一个横切关注点，确保一个组织中的服务和SOA解决方案遵守策略、指导方针和标准，这些均定义为一个应用于组织中的目标、策略和规章的功能。

基于具体应用平台的模型

- 这类参考模型主要由软件组织和企业推出，用自己开发的应用平台和解决方案为其提供支持。
- 由于各自产品的差异性，它们的模型依赖于特定的技术平台，因此不是理想的SOA通用模型。
- 举例：
  - IBM的SOA Foundation参考模型是以企业服务总线ESB为核心的全面企业解决方案，包括建模和组装、部署和服务管理。参考架构呈扁平结构，其中的服务经过模块化集成，通过ESB完成交互。
  - 普元软件EOS工作流对WfMC参考模型进行扩展，提供了灵活的工作流路由模型、激活策略以及版本控制策略。

IBM SOA Foundation



- IBM SOA Foundation从服务角度描述了应用设计中的功能支撑组件，是一个集成的基于开放标准的软件、最佳实践和面向服务架构模式的套件。

IBM SOA Foundation

- 交互服务：提供业务设计的表示逻辑，并支持应用程序和终端用户之间的交互
- 流程服务：把多个服务聚合成为一个与业务过程对应的服务流程
- 信息服务：提供业务设计的数据逻辑
- 业务伙伴服务：来自业务伙伴的服务
- 业务应用服务：使用新平台实现的核心业务逻辑
- 接入服务：将遗留应用和功能整合到SOA架构中

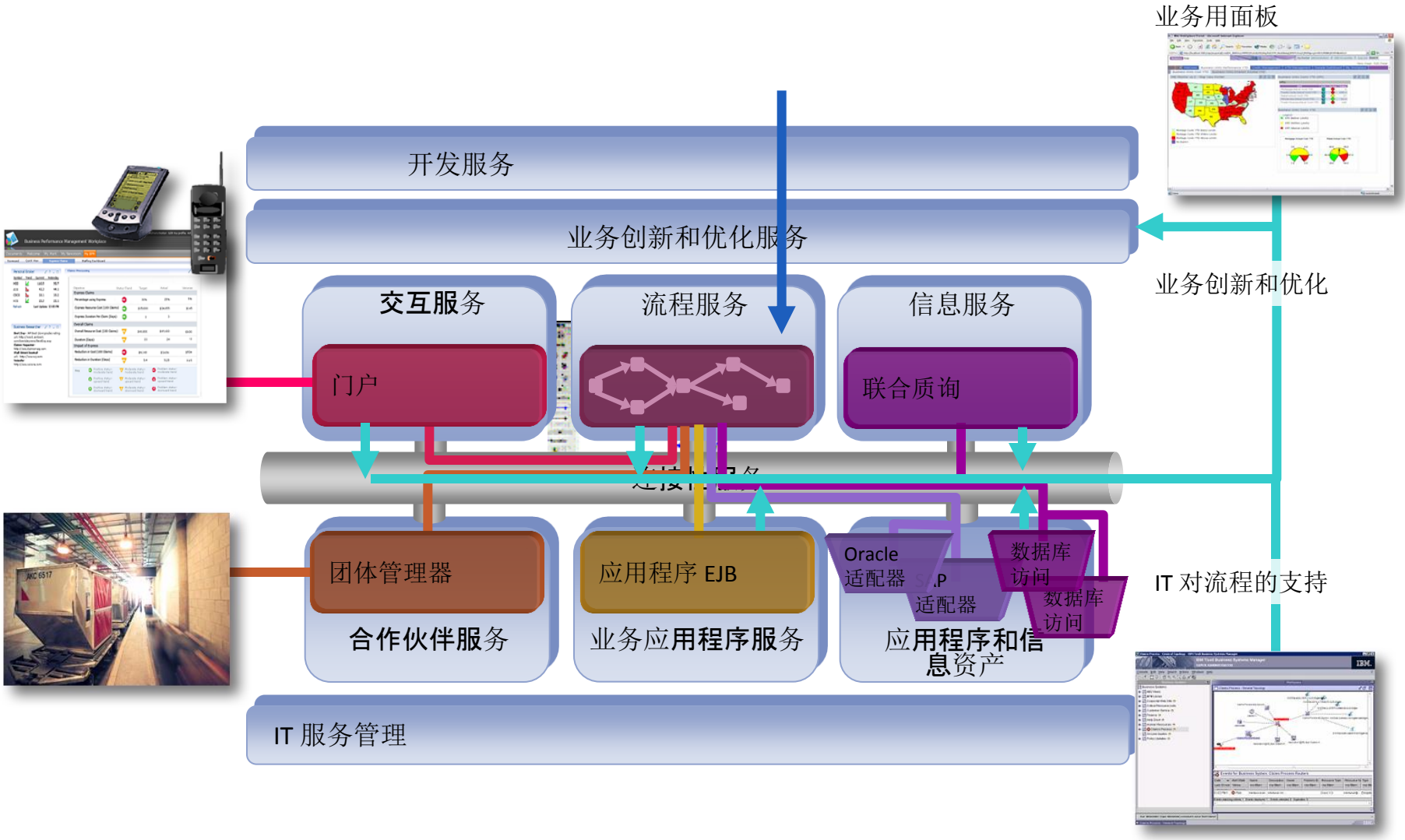
IBM SOA Foundation

- 企业服务总线
  - Enterprise Service Bus,ESB
  - ESB是消息中间件的发展，用总线模式管理和简化应用之间的集成拓扑结构，支持应用间在消息、事件和服务级别上的动态互联互通。
  - ESB是一种架构模式，不能简单地等同于特定的技术或者产品，但实现ESB确实需要各种产品在运行时和工具方面的支持。
    - 运行时支持：WebSphere ESB、WebSphere Message Broker
    - 工具：WebSphere Integration Developer

