**面向服务体系架构（SOA）和业务组件（BC）的思考**

作者：肖建国， 来源：[企业工程论坛](http://www.ee-forum.org)， 发表时间：2010-10-08

**摘要：**在基于面向服务体系架构（SOA）中，“组件化”是一个很重要的概念，如何进行“组件化”开发是搭建企业级业 务基础平台时需要考虑的一个重要课题，本文通过建立业务组件（BC）接口模型及内部结构模型，提供了一个在新开发系统环境下基于Web服务和OSGi标准 的组件化开发模型。

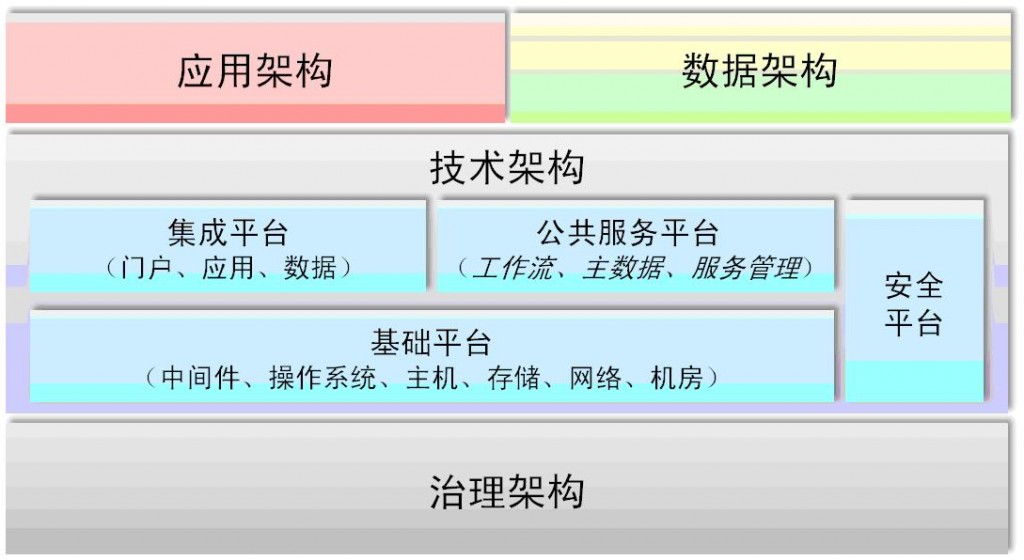
**什么是业务组件（BC）**

组件化、模块化是软件开发中一个很重要的概念，基于面向服务体系架构（Service Oriented Architecture，SOA）下，如何实现组件化，有各种实现方式，下面通过对各种组件概念的对比，从技术角度提出业务组件（Business Component，BC）定义，并结合对总线模式的分析，给出企业服务总线和类总线的实现方案。

**企业架构（EA）**

关于企业架构（Enterprise Architecture，EA）和面向服务体系架构（SOA）在《面向服务体系架构（SOA）和数据仓库（DW）的思考》（以下简称《SOA和DW》） 一文中做了介绍，企业架构包含企业战略、业务架构、IT战略、IT架构四个部分，IT架构如下图IT架构模型所示，包含数据架构、应用架构、技术架构和治 理架构等四个方面，其中技术架构包含集成平台、公共服务平台、基础平台（软件和硬件、网络）和安全平台等，《SOA和DW》着重对如何构建数据架构特别是 数据存储做了详细的阐述，本文基于《SOA和DW》进一步对如何搭建SOA体系进行研究，将着重描述如何基于可扩展的、灵活的企业级的集成平台、公共服务 平台进行组件化开发。

**图 1. IT架构模型**

[](http://www.ee-forum.org/pub/davidxiaocn/2010-10-p1778.html/xiao-soa-bc-1)

**业务组件（BC）**

当前，提到组件（Component）的有很多概念，比如分布式组件DCOM、J2EE、CORBA等，IBM的业务组件模型（Component Business Model，CBM），SOA中的服务组件架构（Service Component Architecture，SCA）等。本文提到的业务组件（Business Component，BC）定义为一个可以独立运行的系统或者模块，业务组件的目的是以方便业务组件独立升级和减少不必要的组件之间的交互为基本原则，通 过一定程度的分离，实现《SOA和DW》中提到的重用（SoftWARe Reuse）。

如果业务组件是共用的，是其它业务组件需要重用的，称之为公共业务组件（简称公共组件），所有的公共组件组成企业架构中技术架构的公共服务平台，比如主数据管理、系统管理、统一认证管理、通用报表等，这些都是公共组件。

