学校代号：10255

学　　号：2131617

基于 ESB 的服务发现和管理的研究与实现

RESEARCH AND IMPLEMENTATION OF THE SERVICE DISCOVERY AND MANAGEMENT BASED ON ENTERPRISE SERVICE BUS

专　　业 ： 软件工程

姓　　名 ： 梁思彬

校内导师 ： 燕彩蓉

校外导师 ： 王占宏

答辩日期 ： 2015 年 5 月 20 日

东华大学 计算机科学与技术学院

School of Computer Science and Technology

Donghua University

基于 ESB 的服务发现和管理的研究与实现

摘 要

ESB(Enterprise Service Bus)，即企业服务总线。它是传统中间件技术与 XML、Web 服务等技术结合的产物。ESB 提供了网络中最基本的连接中枢，是构筑企业神经系统的必要元素。ESB 的出现改变了传统的软件架构，可以提供比传统中间件产品更为廉价的解决方案，同时它还可以消除不同应用之间的技术差异，让不同的应用服务器协调运作，实现了不同服务之间的通信与整合。从功能上看，ESB 提供了事件驱动和文档导向的处理模式，以及分布式的运行管理机制，它支持基于内容的路由和过滤，具备了复杂数据的传输能力，并可以提供一系列的标准接口。

本文重点论述使用了 JMX(Java Management Extensions，即 Java 管理扩展)技术实现的基于 Mule ESB 的 Web 管理控制台的设计和实现方案，为高效、便捷地实现企业级应用集成提供了可行方案。主要工作包括：

(1) 介绍 SOA 面向服务架构、企业应用集成架构中间件技术、Web Service 技术及相关协议、ESB 企业级服务总线等相关概念。

(2) 重点分析并研究 Mule ESB 的架构和技术特性，Mule ESB 的设计思想和功能架构。从功能、非功能以及可行性方向上对基于 ESB 的服务发现和管理进行了详细的、系统的分析。

(3) 针对开源版 Mule ESB 不提供管理控制台管理服务的弊端，设计了基于 Mule ESB 的服务发现和管理系统，包括服务发现、服务管理、服务注册、服务监控以及服务使用统计模块。

(4) 基于(3)的设计实现了 Web 版的管理系统，并验证了系统架构的正确性、可行性和实用性。在总结和展望中，总结了本文的主要内容，并对 ESB 技术的发展以及下一步的工作做了展望。

关键词：企业服务总线，面向服务架构，Java 管理扩展，Mule

RESEARCH AND IMPLEMENTATION OF THE SERVICE DISCOVERY AND MANAGEMENT BASED ON ENTERPRISE SERVICE BUS

ABSTRACT

Enterprise Service Bus is a combination of traditional middleware technology and XML technology, web service and etc. ESB provides a basic network connectivity hub and it is necessary to build enterprise nervous system. ESB has changed the traditional software architecture and it can provide a cheaper solution than the traditional middleware products. At the same time ESB can also eliminate the technical differences between different applications and it can enable different application server provides different services coordination, communication and integration. From the functional point of view, ESB provides an event driven and document orientated processing mode, the distributed operation and management mechanism, which supports content-based routing and filtering, and with a transmission capacity of complex data, and it can provide series of standard interfaces.

This paper focus on the use of Java Management Extensions technology to Mule ESB design and implementation of web based management system, and it provides a feasible solution for the efficient, convenient realization of enterprise application integration. The main work includes:

Firstly, introduce the service oriented architecture, enterprise application integration architecture of middleware technology, Web Service technology and related protocols, enterprise service bus and other related concepts.

Secondly, focus on the analysis and research of ESB architecture and technical characteristics, design ideas and functional architecture of Mule ESB, and put forward the basic mode of ESB universal and complex patterns as well as the general system architecture.

Thirdly, for the open source version of Mule ESB does not provide the drawbacks of management console to management service, and designed a service discovery and management system which is based on the Mule ESB, the system including service discovery, service management, service registration, service monitoring, service statistics module.

Lastly, based on part three the design and implementation of the management system web version and has verified the system correctness, feasibility and practicability. In the summary and

outlook, summarizes the main content of this paper, and make a prospect for the development of ESB technology and the future work.

Liang Sibin(Software Engineering)

Supervised by Yan Cairong/Wang Zhanhong

KEY WORDS: enterprise service bus, service oriented architecture, jmx, mule

目 录

第一章　绪论

1.1 研究背景

随着网络传输能力的日益增进，软件体系结构以及发布形态也发生了巨大的变化。从 SOA 软件体系结构的提出，到企业级服务总线 ESB 的广泛应用，到云计算概念和实现的兴起，都说明了这样一个发展趋势。软件将向着越来越智能化，个性化，可移植化，可配制化的方向不断的发展。

随着企业之间和企业内部各种业务的发展，原有的应用系统越来越不能满足当前业务的需求，于是各个公司开发各种新系统来满足日益发展的各类业务的需要， 使应用系统的数目越来越多。但是由于先前的系统是由不同公司，在不同的平台下，甚至由不同的数据库，不同的语言实现，因此产生包括一些编程语言差异、平台差异、命名冲突、通信协议差异和数据差异所带来的异构性问题不可避免，因此对于企业之间的系统整合，数据和资源的共享，实现企业内部和企业之间的应用集成，已经逐渐成为企业信息化的一个重要命题。

目前，面向服务的架构(Service-oriented Architecture, SOA)逐渐成为企业应用集成的发展主导方向[2]。SOA 是一种定义集成基于服务的软件应用的方法，通过服务调用实现组合实现业务流程。服务是离散的功能单元，封装了可重用的业务逻辑，位置透明、平台独立且可互操作，它具有基于松散耦合、行业标准、业务敏捷性和与协议无关性等优势。典型的 SOA 架构的基本要求：服务间保持松散耦合，基于开放的标准，服务的接口描述与具体实现无关；灵活的架构-服务的实现细节，服务的位置乃至服务请求的底层协议都应该透明；SOA 在相对较粗的粒度上对应用服务或业务模块进行封装与重用等等。

而要实现 SOA 架构的一个核心问题就是如何把所有的功能、应用、数据或者服务通过一种有效的方式连接起来，而服务的提供者和服务的请求者之间仍然需要这种显式的点到点的调用，那么这就不是一个典型的 SOA 架构。因此，在 SOA 中，我们还需要这样一个中间层 ，能够帮助实现在 SOA 架构中不同服务之间的智能化管理。最早出现的集成模式是这样一个 Hub-spoke 的结构[3]，在 SOA 架构中的各服务之间设置一个类似于 Hub 的中间件，这个中间件充当整个 SOA 架构的中央管理器的作用。

现在服务的提供者和请求者之间有了一个智能的中转站，服务的请求者不再

需要了解服务提供者的具体细节。这看上去是一个好的 SOA 结构。事实上，传统的 EAI，就是通过这样一种中转站的方式来试图解决企业内部的应用整合问题。传统的 EAI，往往使用如 COM 和 CORBA 等的消息中间件进行跨平台，分布式的程序交互，修改企业资源规划以达到新的目标，使用 XML、中间件等方法来进行数据分配。因此，实际上传统的 EAI 是部件级的重用。

很不幸的是，由于基于部件的架构没有统一的标准，所以，各个厂商都有各自不同的 EAI 解决方案，如果碰到了异构的系统环境，就必须分别考虑怎样在各个不同的中间件之间周旋，来实现合理的互联方式，在具体实现过程中不得不考虑各种复杂的可能性。由于上述的多种原因，大多数传统 EAI 解决方案都比较笨重，而且不利于规则的修改和重新整合。所以，这也不是理想的 SOA 架构。这时我们就引进了 ESB，于是提出了服务总线的概念[4]。企业服务总线(Enterprises Services Bus，ESB)正是这种趋势中实现 SOA 的集成工具之一。