IBM SOA Foundation

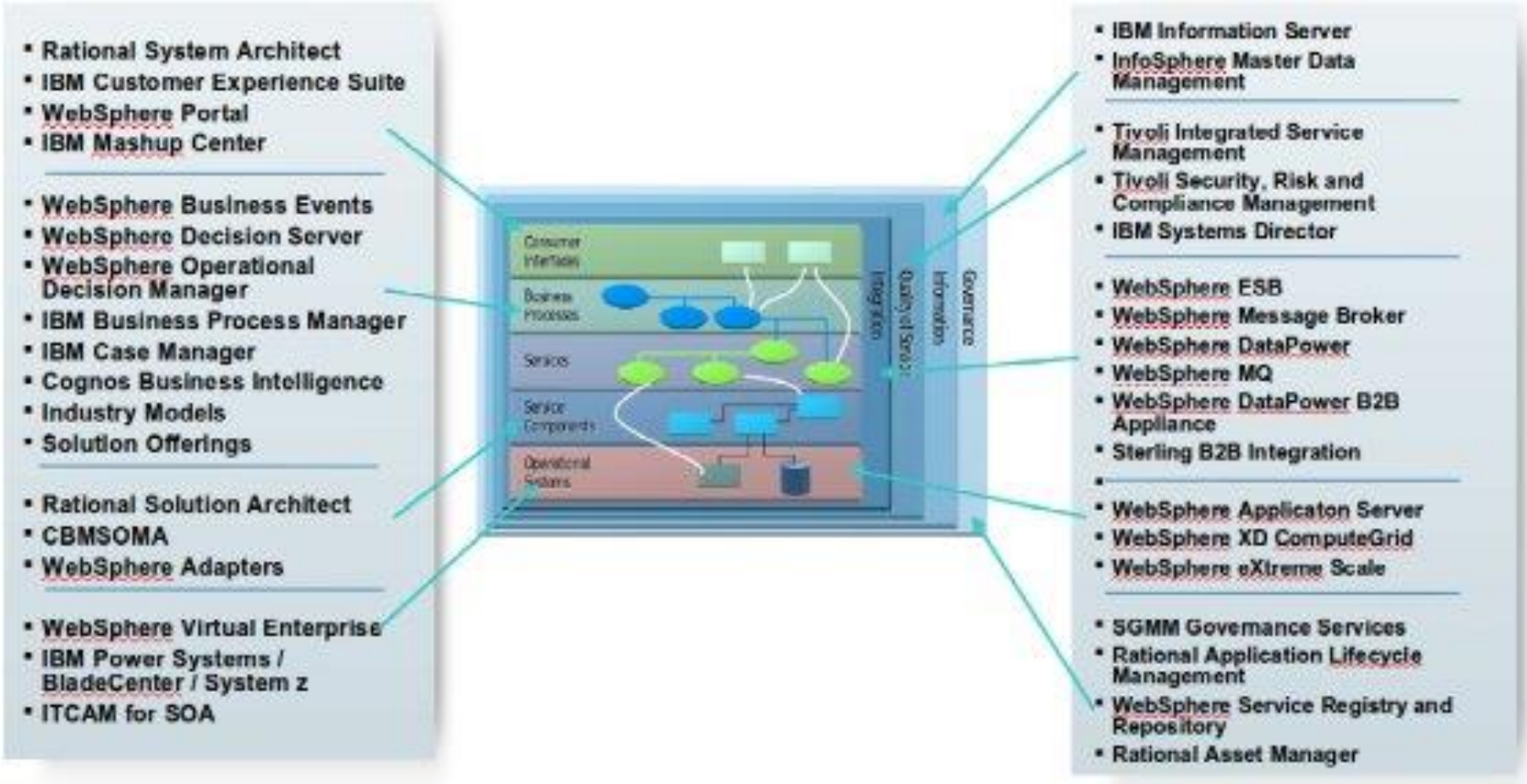
- 业务创新和优化服务：表示用于业务设计的工具和元数据结构，帮助捕获、解码、分析和精炼业务设计
- 开发服务：包括整套架构工具、开发工具、组装工具、方法论、调试辅助程序、资产库、发现代理和发布机制，用以构建基于SOA应用。
- 管理服务：提供管理工具和度量集以监控服务流、底层系统健康状况、资源利用、中断和瓶颈的鉴定、服务目标实现、管理策略执行以及故障恢复。
- 基础设施服务：构成运行SOA应用的IT环境，提供资源的高效利用，确保完善的操作环境，平衡工作负载以满足服务水平目标，隔离工作以避免干扰，执行维护，安全访问可信业务流程和数据，简化系统整体管理。