**组件业务模型（CBM）**

组件业务建模（Component Business Modeling，CBM）是IBM SOA构建的一个方法论，通过将组织活动重新分组到数量可管理的离散、模块化和可重用的业务组件中，从而确定改进和创新机会，把业务从领导，控制和执行三 个方面进行模块化分析，从而有效的实现业务的有组织的提供服务的能力。CBM 的价值是提供一个可以推广的框架，用来创造顺应组织战略的可以运营的指导方向，同时 CBM 也用来按照业务和资源的优先级别和相互关联的程度来构建和顺应战略的发展方向，其中包括建立一个沟通的机制来理解整个业务发展的方向。通过CBM建立了 SOA 的规划的方向，为实施 SOA 奠定基础。

本文所提到的业务组件在粒度上基本对应着组件业务模型（CBM）的粒度，但是本文中的业务组件（BC）更多从技术实现角度考虑，或大于，或小于业务组件模型（CBM）提到的业务组件概念。

**服务组件框架（SCA）**

服务组件框架（Service Component Architecture，SCA）由BEA、IBM、Oracle等中间件厂商联合制定的一套符合SOA思想的规范。服务组件框架（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。SCA组件组成程序集，程序集是服务级的应用程序，它是服务的集合，这些服务 被连接在一起，并进行了正确的配置。在SCA标准下，SCA由域（Domain）、组合构件（Composite）、构件（Component）三个级别 组成，构件对应着细粒度的Web服务，域对应着粗粒度的Web服务。SCA程序集运行在两个级别：第一种情况，程序集是“大规模编程” （Programming in the Large或Megaprogramming）的一组松散连接的服务组件；另一种情况，程序集是“小规模编程”（Programming in the Small）内的一组松散连接的组件。二者的区别在于， “大规模编程”对应着应用，“小规模编程”对应着模块，一般来说，服务组件对应着“小规模编程”，即模块的概念。

本文所提到的业务组件（BC），是比SCA组件更大范围的概念，这几个概念的颗粒度从大到小的排列顺序如下：**系统**（每个企业只有一个系统）、**应用、业务组件（BC）**、**模块、SCA组件**（粗粒度服务）。

**总线模式（Bus）和SOA、OSGi**

总线（Bus）：一般指通过分时复用的方式，将信息以一个或多个源部件传送到一个或多个目的部件的一组传输线。基于总线模式的有很多应用，在微机的 技术中，有三种总线，地址总线Address Bus，数据总线Data Bus，控制总线Control Bus。在通信架构下，交换机也是一种总线，在SOA中，总线一般指企业服务总线（Enterprise Service Bus，ESB），企业服务总线可以连接所有协议的各种接口，但是最理想的是基于XML的Web服务标准。

OSGi——Open Service Gateway Initiative，1999年OSGi 联盟成立，旨在建立一个开放的服务规范，为通过网络向设备提供服务建立开放的标准，是开放业务网关的发起者。OSGi是一个Java框架，该框架能装载以 Bundle为单位的资源。Bundle能提供服务或响应处理请求，而他们之间的依赖都是被管理起来的，正如一个Bundle能从容器中获得它所需要的管 理。每个Bundle都可以有它自己的内部类路径，所以它可以作为独立的服务单元。所有的这些符合OSGi规范的Bundle理论上都可以安装在任何符合 OSGi规范的容器中。OSGi具有可动态改变系统行为，热插拔的插件体系结构，高可复用性，高效性等等。在J2EE环境下，基于总线（Bus）模式的思 考，可以进一步推广到Java Class，基于OSGi的微内核，建立一个类总线（Java Class Bus，JCB）。

通过以上概念的分析，我们可以看到，本文提到的业务组件（BC），是指具体的一个软件实现，业务组件（BC）跟IBM的业务组件模型（CBM）中提 到的业务组件有一定的对应关系，但是一般来说，业务组件（BC）可能包含CBM中的多个业务组件或者一个CBM的业务组件封装成多个业务组件（BC）。另 外CBM更多的是从业务角度来考虑，是业务上的概念，业务组件（BC）则是从技术实现角度考虑。服务组件框架（SCA）定义的粒度和业务组件（BC）相比 来说，SCA划分的还是很细，业务组件（BC）是更粗粒度的一个软件实现概念。

**业务组件（BC）模型**

根据业务组件的作用不同，可以将业务组件分成公共业务组件和普通业务组件，公共业务组件包含统一用户组件、统一认证组件、门户组件、流程组件、报表组件、BI组件、GIS组件等，这些组件的共同特点是多个业务组件或者系统会用到这个业务组件。

组件的粒度和对外接口设计决定了组件的可复用和松耦合（Loose Coupling）特性。粒度过大，灵活性小，难以实现复用，粒度过小，管理成本提升，使得复用性也很难改善；接口和实现的分离,保证各项业务组件在提供 标准化的服务接口的前提下可以替换各种可选的实现,而不会影响系统其它部分的实现，接口设计不当，对于组件的耦合会有很大的影响。