企业服务总线(ESB)是面向服务构架 SOA 的基础设施。目的是集成基于不同操作系统、不同硬件、不同数据库、不同传输协议实现的系统等异构平台之间的应用，为 SOA 架构模型提供服务的协作、交互通信和服务的组合流程等等的基于网络的底层分布式总线功能。它比单一 Hub 的形式更开放，总线结构有无限扩展的可能，它真正体现了 SOA 的理念，所有一切皆为服务，服务在总线(BUS)中处于平等的地位。即使我们需要一些 Hub，那么它们也是以某种服务的形式部署在总线上，相比上面的结构要灵活的多，它可以作用于面向服务的架构，分布式的应用由可重用的服务组成，面向消息的架构，应用之间通过 ESB 发送和接受消息，事件驱动的架构，应用之间异步产生和接收消息等等不同功能。ESB 就是在 SOA 架构中实现服务间智能化管理与集成的中介。

ESB 是逻辑上与 SOA 所遵循的基本原则保持一致的服务集成基础架构，它提供了服务管理的方法和在分布式异构环境中进行服务交互的功能。可以这样说，ESB 是特定环境下(SOA 架构中)实施 EAI 的方式：首先，在 ESB 系统中，被集成的对象被明确定义为服务[5]，而不是传统 EAI 中各种各样的中间件平台，这样就极大简化了在集成异构性上的考虑，因为不管有怎样的应用底层实现，只要是 SOA 架构中的服务，它是基于标准的是一定的。其次，ESB 明确强调消息(Message)处理在集成过程中的作用，这里的消息指的是应用环境中被集成对象之间的相互沟通。以往传统的 EAI 实施中碰到的最大的问题就是被集成者都有各自的消息格式，即自己的描述。

ESB 系统由于集成对象统一到服务，消息在应用服务之间传递时格式是标准的，使得直接面向消息的处理方式成为可能。如果 ESB 能够在底层支持现有的各种通讯协议，那么对消息的处理就完全不考虑底层的传输细节，而直接通过消息的标准格式定义来进行。这样，在 ESB 中，对消息的处理就会成为 ESB 的

核心，因为通过消息处理来集成服务是最简单可行的方式。这也是 ESB 中总线功能的体现。最后，事件驱动成为 ESB 的重要特征[6]。考虑到有些应用服务是长时间运行的，因此，ESB 必须也支持异步服务之间的消息交互，ESB 很容易通过事件驱动机制成为 SOA 的基础架构。

1.2 国内外发展现状

本质上来一说，ESB 就是一种可以提供有保证的、可靠的消息技术的新型中间件技术。ESB 天生就具备的能力就是解决异构环境的连接，因为集成技术一开始就是面向异构环境的。不同的数据、消息遵循不同的协议，采用不同的格式，为了完成它们之间的交互，ESB 就必须具有转换的能力。同时作为 EAI 在 SOA 下的一种形态，ESB 表现的更具开放性，尤其是对 web 服务的支持[7]。

ESB 架构设计时主要考虑数据接入和转换、通讯协议接入和转换、数据处理流程以及服务的注册和管理等方面的内容。

其中数据接入和转换：对各种被集成的应用系统提供的数据格式的支持和转换能力，例如 SOAP、XML、自定义格式以及符合某些行业标准的专有格式(WSDL 等)；数据处理流程：路由、格式转换、数据库读写等对数据的各种处理；通讯协议接入和转换：对各种被集成的应用系统的通讯协议的支持和转换能力，例如 JMS、Socket、HTTP、FTP 等；统一服务注册存储管理[8]：对服务的注册、发布、查询，以及对运营时服务的管控，并且提供服务运营状态的统计分析数据。

目前很多公司有自己的 ESB 产品，IBM 的 Web Sphere 系列是最具代表性的。这些产品的特点是拥有非常强大的功能，并支持各种 Web 服务协议，可以解决数据转换，智能路由等许多集成上的难题。但是，他们架构比较笨重，配置复杂，价格也很高，比较适合大型企业。而中小企业往往并不需要复杂的功能，也无力承担高额的成本。他们需要易配置易扩展的轻量级 ESB 架构。

针对于重量级 ESB 不适应中小企业的现况。一些面向中小企业的轻量级开源 ESB 纷纷出现，其中做得比较好的主要有 ServiceMix 和 Mule 两个项目。Mule ESB 是一个基于 ESB 架构理念的消息平台。Mule 的核心是一个基于 SEDA (Staged Even-Driven Architecture，分阶段事件驱动架构)的服务容器，该容器管理被称为通用消息对象(Universal Message Objects，UMO)的服务对象，而这些对象都是 Pojo (Plain old Java objects，简单的 Java 对象)[9]。ServiceMix 是基于 JBI (Java Business Integration，Java 业务集成)的 ESB。它是开源的基于 JBI API 和语意的 ESB 和 SOA 工具包，以 Apache 许可证方式发布，在服务器端或客户端运行：可以作为独立的 ESB 提供者，也可以作为 ESB 的服务组件。

1.3 研究内容

根据研究对比市场主流开源 ESB 系统的服务注册和系统架构，得出以下结论：主流开源 ESB 系统均没有提供一个可视化的界面对服务进行注册、管理和监控。Mule ESB 是企业服务总线的一个最优秀开源实现之一，然而令人遗憾的是，它也没有提供免费的服务管理控制平台。企业一旦在生产环境中部署了 Mule ESB，那么对于 Mule ESB 和相关服务的管理和监控就变得尤为重要。随着越来越多的应用、服务部署在 Mule ESB 上[11]，系统的复杂度会不断提高，因此企业对于监控系统性能和控制系统的需求也随之而来。

基于 Mule ESB 系统软件的研究和分析，本文研究的重点主要放在服务发现和管理平台实现上，实现对服务的注册、删除、查询和动态管理。经过研究发现 Mule ESB 虽然没有提供免费的管理控制平台软件，但是它内嵌了对 JMX 的支持，因此利用 JMX API 可以方便有效地对 Mule ESB 进行远程监控和管理。

1.4 课题研究难点

主要难点如下：

(1) 如何封装通过 JMX 获取的数据，以方便前台使用？

需要提供 JSON 数据格式的支持，并且提供 JSONP 数据访问。

(2) 如何用更直观的方式展示 Mule ESB 的状态变化？

方便用户及时了解 Mule ESB 运行状态，并进行管理。

(3) 如何用更直观的方式展示 Mule ESB 中注册的服务，方便用户查看调用？

怎样组织将获取到的数据直观地展示在客户端方便用户使用。

(4) 如何体验客户端的体验效果？

一个软件，除了稳定，功能强大，用户体验也很重要。程序开发人员和测试人员在强调其功能和性能的同时，往往忽视了用户体验的重要性，或者说只关注用户体验的界面，易用等方面，而忽视了其他方面。

1.5 课题的创新之处

解决了开源版 Mule ESB 不提供管理控制台管理服务的问题，设计了基于 Mule ESB 的服务发现和管理系统，包括服务发现、服务管理、服务注册、服务监控以及服务使用统计模块等。

1.6 论文组织结构

本文的论文结构如下所示：

第一章介绍了课题的研究背景，描述了当前 ESB 的发展现状，提出了课题的研究内容及难点，阐述了课题的创新之处。

第二章就论文所涉及到的领域概念、技术进行分别介绍，主要包括了 SOA 面向服务架构，WebService 技术的现状，企业服务总线 Mule ESB，最后是 JMX(Java Management Extensions，即 Java 管理扩展)。

第三章详细描述了管理控制台的分析。分析了 Mule ESB 的架构，并简要介绍目前 Mule 已经支持的第三方产品以及一些技术。还详细阐述了非功能需求分析、功能性需求分析和可行性分析。