IBM SOA Foundation

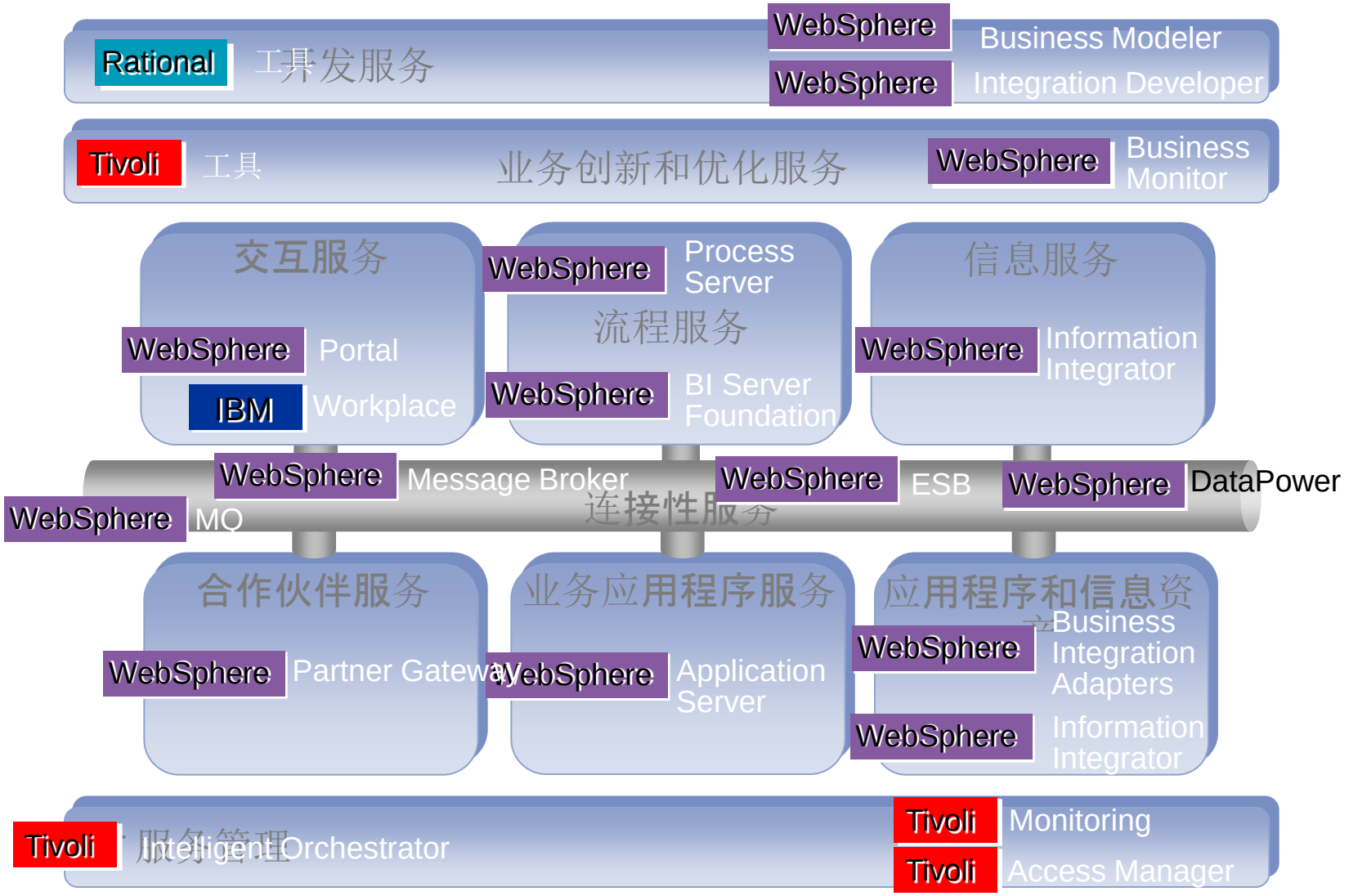


IBM的SOA相关产品

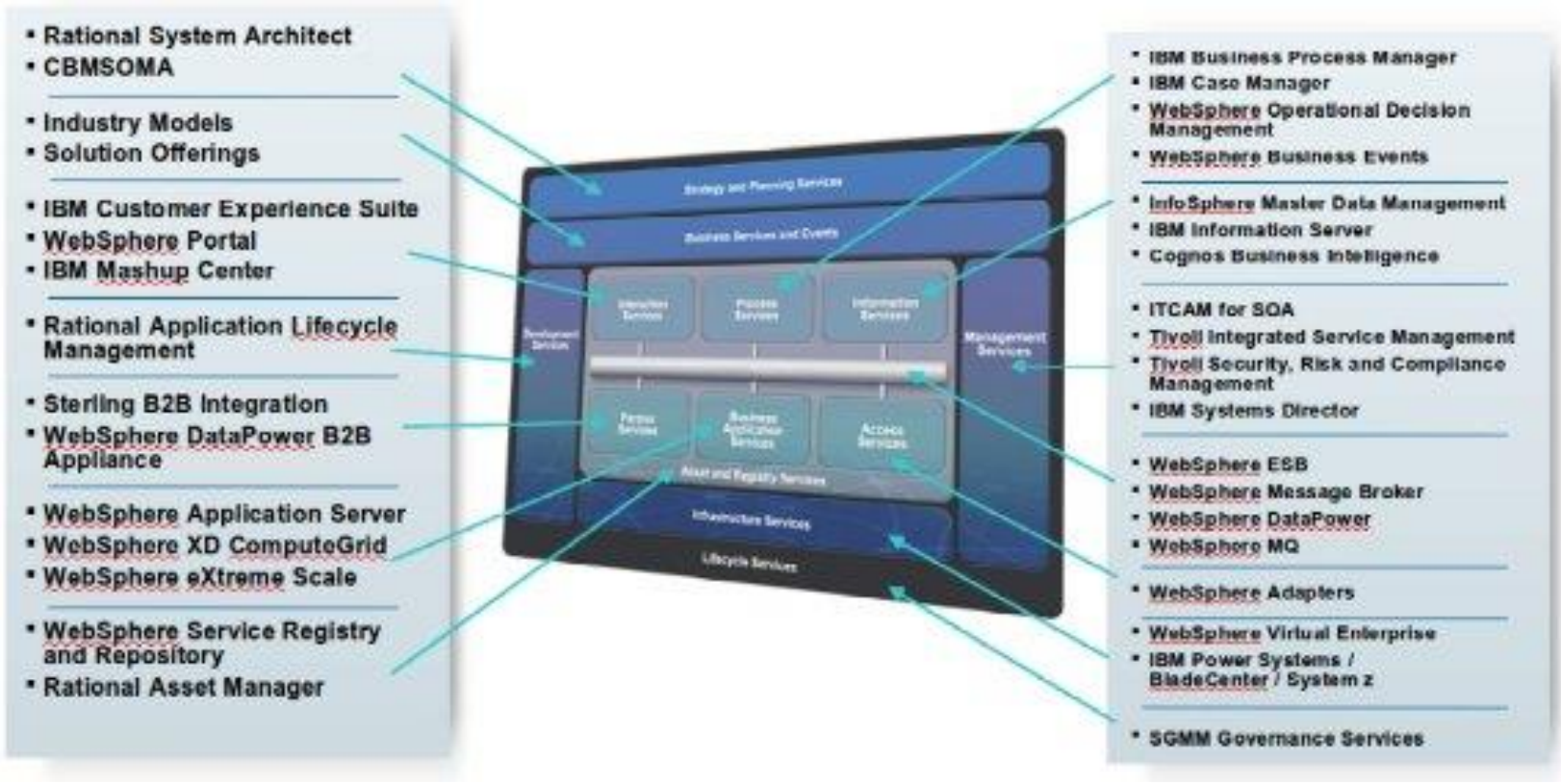
- IBM在SOA架构的每一层都为SOA生命周期的每个阶段提供合适的工具产品。



SOA参考框架下WebSphere产品线



IBM的SOA相关产品