**业务组件的粒度**

业务组件的粒度根据需要可以不同，既可能是独立运行的子系统,也可能是程序模块。业务组件是提高应用系统灵活性和复用的重要基础。业务组件粒度太 小，造成组件数量多，组件之间交互多，管理困难，性能低下；业务组件粒度粗，功能复杂，功能之间关系紧密，升级困难（可以独立升级往往会作为确定一个业务 组件范围的重要因素），很难实现重用。因此找到一个合适的业务组件粒度是很重要的事情。

根据前文所定义的业务组件定义，我们把整个企业的所有软件称之为系统，即一个企业只有一个系统；系统下面划分成若干应用，每个应用完成一个相对独立 的业务功能，比如财务管理、人力资源管理等，一般来说是一个厂商独立完成（后文还会提到，如果是基于一个业务基础平台，多个厂商可以在一个应用中）；应用 下面划分成若干业务组件，业务组件是相对独立的功能，其可以进一步划分成若干模块，从而形成了系统－应用－业务组件－模块这样四个层次的模型。根据SCA 的定义，模块下面可以进一步划分成程序集为更小的粒度。从软件复用角度来看，业务组件是独立部署的最小颗粒，模块是复用的最小颗粒。

除了业务组件需要粒度控制外，Web服务的粒度控制也是一项十分重要的设计任务。通常来说,对于将暴露在整个系统外部的服务推荐使用粗粒度的接口, 而相对较细粒度的服务接口通常用于企业和机构系统架构的内部。从技术上讲,粗粒度的服务接口可能是一个特定服务的完整执行,而细粒度的服务接口可能是实现 这个粗粒度服务接口的具体的内部操作。虽然细粒度的接口能为服务请求者提供了更加细化和更多的灵活性,但同时也意味着引入较难控制的交互模式易变性,也就 是说服务的交互模式可能随着不同的服务请求者而不同。如果暴露这些易于变化的服务接口给系统的外部用户,就可能造成外部服务请求者难于支持不断变化的服务 提供者所暴露的细粒度服务接口。而粗粒度服务接口保证了服务请求者将以一致的方式使用系统中所暴露出的服务。

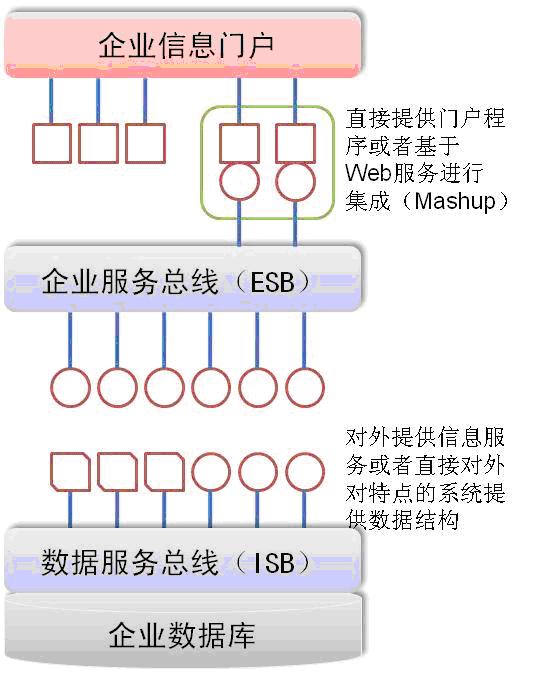
**业务组件的松耦合设计**

耦合性（Coupling）是程序结构中各个模块之间相互关联的度量，它取决于各个模块之间接口的复杂程度、调用模块的方式以及哪些信息通过接口。 耦合性由松到紧可以分成七种：非直接耦合（Nondirect Coupling）、数据耦合（Date Coupling）、标记耦合（Stamp Coupling）、控制耦合（Control Coupling）、外部耦合（External Coupling）、公共耦合（Common Coupling）、内容耦合（CONTENT COUPLING）。非直接耦合是指两个模块之间没有直接关系，这种耦合的模块独立性最强。数据耦合，彼此之间是通过数据参数(不是控制参数、公共数据结 构或外部变量)来交换输入、输出信息的，模块之间的独立性比较强。标记耦合是指一组模块通过参数表传递记录信息，就是标记耦合，这要求这些模块都必须 清楚该记录的结构，并按结构要求对此记录进行操作，应尽量避免这种耦合，它使在数据结构上的操作复杂化了。在业务组件设计模型中业务组件之间尽量实现非直 接耦合（总线模式，推荐使用）和数据耦合（共享库模式，控制使用），通过定义清晰的Web服务进行交互，业务组件内部的模块之间可以通过标准化的Web服 务或者数据表来进行共享。