第四章详细描述了管理控制台的模型设计和功能模块的设计。重点分析了 JMX 代理的设计。

第五章详细的展示了管理控制台的实现情况。分别对应第四章的设计提出了相应的实现方法，展示了实验结果，并对结果进行了分析。

第六章对课题研究做了总结、展望，并提出了改进思想。

第二章　关键技术和规范

2.1 面向服务架构 SOA

SOA 是一种体系架构方法，用于定义、链接和集成具有清晰边界且功能方面自包含的可重用业务服务。在这种类型的体系架构中，您可以对业务流程中的业务服务进行协调。通过采用面向服务的概念（一个独立于应用程序或基础设施 IT 平台以及上下文和其他服务的较高级别的抽象），SOA 将信息技术提升到了一个新的级别，因此更为适合互操作性和异类环境[13][14]。

因为 SOA 构建于主要信息服务提供商认可和支持的标准（如 Web 服务标准等）之上，所以可以快速构建服务和进行互连。可以在不考虑所支持的基础设施的情况下在企业间进行互连，从而为委托、共享、重用现有资产并实现其好处的最大化打开方便之门。通过建立 SOA，可以将内部信息服务基础设施提高到一个更高、可见性更好且可管理的级别[15]。通过可重用服务和高级流程，能以比以往任何时候都方便的方式进行更改，而且更像是分解部件（服务）并将其重新组合为新的与业务一致的流程。这不仅提高了效率和重用，而且还提供了极强的更改和保持信息服务与业务一致的能力。

基本 SOA 体系架构包括服务提供者、可选的服务目录和服务，如图 2-1 所示。应用程序到应用程序的消息传递在信息交换中使用。这个模型和简单 Web 服务之间最为明显的相似点是：二者都将 WSDL 作为存储在服务目录中的调用契约。Web 服务实际上是最基本的 SOA 实现[16]。

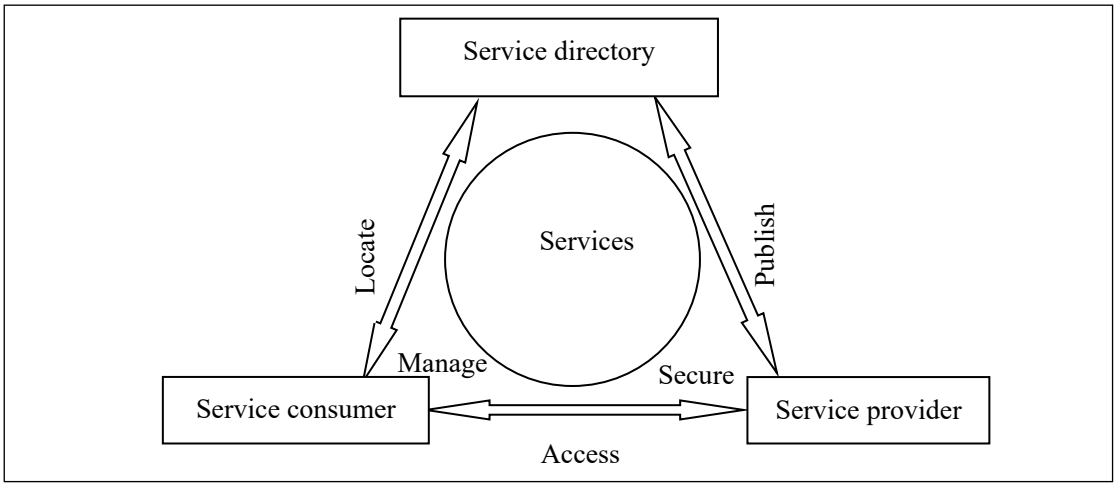


图 2-1 基本 SOA 体系架构

在此业务模型中，基本场景如下：首先，服务提供者创建服务，并决定将其公开并发布。发布是通过将服务信息发布到服务目录上来完成的。另一方面，需要特定服务的服务请求者将在服务目录中搜索满足必要条件的服务。找到服务后，通过使用服务目录中的可用信息，服务请求者能够以正确的方式直接联系服务提供者，从而满足业务需求。

企业服务总线在 SOA 中扮演着重要的角色。从基础的角度而言，它代表着能够连接服务提供者和服务使用者的中枢和基础架构。以下是 ESB 的角色的详细介绍[17]：

(1) 提供管理服务基础设施的方法。

(2) 集中控制和分散处理。

(3) 提供与 SOA 原则一致的集成基础设施：使用强调位置透明性和互操作性的通信协议；强制使用独立于实现的显式接口来定义采用松散耦合的服务；促进采用封装可重用业务功能的方式定义服务。

(4) 支持中介，根据需要在不同各方之间形成请求响应，而且不用更改其中任何一方。

(5) 将安全性和 QoS 应用到 SOA 项目。

2.2 企业应用集成 EAI 和中间件技术

在 20 世纪 60 年代到 70 年代期间，企业应用大多是用来替代重复性劳动的一些简单设计。当时并没有考虑到企业数据的集成，惟一的目标就是用计算机代替一些体力性质的，孤立的工作环节。到了 20 世纪 80 年代，有些公司开始意识到应用集成的必要性和价值。这是一种挑战，很多公司的技术人员都试图在企业系统整体概念的指导下对已经存在的应用进行重新设计升级，以便让它们集成在一起 。然而这种努力收效甚微。20 世纪 90 年代，ERP 应用开始流行的时候，同时也要求它们能够支持已经存在的应用和数据，这就必须引入 EAI (Enterprise Application Integration)[18]。EAI 系统采用类似于代理和集线中心的方式，进行应用集成；其后类似总线的 EAI 体系结构通过中心管道的方式，通过在各节点安放软件适配器和集成引擎。实现分布式智能，进行点到点的、自动的通信，但扩充性差，复用性差。

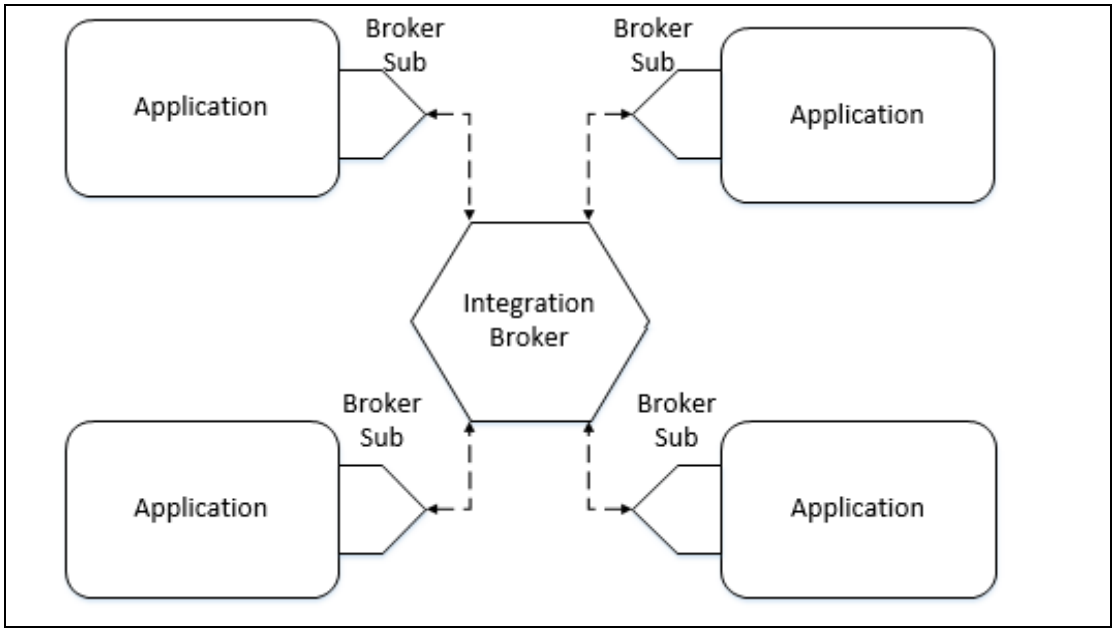


图 2-2 EAI 系统集线中心和代理的方式模型

中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。中间件位于操作系统之上网络通讯和管理计算机资源，也可以是连接两个独立应用程序或独立系统的软件。多个相连接的系统，即使它们具有不同的接口，但通过中间件相互之间仍能交换信息[19]。执行中间件的一个关键途径是信息传递。通过中间件，应用程序可以工作于多平台或不同操作系统的环境。图 2-2 中的 Broker Sub 就是相对应用系统的中间件接口，而 ESB 也是一种可以提供有保证的、可靠的消息技术的新型中间件技术。可以说没有中间件技术就没有 SOA 架构的落地实施。