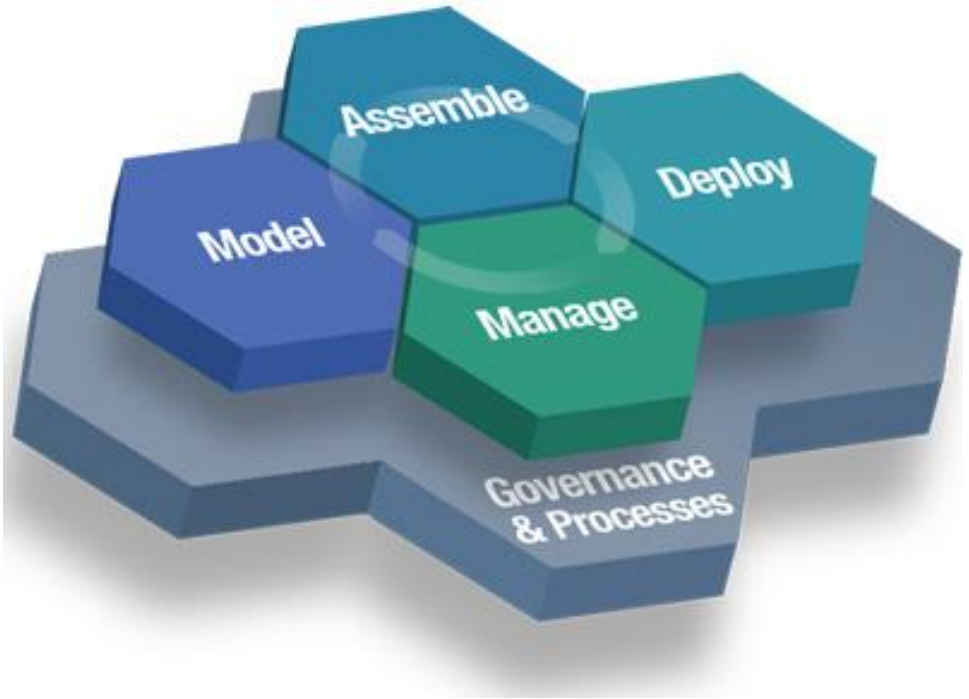


本章内容

1. SOA的基本概念
2. SOA参考模型
3. SOA生命周期
4. SOA的实现技术

SOA生命周期

- IBM SOA Foundation中提出了SOA生命周期
  - 建模Model
  - 组装Assemble
  - 部署Deploy
  - 管理Mange
  - 治理Governance