**业务组件接口模型**

业务组件作为一个可以独立的运行的模块，通过一系列的接口，可以独立完成本业务组件的功能。业务组件通过接口，可以组装到和基于标准的**集成平台**上，并和其他的组件或者系统共同完成企业的业务。基于标准的企业**集成平台**包含企业信息门户（门户标准）企业服务总线（Web服务标准）和数据服务总线（含数据模型），如下图所示：

**图 2. 企业集成平台模型**

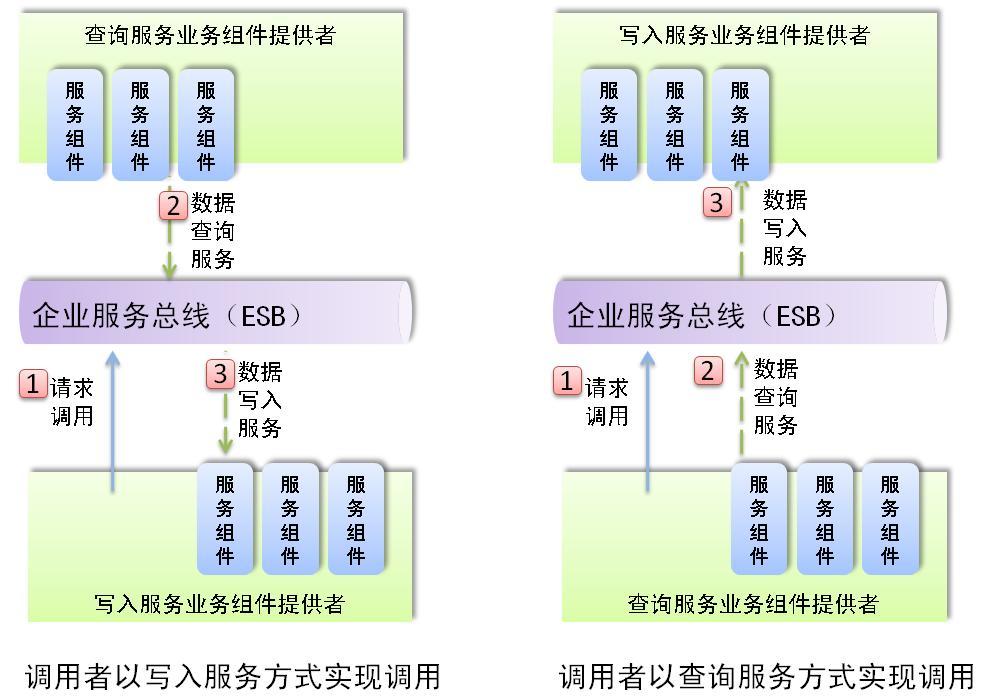
[](http://www.ee-forum.org/pub/davidxiaocn/2010-10-p1778.html/xiao-soa-bc-2)

当企业**集成平台**建立之后，基于标准，业务组件就可以简单的以“插拔”的方式组合到整体应用中。

**业务组件接口通信方式松耦合实现方式**

Web服务根据Web服务内部数据的流向可以分成两类，一是获取服务，即别的业务组件或系统调用Web服务，对外提供数据；二是写入服务，即其他业 务组件或系统调用本业务组件或系统Web服务写入数据。前者比较简单，只要提供服务即可，但是对于后者，在实现过程中将会有两种实现方式：一是主动到 ESB调用Web服务，读取数据，然后直接写入系统，二是提供Web服务到ESB，等待业务组件或系统调用写入。采用第一种方式，不需要对外提供Web服 务，直接调用ESB的Web服务即可，但是服务的调用以及写入是通过代码固化到系统中的；后者对外提供服务，不管调用系统是谁，只要调用既可写入数据，不 会受到外部系统的变化的影响。如果是采用直接到ESB调用服务，然后用代码写入的方式，一旦业务流程发生变化，比如改为别的系统或者模块直接写入，这种方 式就无法适应，需要重新增加一个写入的Web服务，因此比较好的解决方式是提供对外的一个写入Web服务，写入Web服务是独立的，满足别的系统主动写入 的要求，如果是自己主动写入，则采用先到ESB上请求获取数据（本服务仅仅是通知ESB，要获取数据）然后由ESB实现对本次统的写入Web服务，这样， 不管外部的系统如何变幻，接口是不变的，如下图所示：

**图 3. 服务调用顺序**

[](http://www.ee-forum.org/pub/davidxiaocn/2010-10-p1778.html/xiao-soa-bc-3)

除非是写入服务提供者业务需求非常明确，只有本系统调用才会写入，一般建议按照以上独立的写入服务方式来实现。采用独立的写入服务能更好的适应未来被动写入、或者写入操作需要经过评审或者确认之后的操作。