2.3 Web Service 技术

Web Service 可以翻译为 Web 服务，是使应用程序可以基于平台以及编程语言无关的方式进行相互通信的一项技术。Web Service 服务是一个软件接口，它描述了一组可以在网络上通过标准化的 XML 消息传递访问的操作。它使用基于 XML 语言的协议标准来描述要执行的操作或者要与另一个 Web 服务交换的数据。图 2-3 为 Web Service 模型图。

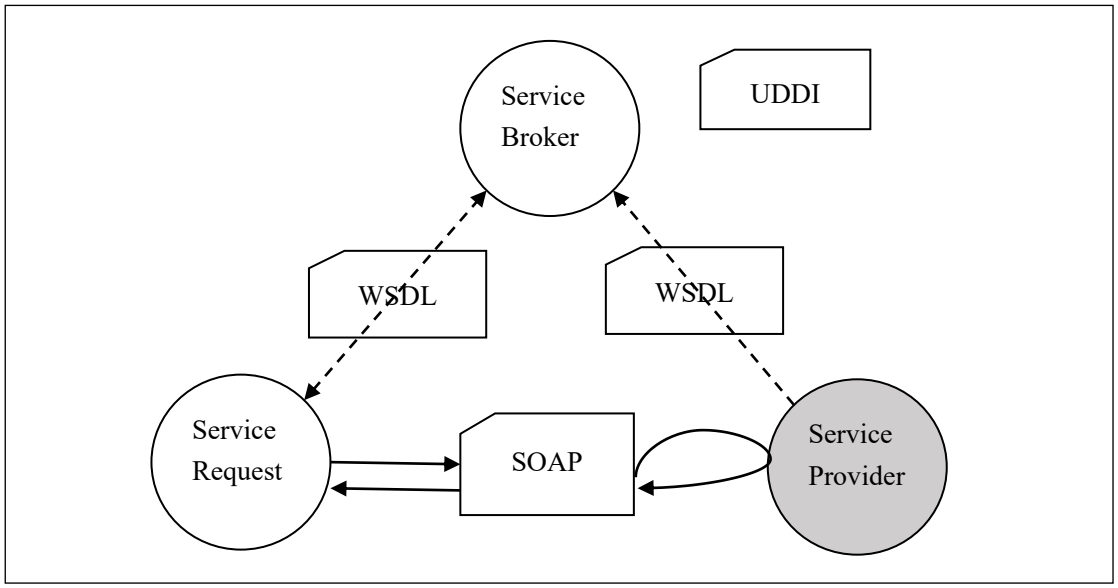


图 2-3 Web Service 模型

Web 服务是一种面向服务 SOA 架构的技术，通过标准的 Web 协议提供服务，目的是保证不同平台的应用服务可以相互操作。根据 W3C 的定义，Web 服务应当是一个软件系统，用以支持网络间不同机器的相互操作[20]。网络服务通常是许多应用程序接口(API)所组成的，它们通过网络，例如国际互联网(Internet)的远程服务器端，执行不同客户所提交服务的请求。

尽管 W3C 的定义涵盖诸多相异而且无法区分的系统，不过通常我们指有关于主从式架构(Client-Server)之间根据 SOAP 协议进行传递 XML 格式消息。无论定义还是实现，WEB 服务过程中会由服务器提供一个机器可读的描述(通常基于 WSDL)以辨识服务器所提供的各种 WEB 服务。另外，虽然 WSDL 不是 SOAP 服务端点的必要条件，但目前基于 Java 的主流 WEB 服务开发框架往往需要 WSDL 实现客户端的源代码生成。一些工业标准化组织，比如 WS-l，它就在 WEB 服务定义中强制包含 SOAP 和 WSDL。

Web Service 是一种新的 web 应用程序分支，他们是自描述、自包含、模块化的应用，可以发布、定位、通过 web 调用。Web Service 可以执行从简单的请求到复杂商务处理的任何功能。一旦部署以后，其他 Web Service 应用程序可以发现并调用它部署的服务[21]。ESB 的一个主要应用对象就是服务本身是独立 Web Service 应用，从狭义上讲 ESB 的连接应用都应该是基于 HTTP 协议上的以 SOAP 协议传输数据的 Web 服务，所以二者之间的联系十分的紧密，而可扩展标记语言 XML 和 SOAP 协议也同时是 ESB 所必须包含的重要技术内容[22][23]。

UDDI 是 Universal、Description、Discovery 和 Integration 的缩写，是一种创建注册表服务的规范，如图 2-4 所示，以便对 Web Service 进行注册、发布供使用者查找。当服务提供者想将自己的 Web Service 向全世界公布，服务提供者为了便于外部找到其服务，可以把自己的 Web Service 注册到 UDDI 商用注册网站。

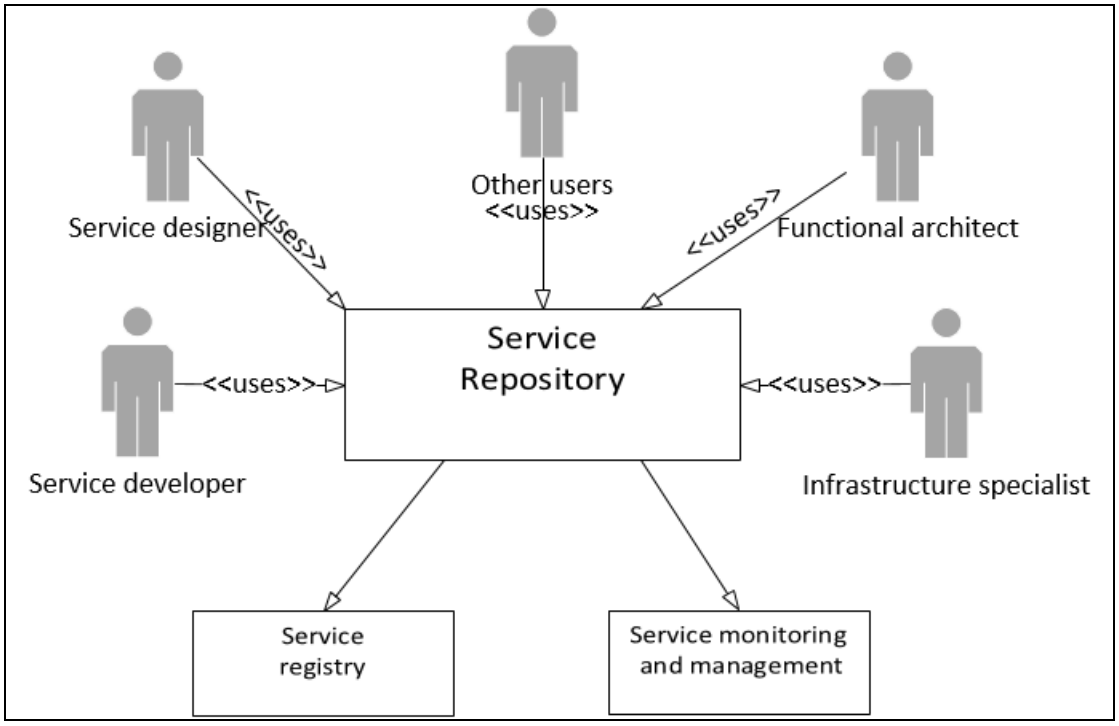


图 2-4 UDDI 注册模型

世界上曾经有微软、IBM、SAP 和 HP 四家著名的 UDDI 注册中心，不过到 2006 年 12 月底，它们都陆续关闭了。这是因为随着 UDDI 技术的发展，现在一般用户已经可以轻松的架设自己的 UDDI 中心，而不必再去其它网站注册。另外需要指出的是，UDDI 对于 Web Service 和 SOA 来说并不是必须的[24]。WSDL 文件中已经给出了服务的地址，外部用户只要能获得 WSDL 文件，就能利用里面的地址对服务进行调用。而本文的研究方向也是对 UDDI 注册方式的一种有力的改进和自我实现的过程。由 ESB 系统自实现更加便捷的 Web 服务注册和管理中心。