SOA生命周期

- 建模：收集分析业务需求，整理业务流程，并为此建模，包括业务分析与设计（要求、流程、目标和主要性能指标）及IT分析与设计（服务标识和规范）。
- 组装：把已有的和新开发的服务组合装配起来，实现业务流程，包括服务实现和组合应用程序的构建。
- 部署：把创建好的应用部署到可靠的集成环境中，包括应用程序和运行时（如ESB）的部署。

SOA生命周期

- 管理：监控和管理系统的运行情况，对发生的异常及时做出响应，并将收集道德信息反馈到生命周期中，从而可持续改进企业的SOA架构，包括操作环境维护、服务性能监视和服务策略执行。
- 治理：对所有生命周期阶段起到巩固支撑作用，为整个SOA系统提供指导，并有助于了解系统全貌。它在生命周期管理中确认责任人所使用的流程，以及相应的操作，为SOA计划提供引导和监督，帮助服务提供者和使用者避免遇到意外情况。

本章内容

1. SOA的基本概念
2. SOA参考模型
3. SOA生命周期
4. SOA的实现技术

SOA的实现技术

- Web Service
  - 由于基于标准传输协议SOAP和HTTP，已经成为主流的标准服务组件，为SOA的发展打下基础。
- Java领域的两套SOA标准
  - JBI(Java Business Integration)
    - 只处理Java的集成
    - JBI API适合低层架构开发，开发人员只需要学会使用JBI建好的ESB平台(如ServiceMix)来集成服务
  - SCA(Service Component Architecture)
    - 可以处理多种平台的集成
    - 适合开发人员创建基于SCA的服务组件
- BPEL(业务流程执行语言)
  - 是一种基于XML的编程语言，用于连接多种不同的应用、自动化跨应用的业务流程，来实现某个特定的目标

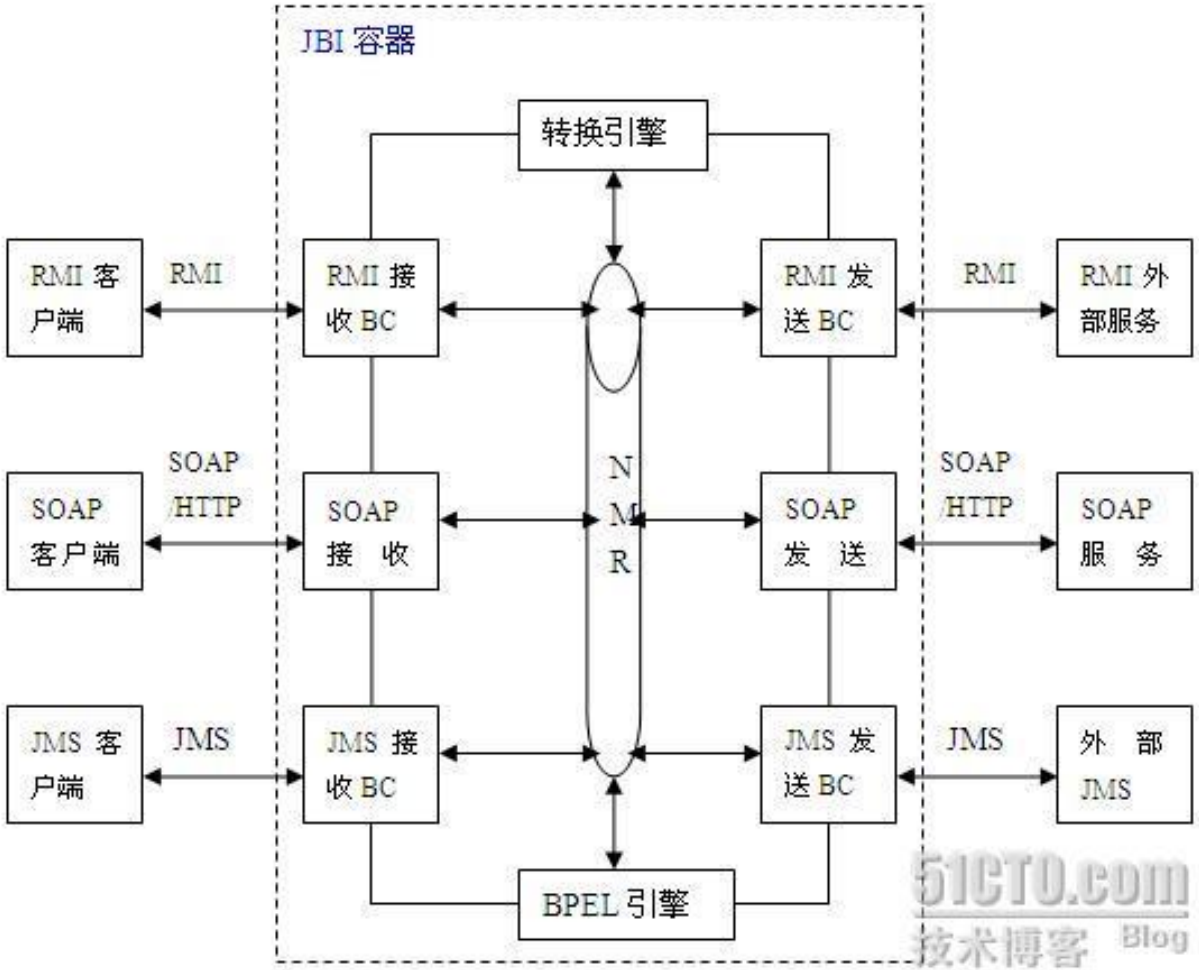
JBI(Java Business Integration)