比如客户信息数据，如果是在CRM系统中创建，财务系统需要客户数据，有三种调用方式，一是财务系统直接到ESB调用CRM的客户信息查询Web服 务，然后写入系统。二是事件机制，CRM系统中的数据变化时，对外提供的客户信息变更服务，服务调用中传递的消息就是变更的信息，调用财务系统的写入服 务。如果还有其它的系统需要客户信息，可以在ESB中定义出发布/订阅关系。三是财务系统先请求ESB调用CRM的查询服务，然后由ESB调用财务系统的 客户信息写入服务，写入数据。如果未来业务流程发生变化，改由CRM直接将客户信息写入财务系统，则直接调用财务的写入服务即可，需要做的仅仅是配置一下 ESB即可，现有的程序不需要改变。第一种方式下，如果改成CRM写入，财务系统需要重新编码，第二种方式如果别的系统来主动查询客户数据，需要另外增加 一个客户信息的查询服务，第三种情况，无论是如何改变化，需要的仅仅是增加一个请求调用即可，对所有的系统影响最小，因此是受外界需求发生变化后影响最小 的方式，更好的解决了松耦合的问题。

**业务组件接口事务处理－无状态的会话设计**

为了保证业务组件的独立和松耦合，SOA中的具体服务应该都是独立的、自包含的请求,在实现这些服务的时候不需要前一个请求的状态,也就是说服务不 应该依赖于其他服务的上下文和状态,即 SOA中的服务应该是无状态的服务。不同的业务组件之间是采用Web服务的方式进行交互，对于事务的控制很困难，应尽量将事务控制在一个业务组件中（关于 事务的控制，详见下文关于组件内部松耦合设计的方案），如果是在不同的业务组件，需要事务控制，则考虑采用流程编排的方式实现，基于BPEL实现事务控 制。

**J2EE架构下业务组件（BC）实现**

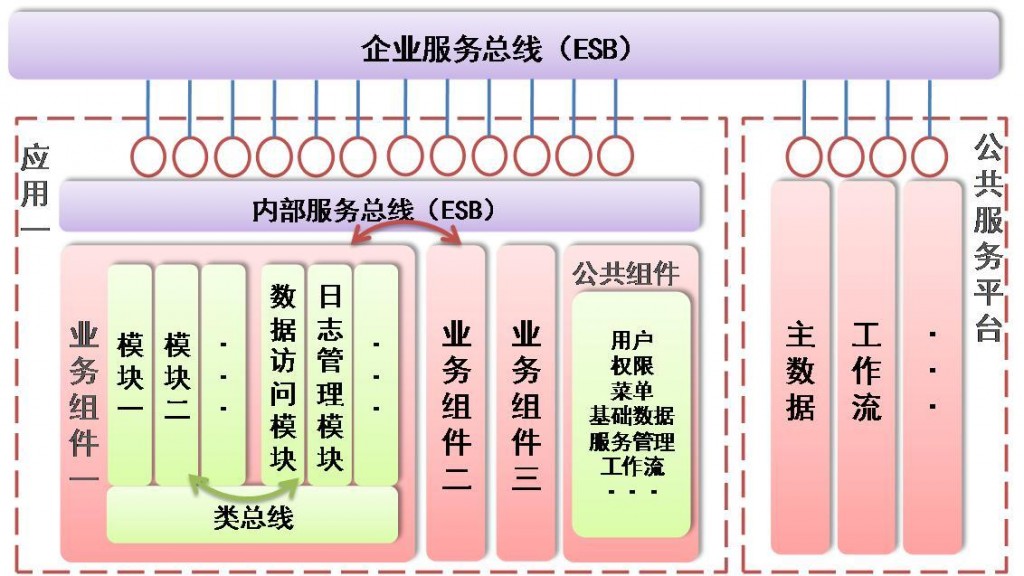
**业务组件松耦合设计－OSGI**

业务组件以Web服务的方式提供接口，通过企业服务总线连接，业务组件内部为了实现高可复用性和高效性，采用基于OSGi标准进行构建模块，实现内 部模块之间的松耦合，即在业务组件内部基于OSGi标准进行模块化设计，将业务组件进一步分解为松耦合的模块（Bundle），使得业务组件本身更加灵 活。

基于OSGi标准，业务组件内部的模块通过一个具有动态加载类功能的微内核连接，统一管理各个模块，为了便于管理，将不同模块之间的类接口采用服务 注册的方式进行管理，具有类动态加载功能的微内核和类接口管理组成类总线（JCB）的基本功能，为了更好的实现重用，有些模块是共用的，比如数据访问模 块、日志管理模块等。

在一个应用中，不同业务组件公用的功能，作为应用内部的公共组件，一个应用中部署一个公共组件即可，各个业务组件共用。在一个业务组件中，不同模块 公用的功能，作为公共模块，相当于工具类，公共模块需要在每个业务组件中部署。公共服务平台作为企业级的公共服务对外提供企业级的Web服务，比如主数据 管理等。业务组件构成如下图所示：