2.4 可扩展标记语言 XML

可扩展置标语言 XML(Extensible Makeup Language)，是一种置标语言，又称可扩展标记语言。置标这个概念是指计算机所能理解的信息符号，通过此种标记，计算机之间可以处理包含各种信息的文章等。如何定义这些标记，既可以选择国际通用的标记语言，比如 HTML，也可以使用像 XML 这样由相关人士自由决定的标记语言，这就是语言的可扩展性[25]。XML 是从标准通用置标语言(SGML)中简化修改出来的。它主要用到的有可扩展样式语言 XSL、可扩展置标语言、XBRL 和 XPath 等。

XML 设计用来携带和传送数据信息，不是用来表现或展示数据，超文本语言(HTML)则用来表现数据，所以 XML 用途的焦点是它说明数据是什么，以及

所携带的数据信息。丰富文件(Rich Documents)-自定文件描述并使其更丰富。属于文件为主的 XML 技术应用。标记是用来定义一份资料应该如何呈现。元数据(Metadata)-描述其它文件或网络资讯。属于资料为主的 XML 技术应用。标记是用来说明一份资料的意义。

设置档案(Configuration Files)-描述软件设置的参数。由于 XML 标记语言有诸多的优势和能力，SOA 架构的许多配置文档，数据文档都是 XML 格式的，Web Service 的 SOAP 协议是一个基于 XML 的可扩展消息信封格式，需同时绑定一个传输用协议。这个协议通常是 HTTPS 或者 HTTP，但也可能是 XMPP 或 SMTP。

WEB 服务描述语言 WSDL(Web Services Description Language)也是一个 XML 格式文档，用以描述服务使用协议和端口访问方式的细节。通常用来辅助生成服务器和客户端代码及配置信息。而 ESB 的优势之一就是能够将以前复杂变化的代码用配置文件来灵活的调配，切断了服务提供者和服务使用者之间静态的逻辑关系，而配置文件基本是以 XML 文件格式为主。

2.5 简单对象访问协议 SOAP

简单对象访问协议(SOAP，Simple Object Access Protocol)是一种标准化的通讯规范，主要用于 Web Service 中[26]。SOAP 的出现是为了简化网页服务器利用 Web Server 技术从 XML 数据库中提取数据时，无需花过多的时间去格式化页面，并能够让不同应用程序之间透过 HTTP 通讯协定，以 XML 格式互相交换彼此的携带信息，使其与编程语言、硬件和平台无关。

SOAP 是一种简单的、轻量的、基于 XML 的协议，它被设计成在 WEB 上交换结构化的和固化的信息。SOAP 包括四个部分：

(l) SOAP 封装：它定义了一个框架，该框架描述了消息中的内容是什么，什么对象应当处理它，以及它是可选的还是必须的。

(2) SOAP RPC 表示：它定义了用于表示远程过程调用和应答的协定。

(3) SOAP 编码规则：它定义了一种序列化的机制，用于交换应用程序所定义的数据类型的实例。

(4) SOAP 绑定：定义了一种使用底层传输协议来完成在节点间交换 SOAP 封装的约定。

SOAP 消息基本上是从发送端到接收端的单向传输，但它们常常结合起来执行类似于请求/应答的模式。所有的 SOAP 消息都使用 XML 文件格式编码。一条 SOAP 消息就是一个包含有一个必需的 SOAP 的封装包，一个可选的 SOAP 标头和一个必需的 SOAP 体块的 XML 文档[27]。

把 SOAP 绑定到 HTTP 提供了同时利用 SOAP 的样式和分散的灵活性的特点以及 HTTP 的丰富的特征库的优点。在 HTTP 上传送 SOA 并不是说 SOAP 会覆盖现有的 HTTP 语义，而是 HTTP 上的 SOAP 语义会自然的映射到 HTTP 语义。在使用 HTTP 作为协议绑定的场合中，RPC 请求映射到 HTTP 请求上，而 RPC 应答映射到 HTTP 应答。然而，在 RP 上使用 SOAP 并不仅限于 HTTP 协议绑定。基于 SOAP 协议的传输特性，Web Service 实现不同的系统之间能够用“软件-软件对话”的方式相互调用，打破了软件应用、各种设备和网站之间的格格不入的状态，实现了基于 Web 无缝集成服务的目标[4][9]。

2.6 企业服务总线 ESB

2.6.1 ESB 概述

各个不同的组织给予 ESB 不同的定义。一些组织认为 ESB 是一种系统架构方法，另一些组织认为 ESB 是一个产品。目前更多的专家认为 ESB 是实现 SOA 这一架构的一种产品。

IDC(Internet Data Center)将 ESB 定义为：基于开放的标准消息总线，用于通过标准的接口和适配器，来提供各程序和组件之间的互操作功能[5]。它支持相互独立的异构环境中的消息、服务及基于事件的交互，并且具有适当的服务级别和可管理性。

ESB 在本质上就是一种可以提供可靠的、有保证的消息技术的新型中间件技术。ESB 中间件产品利用的是面向消息的中间件 MOM 和 Web Service 标准，即 Message-Oriented Middleware 协议接口。ESB 是在 SOA 的基础上提出的构件产品，是基于 SOA 解决方案时企业所使用的基础架构的关键部分[6][11]。简而言之，ESB 以一组丰富的功能启用管理和监控应用程序之间的交互，提供了集成企业内部及跨企业间的新的和现有的软件应用系统的功能。通过使用 ESB，可以在几乎不更改代码的情况下，以一种无缝的非侵入方式使企业已有的系统具有全新的服务接口，并能够在部署环境中支持任何标准。

更重要的是，充当“缓冲器”的 ESB 负责在诸多服务之间转换业务逻辑和数据格式，这种功能使得服务逻辑与服务本身之间相分离，不用在应用程序或者数据发生变化时，改动服务代码，从而使得不同的应用程序可以同时使用同一服务。ESB 提供了基于标准的连接。ESB 形成了一个基于 XML 标准的信息骨架，使得在系统内部和整个价值链中可以很容易地进行异步或同步数据交换。ESB 通过使用 Web 服务、.NET、J2EE 和其他标准提供了更强大的系统连接性[23]。ESB

是灵活的、以服务为导向的应用组合。基于 SOA、ESB 应用模型使得复杂的分布式系统，包括跨多个应用、防火墙和系统的集成解决方案，能够由以前开发测试过的服务组合而成，这使系统具有高可扩展性。

ESB 产品提高了服务的复用率，减少市场反应时间，减少系统架构的成本，提高生产率。依照 SOA 方法简化应用组合，基于标准的通信、转换和连接在进行构建应用时直接提高了复用率，简化了维护，进而减少了系统的总体成本。它是传统中间件技术与 XML、Web 服务等技术结合的产物。ESB 位于整个架构的中心。ESB 提供了网络中最基本的连接中枢，是构筑企业神经系统的必要元素。ESB 可以在多个层面上对其进行描述，这取决于需要的连接的复杂程度。

业务的某些部分仅仅要求能够快速、安全、可靠地将信息从一个应用或服务转移 到另外一个应用或服务，实际上有时候是从多个应用或服务转移到多个应用或服务。

业务的其他部分想获取这样的信息：是否需要改变它们的程序来处理多种不同的但又必须处理的数据格式，而改变之后它们的程序会不会变得非常复杂。对于这些不同数据格式的表格，它们试图找出对发送格式与接收格式进行匹配的方法。