- Java业务集成JBI是SUN发布的一个用于Java组件进行集成的一个标准。
- JBI的本质是一种服务总线思想，目标是创建一个用于各种Java组件服务集成的运行环境。
- JBI是一种思想，JBI思想的实现就是JBI容器。

JBI容器

- JBI容器是为弥补现有J2EE容器的不足而出现的。
- 现有应用服务器的容器(Servlet容器、EJB容器、JMS容器)存在如下不足：
  - 每种容器都有自己特殊的传输协议，相互之间不能直接通信。比如：Servlet容器只能接受HTTP/SOAP的传输协议，EJB容器只能处理RMI的传输协议，JMS只能处理JMS的传输协议。
  - 是一个纯粹的服务提供者，不是一个服务的集成者。也就是说，容器之间不能继承服务。
  - 容器间服务的调用需要编写客户端代码。
- JBI容器以一种可插拔的方式集成不同类型的服务，而不是通过编写客户端代码来实现服务的集成

JBI容器的组成

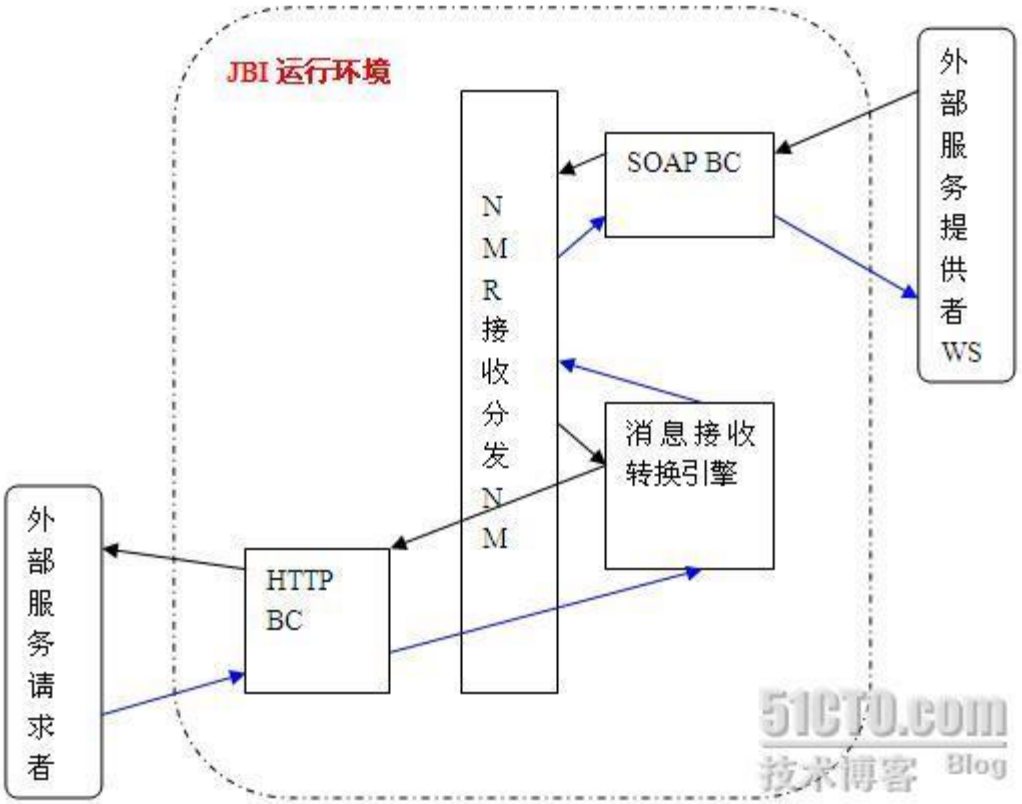


JBI容器的组成

- 绑定组件（BC：Binding Components）：专门用来接收各种不同传输协议的请求，原理是JBI实现了各种不同协议的绑定组件，绑定组件可以细分为接收BC和发送BC。接收BC主要负责发送请求和接收响应，发送BC主要用来调用外部的服务。
- 服务引擎（SE：Service Engines）：这类组件只处理JBI容器内部的消息。JBI容器通常在接收到消息后，需要对请求的消息做一些“处理”，然后再调用外部服务的提供者。根据功能的不同，将SE组件分为以下三种类型：
  - Transform SE：专门处理各种传输协议和格式变化。
  - BPEL SE：专门负责将Web Service进行流程编排。
  - Rules SE：专门负责通过规则将各种服务进行集成。
- 规格化消息路由器（Normalized Message Router）：是JBI内部消息系统的核心，所有的组建之间不能交换消息，只能通过NMR来传递。