**图 4. 业务组件模型（公共类－公共组件－公共服务平台）**

[](http://www.ee-forum.org/pub/davidxiaocn/2010-10-p1778.html/xiao-soa-bc-4)

注：

业务组件中的公共组件和公共服务平台的差异是公共组件是应用内部的，提供应用级别的服务；公共服务平台则是面向企业整个系统的，是提供系统级的服务，两者有时候可以互相替换，主要是看其处于那个级别。

**基于IBM产品体系的实现**

本文提到的集成平台，基于IBM的产品共体系在实际搭建的时候需要包含应用服务器（产品：WAS）、流程整合服务器（产品：WPS，实现服务总线和流程编排）等，关于集成平台的详细描述，详见《SOA和DW》一文中“基于IBM产品体系的实现”的描述。

**业务组件（BC）在WAS中的部署**

首先来看一下在J2EE架构模式下文件格式（以WAS为例）。在J2EE架构下，文件格式有三种，分别是EAR、WAR、JAR，另外还有一个特殊的JAR即基于OSGi标准的Bundle，共四类文件：

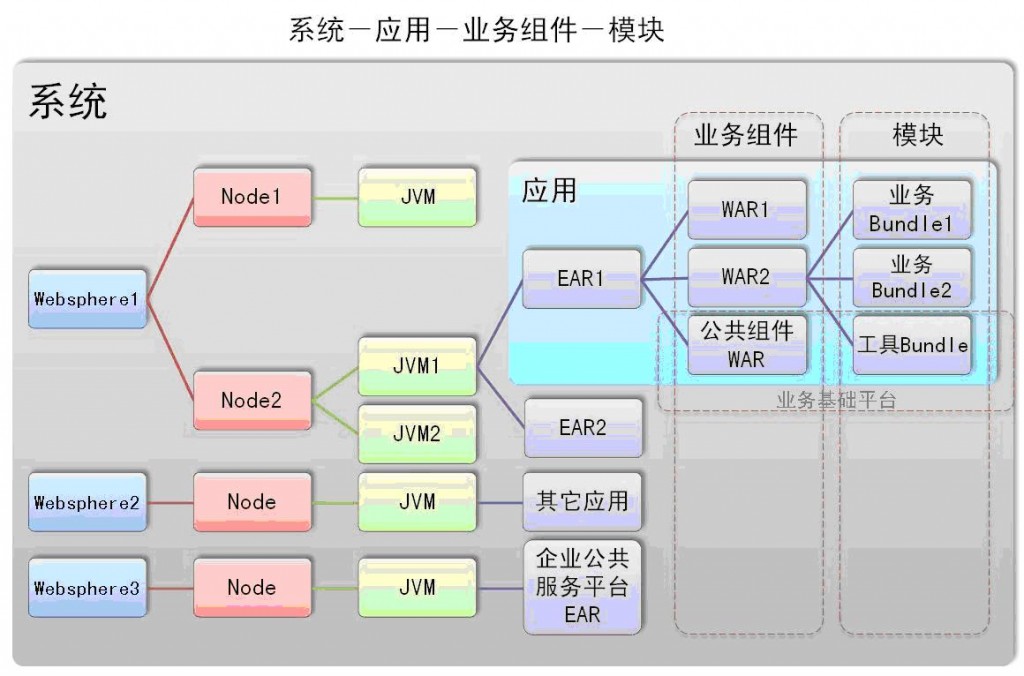
* **EAR文件（file Enterprise Achieve）**除了包含JAR、WAR以外，还包括EJB组件、部署文件application-client.xml、web.xml、application.xml等全部企业应用程序。
* **WAR文件（file web Achieve）**包 含Servlet、JSP页面、JSP标记库、JAR库文件、HTML/XML文档和其他公用资源文件，如图片、音频文件等全部Web应用程序。在一个 EAR文件中可以有多个WAR。〔在WAS环境下，如果设置在EAR中用一个类加载器，这样不同的WAR之间可以直接调用Java Class，WAR之间是紧耦合的，不建议采用。〕
* **JAR文件（Java Achieve）**按Java格式压缩的类包，包含内容 class、properties文件，是文件封装的最小单元。但是普通的JAR文件，只是一个文件的集合，没有具体的意义。
* **Bundle，**基于OSGi标准的一种特殊的JAR文件，每个 Bundle也包含一个 META-INF/MANIFEST.MF文件，这个文件会宣布导出哪些包(Package)以及导入哪些包。只有那些导出包中的类才能被其他 Bundle 所使用，而其他包都只面向包的内部成员，包里的类也只能在自身Bundle 中使用。一个简化的处理思路是直接采用包(Package)，但是无法实现类的动态加载和对接口进行管理，不具有松耦合特性。〔WAS从版本 6.1 开始支持 OSGi〕

