

面向服务的模块化系统体系架构

技术领域

本发明涉及分布式服务器集群系统领域，尤其涉及一种面向服务的模块化系统体系架构。

背景技术

长久以来，服务器端的高层架构大体被区分为对立的两类：SOA（Service-oriented architecture）以及 AIO（All in one）。SOA 将一个完整的应用分割为相互独立的服务，每个服务提供一个单一标准功能（如：会话管理、交易评价、用户积分等等）。服务间通过 RPC、WebAPI 等 IPC 机制暴露功能接口，并以此相互通信，最终组合成一个完整的应用。

而 AIO 则相反，它将一个应用规约在一个独立的整体中，SOA 中的不同服务在 AIO 架构下呈现为不同的功能组件和模块。AIO 应用的所有组件通常都运行在一个地址空间（同一进程）内，所有组件的代码也常常放在同一个产品项目中一起维护。

AIO 的优势是部署简单，不需要分别部署多个服务，并为每个服务实现一套高可用集群。与此同时，由于可避免网络传输、内存拷贝等 IPC 通信所带来的大量开销，因此 AIO 架构的单点效率通常远高于 SOA。

另一方面，由于 AIO 架构中组件依赖性强，组件间经常知晓并相互依赖对方的实现细节，因此组件的可重用性及可替换性差，维护和扩展也较困难。特别是对于刚加入团队的新人来说，面对包含了大量互相深度耦合之组件和模块的“巨型项目”，常常需要花费大量努力、经历很多挫折并且犯很多错误才能真正接手。而即使对于老手来说，由于模块间各自对对方实现细节错综复杂的

依赖关系，也容易发生在修改了一个模块的功能后，莫名其妙地影响到其它看起来毫不相干功能的情况。

与此相反，SOA 模型部署和配置复杂——现实中，一个大型应用常常被拆分为数百个相互独立的服务，《程序员》期刊中的一份公开发表的论文显示，某个国内“彻底拥抱”SOA 的著名（中国排名前 5）电商网站将他们的 Web 应用拆分成了一千多个服务。可以想象，在多活数据中心的高可用环境内部署成百上千个服务器集群，并且配置他们彼此间的协作关系是多大的工作量。最近的携程（ctrip.com）网络瘫痪事件也是因为上千台服务组成的庞大 SOA 架构导致故障恢复缓慢。

除了部署复杂以外，SOA 的另一个主要缺点就是低效——从逻辑流的角度看，几乎每次来自客户端的完整请求都需要依次流经多个服务后，才能产生最终结果并返回用户端。而请求（通过消息中间件）每“流经”一个服务都需要伴随多次网络 IO 和磁盘访问，多个请求可累计产生较高的网络时延，使用户请求的响应时间变得不可确定，用户体验变差，并额外消耗大量资源。

此外，无论是每个 Service 各自连接不同的 DBMS 还是它们分别接入同一个后端分布式 DBMS 系统，实现跨服务的分布式事务支持工作都要落到应用层开发者手中。而分布式事务（XA）本身的实现复杂度恐怕就已超过大部分普通应用了，更何况还需要为分布式事务加上高可靠和高可用保证——需要在单个数据切片上使用 Paxos/Raft 或主从+Arbiter 之类的高可用、强一致性算法，同时在涉及多个数据切片的事务上使用 2PC/3PC 等算法来保证事务的原子性。因此 SOA 应用中的跨 Service 事务基本都只能退而求其次，做到最终一致性保证，即便如此，也需要增加大量的额外工作——在稍微复杂点的系统里，高可用，并能在指定时间内可靠收敛的最终一致性算法实现起来也不是那么容易。

与此同时，大部分 SOA 系统经常需要使用消息中间件来实现消息分发服务。如果对消息中间件的可用性（部分节点故障不会影响正常使用）、可靠性（即使在部分节点故障时，也确保消息不丢失、不重复、并严格有序）、功能性（如：发布/订阅模型、基于轮转的任务分发等）等方面有所要求的话，那么消息中间件本身也容易成为系统的瓶颈。