业务的其他部分正计划在企业内部移动大量的 XML，他们知道这将对处理器的负载产生一定影响：因为每台服务器在处理这些数据集的同时，还要在不同的 Web 服务之间保证安全性或者处理大量基于 XML 的数据消息，必将导致处理速度变慢。企业服务总线(ESB)帮助解决了维护冲突的问题和应用接口潜在的兼容性问题 [24]。SOA 的目的之一就是关注于服务如何支持业务，以及如何具有连接到环境中其他部分的本质能力，ESB 使得假设变为现实。

ESB 是一种可以提供可靠的、有保证的消息技术的最新方法。ESB 中间件产品利用的是 Web 服务标准和与公认的可靠消息 MOM 协议接口。例如 Tibeo 的 Rendezvous、IBM 的 Websphere MQ 和 Sonic Software 的 SonicMQ。ESB 产品的共有特性包括：连接异构的 MOM、利用 Web 服务描述语言接口封装 MOM 协议，以及在 MOM 传输层上传送简单对象应用协议(SOAP)传输流的能力。大多数 ESB 产品支持在分布式应用之间通过中间层如集成代理实现直接对等沟通。企业服务总线可以帮助你实现 SOA 的目标。ESB 位于 SOA 的中心，并通过减少接口的数量，大小和复杂度使得 SOA 更为强大。它是用来整合应用和服务的一个灵活的基础架构。

ESB 主要完成一下几件事情：

(1) 服务之间的消息路由。

(2) 请求者和服务之间的消息格式的转换。

(3) 处理各种来自不同业务的事件。

(4) 请求者与服务之间的传输协议的转换。

(5) 保证服务质量的安全可靠。

ESB 的出现改变了传统的软件架构，可以提供比传统中间件产品更为廉价的解决方案，同时它还可以消除不同应用之间的技术差异，让不同的应用服务器协调运作，实现了不同服务之间的通信与整合。从功能上看，ESB 提供了文档导向和事件驱动的处理模式，以及分布式的运行管理机制，它支持基于内容的路由和过滤，具备了复杂数据的传输能力，并可以提供一系列的标准接口。当然，没有一家 ESB 厂商满足于只提供一条可靠的通道。企业应用集成、MOM、业务流程管理、集成代理、数据转换、指挥协调、事件通知、发布与订阅、基于内容的路由、事务处理等等。它们都是 ESB 的功能模块。

2.6.2 Mule ESB

MULE 是一个以 Java 为核心的轻量级的整合平台和消息框架，基于 EIP(Enterprise Information Portal，企业信息门户)而实现的。Mule 的核心组件是 UMO(Universal Message Objects，通用消息对象)，从 Mule 2.0 开始 UMO 这一概念已经被组件 Compose 所代替)，UMO 实现整合逻辑。UMO 可以是 POJO(Plain Ordinary Java Object，简单的 JAVA 对象)，也可以是 JavaBean 等等。它支持 30 多种传输协议(TCP，EMAIL，FILE，FTP，UDP，HTTP，SOAP，JMS 等)，并整合了许多流行的开源项目，比如 CXF，Axis，Spring，ActiveMQ，Drools 等[2]。虽然 Mule 没有基于 JBI 来构建其架构，但是它为 JBI(Java Business Integration，JAVA 业务集成)容器提供了 JBI 适配器，因此可以很好地与 JBI 容器整合在一起。而 Mule 更关注其灵活性、易开发性以及高效性。从 2005 年发表 1.0 版本以来，Mule 吸引了越来越多的关注者，成为开源 ESB 中的一支独秀。目前许多公司都使用了 Mule，比如 HP，Sony，Walmart，Deutsche Bank 以及 CitiBank 等公司。Mule 3 版本以后集成 OSGI(Open Service Gateway Initiative，面向 Java 的动态模型系统)，支持动态热部署。

2.6.3 ESB 的主要应用领域

为了应对越来越复杂、繁琐的企业级信息系统平台，面向服务体系架构是能够将应用程序的不同功能单元通过服务之间定义良好的接口和契约联系起来。选用标准接口包装旧的应用程序、把新的应用程序构建成服务，那么其他应用系统就可以很方便的使用这些功能服务。只要 IT 人员选用标准接口包装旧的应用程序、把新的应用程序构建成服务，因为 SOA 使用户可以不受限制地重复使用软件，把各种资源互连起来，那么其他应用系统就可以很方便的使用这些功能服务。

支撑 SOA 的关键是其消息传递架构-企业服务总线(ESB)。ESB 是传统中间件技术与 XML、Web 服务等技术相互结合的产物，用于实现企业应用不同消息和信息的准确、安全传递和高效。让不同的应用服务协调运作，实现不同服务之间的整合与通信。ESB 在不同领域具有非常广泛的用途：

电信领域：ESB 能够在全方位支持电信行业 OSS(Operation Support System，运营支撑系统)的应用整合概念。是理想的电信级应用软件承载平台。

电力领域：ESB 能够在全方位支持电力行业 EMS 的数据整合概念，是理想的 SCADA(Supervisory Control And Data Acquisition，数据采集与监视控制系统)系统数据交换平台。

金融领域：ESB 能够在全方位支持银企间业务处理平台的流程整合概念，是理想的 B2B(Business-to-business e-commerce model，企业对企业的电子商务模式)交易支撑平台。

电子政务：ESB 能够在全方位支持电子政务应用软件业务基础平台、信息共享交换平台、决策分析支撑平台和政务门户的平台化实现。

2.7 Java 管理扩展 JMX

JMX 在 Java 编程语言中定义了应用程序以及网络管理和监控的设计模式、体系结构、应用程序接口以及服务。通常使用 JMX 来监控系统的运行状态或管理系统的某些方面，比如重新加载配置文件、清空缓存等[1]。优点可以非常容易的使应用程序具有被管理。伸缩性的架构每个 JMX Agent 服务可以很容易的放入到 Agent 中，每个 JMX 的实现都提供几个核心的 Agent 服务，你也可以自己编写服务，服务可以很容易的被部署，取消部署。主要提供接口，允许有不同的实现。

JMX 体系结构分为以下四个层次，如图 2-5 所示：

(1) 设备层(Instrumentation Level)：主要定义了信息模型。在 JMX 中，各种管理对象以管理构件的形式存在，需要管理时，向 MBean 服务器进行注册。该层还定义了通知机制以及一些辅助元数据类。

(2) 代理层(Agent Level)：主要定义了各种服务以及通信模型。该层的核心是一个 MBean 服务器，所有的管理构件都需要向它注册，才能被管理。注册在 MBean 服务器上管理构件并不直接和远程应用程序进行通信，它们通过协议适配器和连接器进行通信。而协议适配器和连接器也以管理构件的形式向 MBean 服务器注册才能提供相应的服务[1]。

(3) 分布服务层(Distributed Service Level)：主要定义了能对代理层进行操作的管理接口和构件，这样管理者就可以操作代理。然而，当前的 JMX 规范并没