从以上文件结构可以看到，JAR之间可以进行类的调用，很容易实现不同JAR之间的事务处理，且具有更高的性能，结合OSGi，通过“类总线”进行 管理所有JAR（Bundle）对外开放的接口，从而以“总线”（Bus）的方式管理不同JAR之间的类调用。不同的WAR之间各种资源不共享，为了实现 重用，需要设置一些公共模块，即公共Bundle；不同的WAR之间交互，需要以Web服务的方式进行连接，需要建立企业服务总线（ESB）管理所有的 Web服务，实现WAR之间调用。在一个EAR中，可以设置不同WAR之间共享Session，从而方便的实现单点登录以及其它的公共信息，这些信息将在 这个EAR环境中共享。

根据前文所述业务组件（BC）的定义，业务组件适合于在WAR文件层面进行划分，即一个WAR作为一个业务组件，一个或者几个WAR组成一个应用 （EAR），多个应用构成企业的系统；业务组件内部进一步划分为多个模块（Bundle），每个模块相对独立，可以独立维护，独立升级和安装，以插件的方 式通过类总线进行关联。为了实现重用，在EAR层面，将企业级的业务组件单独部署，比如主数据、统一认证、工作流等，建立公共服务平台；在WAR层面，各 厂商的公共的业务组件单独封装在公共组件（WAR）中，如系统管理、系统参数管理等。在模块级别采用公共模块的方式，在各个WAR中以工具模块 （Bundle）方式提供，如数据库访问、日志（**Log4j**）等。公共组件（WAR）、内部服务总线（ESB）和类总线（JCB）、工具模块（Bundle）组成应用系统的业务基础平台（Business Platform）（如浪潮的Loushang平台）。

在系统部署的时候，一台服务器上安装一个Websphere实例，一个实例根据主机的性能可以安装多个节点（Node），每个节点（Node）可以 安装多个虚拟机（JVM），每个虚拟机可以安装多个EAR应用，每个EAR有多个WAR，不同WAR之间文件不会冲突，WAR内部采用OSGI标准分成多 个模块（Bundle）。不同公司的系统是不同的EAR，同一个公司可以有多个EAR。如下图所示：

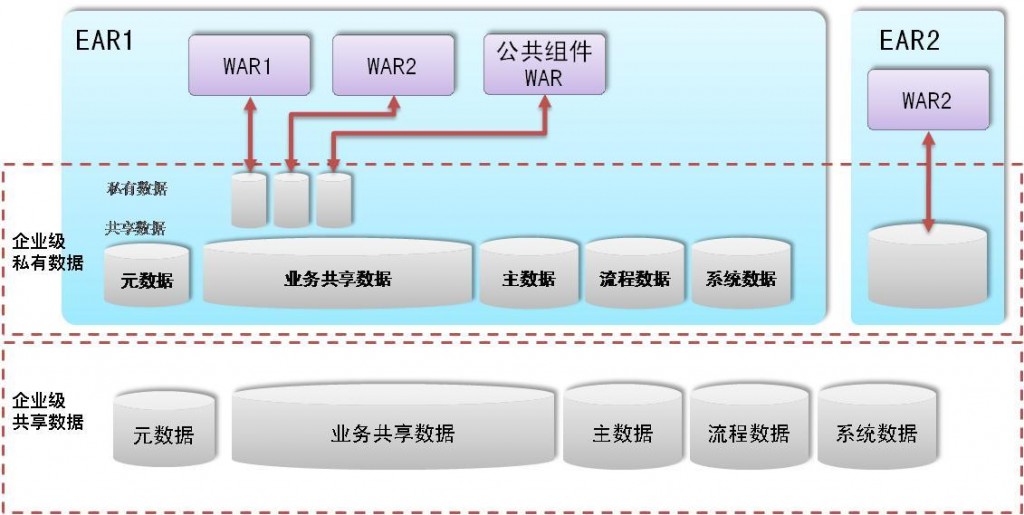
**图 5. WAS部署模型（系统－应用－业务组件－模块）**

[](http://www.ee-forum.org/pub/davidxiaocn/2010-10-p1778.html/xiao-soa-bc-5)

EAR之间数据交换采用通过企业级服务总线（ESB）的Web服务，大数据量数据共享在数据总线上通过企业级的共享数据库进行共享，通过数据总线共享的数据不是实时的，是有一定的延误或者准实时的。共享库是企业的资产，不隶属于任何一个厂商。