JBI容器的工作原理

- 外部请求者将一个HTTP请求发送给JBI容器，容器的HTTP BC接收请求，并将请求的消息格式化为NM发送给消息接收转换引擎，然后再将NM发送给NMR，由NMR再将NM发送给SOAP BC，SOAP BC将NM转换为SOAP消息发送到外部的WS组件。执行后，消息按照原路返回。



面向服务的编程模型

- 为了促进面向服务应用的开发，IT公司联合在2005年11月发布了服务组件架构（Service Component Architecture SCA）和服务数据对象（Service Data Object SDO）规范。
- 通过SCA和SDO获得了更高的灵活性和更高的开发效率。可以在不改变应用程序情况下,使用不同的技术来作为组件的实现,或者改变通信协议等等,同时模块也可以根据容易的被重用和组装，易于修改和变更。

面向服务的编程模型——服务组件架构(SCA)

- 服务组件架构（Service Component Architecture——SCA）提供了一套可构建基于面向服务的应用系统的编程模型。它的核心概念是服务及其相关实现。服务由接口定义，而接口包含一组操作。服务实现可以引用其他服务，称为引用。服务可以有一个或多个属性，这些属性是可以在外部配置的数据值。大大简化服务开发。

SCA中的基本概念

- 服务组件
  - 服务组件是SCA中的基本组成元素和基本构建单位，也是我们具体实现业务逻辑的地方。
  - SCA服务组件的主要接口规范是基于WSDL的，也提供Java接口。
  - WebSphere Process Server充分利用了SCA的这种组件架构，并在产品中提供了一些与业务联系比较紧密的组件，比如业务流程，人工任务，业务状态机，业务规则等。这样用户就可以直接 利用这些服务组件，构建自己的业务流程或其它业务集成的应用。

SCA中的基本概念

- 服务模块
  - 服务模块由一个或多个具有内在业务联系的服务组件构成。把多少服务组件放在一个模块中，或者把哪些服务组件放在一起主要取决于业务需求和部署上灵活性的要求。模块是SCA中的运行单位，因为一个SCA模块背后对应的是一个J2EE的企业应用项目。
  - 由于模块是一个独立部署的单元，这给应用的部署带来很大的灵活性。
  - 在WID工具中，我们只要简单地通过接口与引用之间的连线， 就可以指定它们之间的调用关系而不需要写一行代码。另外，我们可以在这些连线上面设定需要的QoS要求，比如事务，安全等。

SCA中的基本概念

- 导入和导出
  - 导入（Import）的作用是使得模块中的服务组件可以调用模块外部的服务。导出（Export）的作用是使得模块外部的应用可以调用模块中的服务组件。
  - 由于涉及到模块内外的调用，需要指定专门的绑定信息，包括目标服务或源服务的调用方式，位置信息，调用的方法等。
  - 在WebSphere Process Server V6.0中，导入端点提供了四种绑定方式：JMS绑定，Web Service绑定，SCA绑定和无状态会话BEAN的绑定。导出端点提供了三种绑定方式：JMS绑定，Web Service绑定和SCA绑定。对于SCA模块之间的调用，把绑定方式设置为SCA绑定，但是对于非SCA模块与SCA模块之间的调用，只能选择其它绑定方式。

SCA中的基本概念

- 共享库
  - 共享库就是存放这些共享资源的地方。共享库可以通过与模块类似的方式在WID中创建，但是共享库包含的内容只有：数据定义，接口定义，数据映射和关系。
  - 当一个模块需要用到共享库中的资源的时候，我们只需要使模块依赖于共享库即可。
  - 从部署的角度，一个共享库会对应一个JAR包。在部署的时候，模块所对应的J2EE企业应用 会会自动包含所依赖的共享库JAR包。

SCA中的基本概念

- Standalone Reference
  - 模块中的服务组件是不能直接被外部Java代码使用的，为了外部的Java代码，比如JSP/Servlet使用模块中的服务组件，WID工具在模块中提供了一个特殊的端点，叫做Standalone Reference。这个端点只有引用（Reference），而没有接口（Interface）。只要把这个端点的引用连接到需要调用的服务组件的接 口，外部的Java代码通过这个引用的名称来调用相应的服务组件了。

同步调用和异步调用

- 同步调用是一种阻塞式的调用方式，即客户端代码一直阻塞等待直到被服务端返回为止。 这种调用方式相对比较直观，也是大部分编程语言直接支持的一种调用方式。
- 如果我们面对是基于粗粒度的服务组件，面对的是一些需要比较长时间才能有响应的应用场景，那么我们就需要一种非阻塞式调用方式，即异步调用方式。
- SCA编程模式提供了三种方式的异步调用：
  - 单向调用方式
  - 延迟响应方式
  - 请求回调方式

同步调用和异步调用

- 单向调用方式：客户端发出请求之后就不再关心服务端的情况，包括是否执行成功，返回值是什么等等
- 延迟响应方式：客户端在发出调用请求之后继续执行，但是经过一段时间之后，客户端再调用相应的方法去检索返回结果，并通过参数指定如何根据调用的结果而 执行进一步动作。由于是异步调用方式，因此，在第一次发出调用请求的时候，服务端需要返回一个称为票据（Ticket）的对象。这个对象会作为第二次发出 检索结果请求时的一个参数。
- 请求回调方式：能得到服务端的响应，这个响应是由服务端通过回调方式来触发的，而不像延迟响应方式由客户端来主动检索的。