有给出这一层的具体规范。

(4) 附加管理协议 API：定义的 API 主要用来支持当前已经存在的网络管理协议，如 SNMP、TMN、CIM/WBEM 等[1]。

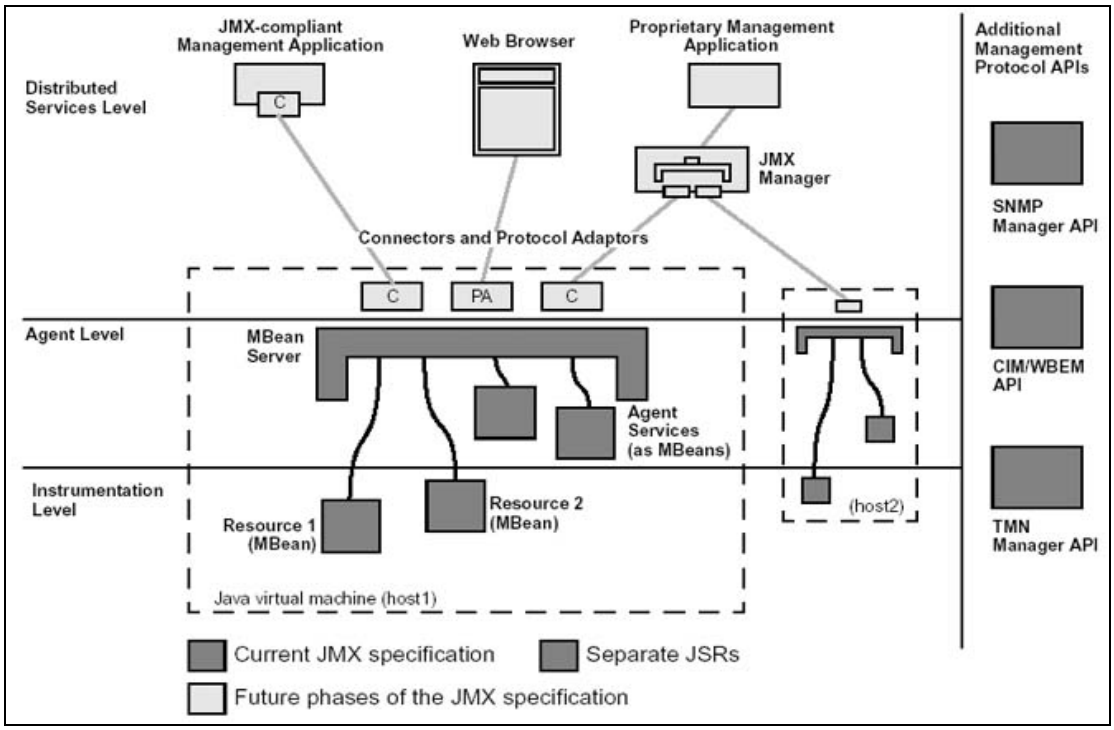


图 2-5 JMX 架构图

设备层定义了如何实现 JMX 管理资源的规范。一个 JMX 管理资源可以是一个 Java 应用、一个设备或一个服务，它们可以用 Java 开发，或者至少能用 Java 进行包装，并且能被置入 JMX 框架中，从而成为 JMX 的一个管理构件(Managed Bean)，简称 MBean。管理构件可以是标准的，也可以是动态的，标准的管理构件遵从 JavaBeans 构件的设计模式；动态的管理构件遵从特定的接口，提供了更大的灵活性。该层还定义了通知机制以及实现管理构件的辅助元数据类。[1]

在 JMX 规范中，管理构件定义如下：它是一个能代表管理资源的 Java 对象，遵循一定的设计模式，还需实现该规范定义的特定的接口。该定义保证了所有的管理构件以一种标准的方式来表示被管理资源。管理接口就是被管理资源暴露出的一些信息，通过对这些信息的修改就能控制被管理资源。一个管理构件的管理接口包括：能够执行的操作；能发出的通知事件；能被接触的属性值；管理构件的构建器。管理构件通过公共的方法以及遵从特定的设计模式封装了属性和操作，以便暴露给管理应用程序。例如，一个只读属性在管理构件中只有 Get 方法，可读写的属性既有 Get 又有 Set 方法[10]。其余的 JMX 的构件，例如 JMX 代理提供的各种服务，也是作为一个管理构件注册到代理中才能提供相应的服务。JMX 对管理构件的存储位置没有任何限制，管理构件可以存储在运行 JMX 代理的 Java 虚拟机的类路径的任何位置，也可以从网络上的任何位置导入。JMX 定义

了四种管理构件：标准、开放、动态和模型管理构件。每一种管理构件可以根据不同的环境需要进行制定。

代理层是一个运行在 Java 虚拟机上的管理实体，它活跃在管理资源和管理者之间，用来直接管理资源，并使这些资源可以被远程的管理程序所控制。代理层由一个 MBean 服务器和一系列处理被管理资源的服务所组成。Mbean 服务器为代理层的核心，设备层的所有管理构件都在其注册，管理者只有通过它才能访问管理构件。管理构件可以通过以下三种方法实例化和注册：管理代理本身；通过另一个管理构件；远程应用程序。注册一个管理构件时，必须提供一个唯一的对象名。管理应用程序用这个对象名进行标识管理构件并对其操作。这些操作包括：发现管理构件的管理接口；执行管理构件中定义的操作；读写属性值；获得管理构件发出的通告；基于对象名和属性值来查询管理构件。

MBean 服务器依赖于协议适配器和连接器来和运行该代理的 Java 虚拟机之外的管理应用程序进行通信。协议适配器通过特定的协议提供了一张注册在 MBean 服务器的管理构件的视图。例如，一个 HTML 适配器可以将所有注册过的管理构件显示在 Web 页面上。不同的协议，提供了不同的视图。为使代理和管理应用程序进行通信连接器，必须提供管理应用一方的接口，即针对不同的协议，连接器必须提供同样的远程接口来封装通信过程。当远程应用程序使用这个接口时，就可以通过网络透明的和代理进行交互，而忽略协议本身。连接器和适配器使 MBean 服务器与管理应用程序能进行通信[1][12]。因此，一个代理要被管理，它必须提供至少一个协议适配器或者连接器。面临多种管理应用时，代理可以包含各种不同的协议适配器和连接器。代理服务可以对注册的管理构件执行管理功能。通过引入智能管理，JMX 可以帮助我们建立强有力的管理解决方案。代理服务本身也是作为管理构件而存在，也可以被 MBean 服务器控制。

服务层规定了实现 JMX 应用管理平台的接口。这一层定义了能对代理层进行操作的管理接口和组件。这些组件能：

(1) 通过各种协议的映射（如 SNMP、HTML 等），提供了一个 JMX 代理和所有可管理组件的视图。

(2) 为管理应用程序提供一个接口，以便它通过一个连接器能透明和代理层或者 JMX 管理资源进行交互。

(3) 收集多个 JMX 代理端的管理信息并根据管理终端用户的需要筛选用户感兴趣的信息并形成逻辑视图送给相应的终端用户。

(4) 分布管理信息，以便构造一个分布式系统，也就是将高层管理平台的管理信息向其下众多的 JMX 代理发布。

(5) 提供了安全保证。通过管理应用层和另一管理代理和以及他的设备层的联合，就可以为我们提供一个完整的网络管理的解决方案。这个解决方案为我们

带来了独一无二的一些优点：根据需要部署、动态服务、轻便、还有安全性。

附加管理协议 API，该层提供了一些 API 来支持当前已经存在的一些管理协议。这些附加的协议 API 并没有定义管理应用的功能，或者管理平台的体系结构，他们仅仅定义了标准的 Java API 和现存的网络管理技术通信，例如 SNMP。网络管理平台和应用的开发者可以用这些 API 来和他们的管理环境进行交互，并将这个交互过程封装在一个 JMX 管理资源中[1]。例如，通过 SNMP 可以对一个运行有 SNMP 代理的交换机进行管理，并将这些管理接口封装成为一个管理构件。在动态网络管理中，可以随时更换这些管理构件以适应需求。这些 API 可以帮组开发者根据最通常的工业标准来部署他们的管理平台和应用。新的网路管理的解决方案可以和现存的基础结构合为一体，这样，现存的网络管理也能很好的利用基于 Java 技术的网络管理应用。这些 API 目前在 JCP(Java Community Process)内作为独立的 JSR(Java Specification Request)开发。