WAR之间数据交换采用通过企业或者内部的服务总线（ESB）的Web服务，不同的WAR共用一个数据库，但是数据表根据WAR的职责进行划分，每 个WAR可以只读所有的共享的数据表，但是只能写自己控制的表，对其它WAR控制的表、表结构很复杂或者易变的表的读作操，都应通过Web服务进行，以实 现WAR之间的松耦合。一个EAR中的共享数据库（对企业来说是私有数据层）在各个WAR之间结合更加紧密，一般采用直接访问的方式，不在私有数据层存放 共享库数据，私有数据仅仅是一些控制类或者跟业务无关，不需要共享的数据。企业级共享库一般从性能、不同厂家便于控制等角度考虑，数据同步是准实时的，数 据在各个EAR的私有数据层一般会有一个拷贝，让各个EAR之间相对更加独立。以上仅仅是一般原则，如果是一个厂商，两个EAR之间也可以像一个EAR共 享一个数据库。EAR和共享库之间的关系如下图EAR和共享库的关系所示：

**图 6. EAR和共享库的关系**

[](http://www.ee-forum.org/pub/davidxiaocn/2010-10-p1778.html/xiao-soa-bc-6)

Bundule之间数据交换可以类似WAR的Web服务调用，也可以直接通过类总线，调用Bundule对外发布的接口，特别是需要具有事务处理能力和更高的性能要求的情况。一个WAR内原则上不再划分数据表的控制，由WAR自己内部进行管理。

一般来说，WAR是相对比较稳定的，不需要完全进行替换升级，日常的变更是通过Bundle来实现的，由于Bundle在应用方面是基于插件的方式 进行开发的，数据方面由于业务组件本身是基于标准信息服务或者作为企业资源的开放的数据表、视图（Web服务和数据模型，作为企业的两个资产），因此可以 实现热插拔，保证了系统可以连续运行，而不至于因为系统升级而影响到业务的运行或者最小程度的营销业务的运行。

在实际部署的情况下，有可能会出现整个企业只有一个EAR（功能简单，最极端情况），或者一个厂商把应用分别部署在不同的EAR，需要根据企业规 模，主机部署等，从性能、易于管理等角度考虑。由于WAR之间是松耦合的，基于企业服务总线（ESB）和信息服务总线（ISB），可以实现灵活的组装，因 此一个或者多个WAR可以进行灵活组装部署。

以上基于J2EE的应用服务器WAS（IBM WebSphere Application Server，WAS），对J2EE架构模式下的文件体系进行分析，针对前文提到的业务组件模型实现提供一个实现建议。

**业务组件（BC）实现举例**

一般企业常用到的应用包含人力资源、协同办公、财务管理、营销管理、生产管理等几个主要的应用，根据前文所述架构，基于J2EE环境按照以下方式进 行构建：首先，人力资源、协同办公、财务管理、营销管理、生产管理可以认五个应用，一般来说可能由五个厂商分别提供，因此可以作为五个独立的EAR，分别 进行部署，考虑到为减少不同厂商之间的影响，一般会分别安装在五台服务器上，或者安装在不同的虚拟机上，（有时候可能是一个厂商提供，会在一个EAR中， 如财务管理和人力资源在一块，因此WAR是可以灵活选择进行部署的）。五个系统之间如果需要实时交互，则完全通过企业的集成平台中的ESB进行交互。五个 应用之间可以建立企业级的共享数据库。

对于营销管理应用（EAR）来说，其内部可以进一步划分成客户关系、供应商管理、购销存管理等多个个业务组件，作为三个独立的WAR，三个业务组件 之间如果需交互通过营销管理应用EAR自己的ESB（从企业角度来说，需要建立联邦ESB）进行交互，或者通过企业的集成平台中的ESB进行交互。三者共 用一个数据库，逻辑上把表划分成三个部分，分别由三个业务组件进行管理，每个业务组件有自己的控制表，对于其它业务组件控制的表只读，如果需要访问，如前 文所述，通过其它业务组件提供的服务进行访问。

客户关系管理还可以进一步划分成市场管理、销售管理、服务管理等多个模块，为了便于管理，可以在此基础上再划分的更细一层，比如市场管理可以进一步 划分成客户分类管理、客户群管理、客户拜访管理等模块（Bundle），模块（Bundle）之间可以通过类直接调用，不同的模块之间的调用通过类总线实 现。从管理和系统升级调度考虑，模块不宜太少，也不易太多。

以上就一般企业常用到的功能按照前文所描述的模型基于J2EE架构上具体实现做了说明，可以作为组件化开发的一个参考。

**总结**

本文通过对业务组件（BC）的定义，特别是描述和和当前相关的CBM、SCA等组件概念的关系，描述了基于SOA的组件化编程在技术上的具体实现， 并对业务组件在对外接口、内部结构构成等技术要求做了说明，从而搭建一个灵活、易于扩展、易于维护的组件化开发模式，为企业搭建**集成平台**之后如何进行全新的组件化开发提供了一个参考。

（注：本文初次发表于 IBM developerWorks 中国网站（[原文链接](http://www.ibm.com/developerworks/cn/webservices/1003_xiaojg_soabc/)）。这里发出的是修订版。除文字修改，还增加了插图等）