发明内容

本发明的目的是：为了解决上述问题，提供了一种面向服务的模块化系统体系架构，独立运行的服务被替换成了支持动态插拔的跨平台功能插件（IPlugin）；而插件则通过（并仅可通过）应用程序编程接口枢纽（API Nexus）来动态地暴露（注册）和隐藏（注销）自身所提供的功能接口，同时也使用 API Nexus 来消费其它插件提供服务。

为了实现上述目的，本发明的技术方案是：

一种面向服务的模块化系统体系架构，包括插件和应用程序编程接口枢纽；所述插件将服务器应用中的不同功能抽象成了一个个可插拔的业务模块（BMOD，通常每个业务模块对应一个插件），通过在所述应用程序编程接口枢纽（API Nexus）上注册应用程序编程接口分发器（API Dispatcher），所述插件中的业务模块动态地将当前能够提供的服务以应用程序编程接口（API）的形式暴露给当前系统以及其下的其它业务模块或其它插件。

进一步的，所述应用程序编程接口动态的注册和注销机制匹配所述插件接口的动态插拔能力。

进一步的，每个所述插件均可携带任意复杂度的配置信息，它们被分割为常规配置部分、高级配置部分和内部配置部分，且每个部分的内容可分别由

所述插件类别或具体实现自行定义。

进一步的，每个所述插件均可携带两个包含任意资源的虚拟文件系统(VFS)，它们分别为：用于对外服务的资源虚卷，以及对内自用的私有虚卷。

进一步的，所述应用程序编程接口枢纽包括分发器注册单元、分发器注销单元、发起应用程序编程接口调用请求单元、提交应用程序编程接口调用请求单元以及应用程序编程接口分发器匹配单元。

本发明完全继承了 SOA 架构高内聚、低耦合的优点，每个插件如独立的服务一样，有清晰的接口和边界，可容易地被替换和重用。在可维护性上， μ SOA 也与 SOA 完全一致，每个插件都可以被单独地开发和维护，开发人员只需要管好自己维护的功能插件即可。通过加入新插件以及对现有功能插件的重新组合，甚至可比 SOA 模式更容易地对现有功能进行变更和扩展。

而在性能方面，由于所有功能插件都运行在同一个进程内，因此通过 API Nexus 的相互调用不需要任何网络 IO、磁盘访问和内存拷贝，也没有任何形式的其它 IPC 开销，因此其性能和效率均可与 AIO 架构保持在相同量级。

与此同时， μ SOA 的部署与 AIO 同样简单——部署在单个节点即可使用，只需部署一个集群即可实现高可用和横向扩展。在配置方面也远比 SOA 简单，仅需要比 AIO 应用多配置一个待加载模块列表而已，并且这些配置也可通过各种配置管理产品来实现批量维护。简单的部署和配置过程不但简化了运营和维护工作，也大大方便了开发和测试环境的构建。

此外， μ SOA 也在极大程度上避免了对消息中间件的依赖，取而代之的是通过 API Nexus 的直接 API 调用；或是在需要削峰填谷的场合中，使用由内存零拷贝和无锁算法高度优化的线程间消息队列。这一方面大大增加了吞吐，避免了延迟，另一方面也避免了部署和维护一个高可用的消息分发服务集群所带来

的巨大工作量——μ SOA 集群内的节点间协作和协调通信需求已被将至最低，对消息分发的可靠性、可用性和功能性都没有太高要求。在多数情况下，使用 Gossip Protocol 等去中心化的 P2P 协议即足以满足需要，有时甚至可以完全避免这种集群内的节点间通信。

附图说明

图 1 是本发明实施例一示意图。

图 2 是本发明实施例二示意图。

具体实施方式

以下结合附图进一步说明本发明的实施例。

为了使本发明的目的、技术方案和优点更加清楚明白，下面结合功能图和流程图，对本发明做进一步详细说明。以下的示意性实施方式及其说明用于解释本发明，但并不作为对本发明的限定。