SCA客户端的调用方式

- 从接口的角度，SCA的客户端编程模型有两种方式：
  - 静态调用方式：
    - 静态调用方式是一种类型安全的方式，也是在一般Java编程中最为常见的方式。
    - 静态方式就是直接拿到实际实现的接口类型，也即直接拿到Java接口。
  - 动态调用方式：
    - 动态调用方式是一种类型安全的方式。
    - 客户端通过invoke方法的字符串参数的方式来指定具体要调用的方法名称。
    - 有参数传递都是通过DataObject的方式，即SDO的方式。
- 当提供的接口类型是WSDL类型的，那么客户端的调用方式只能是动态调用方式。如果提供的接口类型时Java类型的，那么客户端的调用方式可以是 动态调用方式，也可以是静态调用方式。

面向服务的编程模型——服务数据对象(SDO)

- 服务数据对象（Service Data Object——SDO）是一种针对在不同的数据源之间使用统一的数据编程模型的规范说明，为通用的应用程序模提供健壮的支持，是应用程序、工具、框架等更容易的进行数据的增、删、查、改、约束、更新等操作，旨在创建一个统一规范的数据接入层 并 使用一种“易用”的方法，将混杂的数据源整合到工具集和框架中。使数据对象拥有清晰定义的数据模型，使得服务访问数据和在服务之间交换数据更方便、高效。

SDO

- SDO包括一个架构和API，SDO可以：
  - 简化J2EE数据编程模型
  - 对SOA架构中的数据进行抽象
  - 统一数据应用开发
  - 支持并集成XML
  - 吸收J2EE模式以及最佳实践

SDO的组件

- 数据对象-Data objects
  - 数据对象是SDO的基本组件。实际上，SDO的名字中就包含了数据对象。数据对象时结构化数据的SDO表示，数据对象是泛化的，它提供了对DMS构建的结 构化数据的一般视图。 例如，使用JDBC DMS必须要知道持久化技术（例如关系数据库）、以及如何配置并访问它，但SDO客户端则无需知道这些。数据对象将“data”保存在属性 properties中，数据对象提供了方便的创建和删除方法（createDataObject()\delete()），并用反射方法获取他们的类型（实例化 类、名称、属性、命名空间）。数据对象被数据图连接起来，并包含在数据图中。

SDO的组件

- 数据图-Data Graph
  - 数据图提供了一个数据对象树的容器，它由DMS生成，提供给SDO客户端使用。一旦Data graph被改动，它会被传回DMS从而更新数据源。SDO客户端能遍历数据图，读取并修改其数据对象。由于SDO客户端与DMS及数据源是非连接的，他们只能看到数据图，所以SDO是一个分布式的架构。此外，数据图可以包含来自不同数据源的数据对象。data graph包含一个根数据对象、与根数据对象相关的数据对象、以及一个变更汇总。当它在应用组件间传输给DMS，或者保存到磁盘，data graph会被序列化为XML。SDO规范对这种序列化提供了XML Schema。

SDO的组件

- 变更汇总 -Change summaries
  - data graph中包含了变更汇总，用于表示对被DMS返回的data graph所做的修改。当data graph最初传给客户端时，change summaries是空的，data graph一旦被修改change summaries就会填充值。DMS在后台利用change summaries将变更更新到数据源，通过提供数据图中的变更属性列表（连同其原始值）、创建和删除的数据对象，change summaries使得DMS能高效地、增量地更新数据源。
  - 当change summaries的logging被激活时，信息才会添加到数据图中的change summaries。change summaries提供了DMS可以讲logging置为on或off的方法，在示例程序中我们将讨论更多细节。

SDO的组件

- 属性、类型、序列-Properties,types,sequences
  - 数据对象将其内容保存在一系列属性中；每个属性有一个类型，要么是基本类型(int)，要么是常用的数据类型(Date)，或者是另一个数据对象的类型(对象引用)。
  - 每个数据对象都为其属性提供了读写的访问方法，还提供了一些重载版本的访问方法，允许传入属性名(String)、编号number(int)、或者属性元对象自身 来访问一个属性。String访问方法还提供了XPath-like的语法来访问属性，例如，可以在company数据对象上调用get("department[number=123]")，获取number为123的部门。
  - 序列Sequences更加高级，它允许在属性-值pair中保持一个顺序。

业务流程执行语言BPEL

- BPEL(Business Process Execution Language,业务流程执行语言)是一种使用XML编写的编程语言。用于自动化业务流程，也曾经被称作WSBPEL和BPEL4WS。
- BPEL是一门用于自动化业务流程的形式规约语言。用XML文档写入BPEL中的流程能在Web服务之间以标准化的交互方式得到精心组织。这些流程能够在任何一个符合BPEL规范的平台或产品上执行。 所以，通过允许用户在各种各样的创作工具和执行平台之间移动这些流程，BPEL使得他们保护了在流程自动化上的投资。