一种面向服务的模块化系统体系架构, 包括插件和应用程序编程接口枢纽; 所述插件将服务器应用中的不同功能抽象成了一个个可插拔的业务模块 (BMOD, 通常每个业务模块对应一个插件), 通过在所述应用程序编程接口枢纽 (API Nexus) 上注册应用程序编程接口分发器 (API Dispatcher), 所述插件中某个的业务模块动态地将当前能够提供的服务 (Service) 以应用程序编程接口 (API) 的形式暴露给当前系统以及其下的其它业务模块或其它插件。

插件 (IPlugin) 是一套跨平台的通用功能插件接口, 通常以动态链接库 (dll/so) 的形式提供, 整个插件对外仅需要暴露一个形如 “extern “C” void* CreateInstance(void);” 的 API 接口。也可以静态库或源码的形式直接嵌入到

项目中，此时无需暴露任何外部接口。插件可以根据处理器、操作系统、发行版（MBCS / UNICODE）等当前环境因素自动执行最优匹配。

优选的，所述应用程序编程接口动态的注册和注销机制匹配所述插件接口的动态插拔能力。

优选的，所述每个插件均可携带和接受任意复杂的树形配置信息，配置信息被区隔为常规配置、高级配置和内部配置等多个隔离的部分，每个部分的内容可分别由插件类别或具体实现自行定义（例如：所有数据库连接器类型的插件均可接受相同的常规配置信息，其中包括数据库服务器地址、登陆账号等通用配置；不同的数据库连接器插件可分别定义各自的高级配置信息，如 SQLite 连接器插件的高级配置信息中，可包含是否启用加密，以及加密算法等；而 MS SQL Server 连接器的高级配置信息中，则可包含连接超时，ODBC 驱动名等配置）。

优选的，每个所述插件携带两个包含任意资源的虚拟文件系统（VFS）分别是：用于对外服务的资源虚卷（通常包含页面、图片、语言包等资源）；以及用于对内自用的私有虚卷（例如：包含报表格式模板、数据字段映射表等内部资源）。。

优选的，IPlugin 定义了一套完整、自描述、灵活、可管理的接口规范。以此为基础，本发明定义了数据库连接器（DBC）插件类型。DBC 提供了如下功能：

作为数据中间件，DBC 比各个数据库产品的客户端 SDK 更易用，无需编写任何 SQL 或 NoSQL 语句，可直接使用与数据库产品无关的树形配置项来定义表、索引和数据分片规则等。

支持基于修订版（Revision）字段的 CAS（Compare and Swap）式原子更新操作。此算法可以较好地解决多个节点争抢更新同一记录的问题。

提供对用户透明的数据加密服务，为底层数据库添加数据传输和存储时的

可靠强加密保护。在底层产品或服务不支持存储及传输强加密的场合，将由中间件来透明地完成相应的强加密保护。

提供对用户透明的数据压缩服务，为底层数据库添加数据传输和存储时的实时数据压缩支持。用户可以分别为数据库中的每张表或每个集合单独配置压缩选项。透明压缩服务可与数据加密等其它服务同时启用。

使用一种通用的数据查询对象来描述复杂查询，用户无需编写任何查询语句，也不用考虑底层数据库产品兼容性。通过图形化的界面或使用数据查询对象可编程接口即可构造数据库产品无关的复杂查询表达式。

智能地使用一种我方自主实现的数据查询引擎来提供诸如：支持统一字符集编码（UNICODE）和高级正则表达式（ARE）规则的正则匹配、支持表中套表的关联查询等各类高级查询功能，并支持虚拟字段和用户自定义查询操作（比如：用户可以对部门和地理位置定义“属于”查询操作等等）。与此同时仍尽可能地使用底层数据库产品提供的索引等高效查询能力来提高效率。

高可移植性，已经实现的 DBC 插件覆盖了 SQLite、MySQL、Microsoft SQL Server、ORACLE、DB2、PostgreSQL，以及 MongoDB、Clutrix 等常用的数据库产品，用户也可容易地自行扩展符合 DBC 接口规范的新插件。由于 DBC 对外暴露的配置、查询、更新、插入、以及事务等接口均与具体产品无关，因此简单地更换 DBC 插件即可方便地不同数据库产品间进行切换。这极大地降低了应用对某个具体数据库产品的依赖性。

优选的，所述应用程序编程接口枢纽包括分发器注册单元、分发器注销单元、发起应用程序编程接口调用请求单元、提交应用程序编程接口调用请求单元以及应用程序编程接口分发器匹配单元。

分发器注册单元（Register）：将指定的 API 分发器（API Dispatcher）注

册到当前 API Nexus 中。注册时需要指定该分发器可处理的 API 种类（亦作：API 类别）。API 种类可以通过前缀、后缀、专门的描述字段或其它规则匹配。在任意给定时间里，只能为每个确定类别的 API 指定唯一的一个 API 分发器。API Nexus 会将所有该类别的 API 请求转交给此分发器来处理。

分发器注销单元（Unregister）：将指定的 API 分发器从当前 API Nexus 中注销。注销后，该 API 分发器将与当前 API Nexus 实例完全分离。该分发器所占有的 API 类别可以被其它 API 分发器重新注册和占有。

发起应用程序编程接口调用请求单元（MakeCall）：API Nexus 根据请求中提供的信息，自动将该 API 请求转交到匹配的 API 分发器来处理，并返回处理结果。若当前 API Nexus 实例中，不存在与指定请求匹配的 API 分发器，则返回错误信息。

提交应用程序编程接口调用请求单元（PostCall）：异步地执行请求，若当前 API Nexus 实例中，已包含与请求匹配的 API 分发器，则本请求被立刻执行。否则将此请求暂存在一个专门的待决请求队列中，等待对应 API 类别被成功注册后立刻执行。

应用程序编程接口分发器匹配单元（MatchDispatcher）：尝试匹配当前 API Nexus 中，是否存在满足指定条件的 API 分发器，并将结果返回给调用者。

实施例一

一种基于 μ SOA 的分布式 ERP 服务器系统案例

图 1 简单阐明了一种基于 μ SOA 的分布式企业管理平台（ERP）服务器集群系统。系统分为两大部分：基础平台和可插拔业务模块。

基础平台向下封装所有与操作系统、网络、数据库访问，以及其它与后端通用服务相关的底层操作；向上则作为各个业务模块的运行支撑环境，并为这

些业务模块提供插件管理 (IPlugin)、API 管理 (API Nexus)、用户管理、配置管理、数据库管理、网络伺服等基础服务。

而业务模块则可动态地在基础平台上插拔和替换。从平台的角度讲, 每个可插拔业务模块都是一个运行在 ERP 服务基础平台上的插件。业务模块依托于《白杨应用支撑平台》和 ERP 服务基础平台提供的运行时环境, 消费基础平台提供的各项服务, 并服从基础平台的管理和调度命令。在必要时, 多个业务模块间也可通过 API Nexus 提供的调用分发机制来实现相互通信。

从用户的角度讲, 诸如财务报销、绩效管理、薪资管理、生产管理等所有实际业务功能其实都是由各个业务模块实现的。一个完整的分布式 ERP 服务器集群 App 节点就是由 ERP 服务基础平台加上各个业务模块组合而成的。

从这个角度看, 基础平台好似一套空房间, 而业务模块则是各种可以摆放在房间中的家居物件。房间提供了水电煤、电信等基础设施, 与其中部署的家具和设备一起为入驻人员提供针对特定用途优化的运行时支持环境。

实施例二

一种基于 μ SOA 的分布式网游服务器系统案例

图 2 展示的结构与图 1 类似, 系统仍然被分为了基础平台和业务模块插件两大部分, 只不过 μ SOA 实现的具体业务逻辑由企业管理变为网络游戏。可见 μ SOA 系统对各类不同的具体业务均有很好的适应能力。具体过程与实施例一中一致, 不再赘述。

综上所述, 本发明将服务器应用中的不同功能独立抽象成了一个个不同的可插拔的业务模块, 起到了将不同功能各自分割和封装为子服务的隔离效果, 且允许不同业务模块间高效、动态地相互暴露接口给对方使用。