2.8 本章小结

本章详细介绍了与 ESB 系统相关的诸多理论基础和技术规范。以 SOA 面向服务架构开始，分别的介绍了 SOA 的核心规范包括的几个内容：SOAP 通信协议，WSDL 网络服务描述语言以及 UDDI 服务管理机制。进而就企业应用集成 EAI 和中间件的发展过程和优缺点展开讨论。Web Service 技术是本章中介绍比较详细的部分，并针对这种技术基于的可扩展置标语言 XML 和简单对象访问协议 SOAP 进行比较详细的展开。在了解了上述多种技术规范和理论概念之后，本章后面的部分着重的引出了企业服务总线 ESB 的相关概念。主要从如下几个方面：ESB 定义、ESB 的技术结构、ESB 的功能特性、ESB 引用的方式、主流 ESB 开源软件特性、ESB 的应用领域等。最后详细的介绍了 Mule ESB 控制管理的实现技术 JMX。JMX 体系结构分为四层，即设备层、代理层、分布服务层和附加协议 API。但 SUN 当前只实现了前两层的具体规范，其余的规范还在制定当中。JMX 代理要和远程应用程序通信，需要提供至少一个连接器和协议适配器。

第三章　基于 ESB 的服务发现和管理的分析

3.1 Mule ESB 架构分析

3.1.1 Mule ESB 架构

Mule ESB 是一个轻量级的分布式消息框架，通过不同的协议与各种服务和应用程序通信。其关键特性如下所示：插件式的连接组件支持 SMTP、POP3、JMS、TCP、UDP、JDBC、Multicast、Http、File 等传输协议；支持异步通信、同步通信和请求-应答式的事件处理方式；对标准 WebService 规范的支持：支持强大的应用集成的能力[2][7]。

其技术架构如图 3-1 所示：

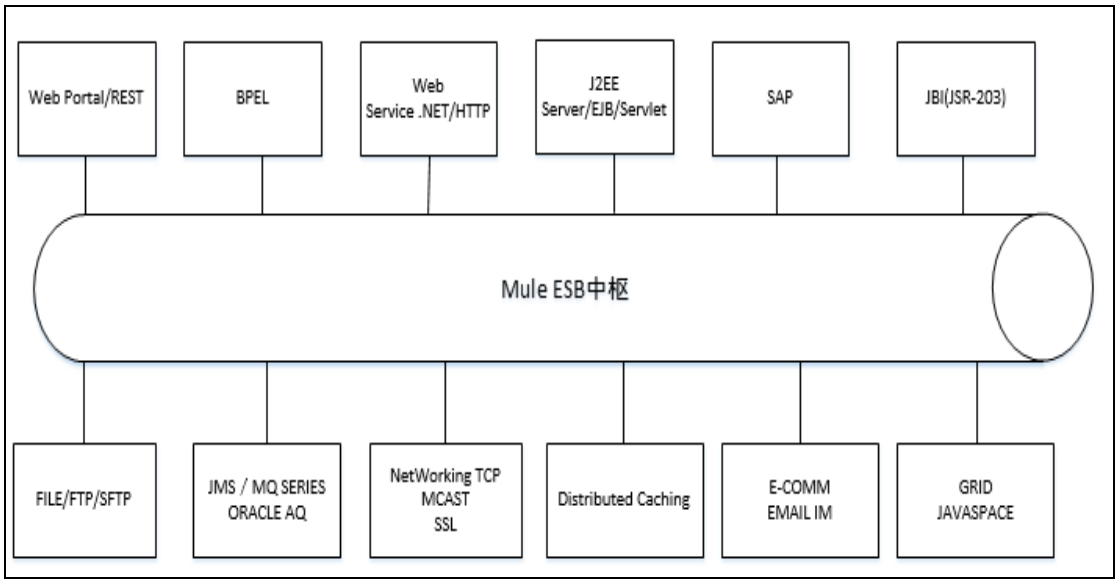


图 3-1 Mule ESB 架构

Mule ESB 是一个消息代理，一个消息 ESB 框架，一个分级事件驱动的框架(SEDA)。SEDA 定义了一个高度并行、依照分级队列的企业级平台。 Mule ESB 使用 SED 的概念增加事件处理的性能。

事实上，Mule ESB 超出了传统意义上的 ESB，它包含一个在不同系统之间分配信息的对象经纪人，我们更愿意把它定义为一个轻量级的消息框架。它的目的是管理消息组件，这些组件通常称为 UMO（Universal Message Objects，通用

消息对象)，它们可以共存在同一个 VM 中或者分散在你的网络中。UMO 和其它应用软件系统之间的通信都是通过 endpoints 来进行的[8]。

Mule ESB 支持同步、异步的请求响应事件，事件处理和传输实用不同的技术例如 HTTP、JMS，电子邮件和基于 XML 的 RPC。 Mule ESB 能很容易地嵌入到任何应用框架中，明确支持 Spring 框架。Mule ESB 也支持预定义的，动态的，基于内容的和基于规则的消息路由。Mule ESB 使得预定义的和计划性的事务更容易，包括 XA 事物支持。Mule ESB 提供一个有代表性的状态调用(REST) API 提供给与 Web 的事件访问。

Mule ESB 模式驱动系统中所有服务，这个系统有着一个分离的消息通讯中枢。服务注册在总线上，但不知道其他任何被注册的消息；因此，每个服务只关心处理它收到的事件。

Mule ESB 主要包含下列内容：

(1) Mule 服务器：一个在 Mule 应用环境中自动加载的服务器应用程序。

(2) 描述器：描述器组件描述了一个 Mule ESB UMO 属性。新的 Mule ESB MUO 对象能被它们所关联的描述器初始化。一个描述器包含：UMO 组件名，UMO 组件实现类，UMO 组件版本，入站和出站提供者，异常策略，入站和出站路由器，接收和发送切入点，拦截器，入站和出站转换器，各种各样的特性。

(3) 通用消息对象(UMO) API：定义了所有被 Mule ESB 管理的任务和对象交互。

(4) 通用消息对象(UMO)组件：在 Mule 系统中，UMO 组件可以是任何在系统中接收、处理和发送事件消息的组件。

(5) 连接器：连接器是一些组件，它们可以连接到外部系统或其他协议、管理那些系统或协议的状态。一个连接器负责发送消息到管理消息接收器、外部消息接收器的注册和注销。

(6) 终端调解者：当 UMO 组件接收到一个事件时，终端调解者决定去调用它的什么方法。

(7) 转换器：转换器组件负责双向转换消息或事件的有效载荷？当一个事件到达接收的对象之前，转换器可以链接到一起去进行一系列的装换操作。

(8) 消息适配器：消息适配器提供一种公共的方式去读外部系统的异构数据。

(9) 提供者：提供者是一些组件，管理把事件数据发送到外部系统、从外部系统接受事件数据和转换事件数据等事项。在 Mule ESB 框架里，他们能连接到外部系统或其他组件。一个提供者就像一个从外部系统进入 Mule ESB 或从 Mule ESB 内部访问外部系统的桥接器。实际上，提供者有一组对象组成，可以与下层系统连接并与之通信。提供者的组成部件有：连接调度者，传送系统到系统；连

接器，负责连接到下层系统；转换器，转换从系统接收到的或要发送到系统的数据；终端，所建立连接的通道地址；消息接收器，从系统接收事件；事务配制，定义连接的事物属性。

(10) 消息接收器：消息接收器是一系列终端监听线程，负责从外部系统接收数据。

(11) 消息调度者：消息调度者发送（同步）或派遣（异步）事件到下层系统。

(12) 消息路由器：消息路由器是一系列组件，可以使被配制的 UMO 组件依据消息或其他配制路由一个消息到不同的提供者。

(13) 代理：代理是一些绑定到外部服务的组件。例如 JME 服务器。

(14) Mule 模型：一个 Mule ESB 模型封装和管理一个 Mule ESB 服务器实例运行时的行为。一个模型包含：UMO 组件，描述器，一个终端调解者，一个生命周期适配器工厂，一个池化工厂，一个组件调解者，一个异常策略。

(15) Mule 管理器：Mule ESB 管理器，如图 3-2 所示，它维护和提供以下服务：终端、转换器、代理提供者、连接器、拦截器堆栈、应用程序属性、一个 Mule ESB 模型、一个 Mule ESB 服务器、事物管理器、Mule ESB 配制。

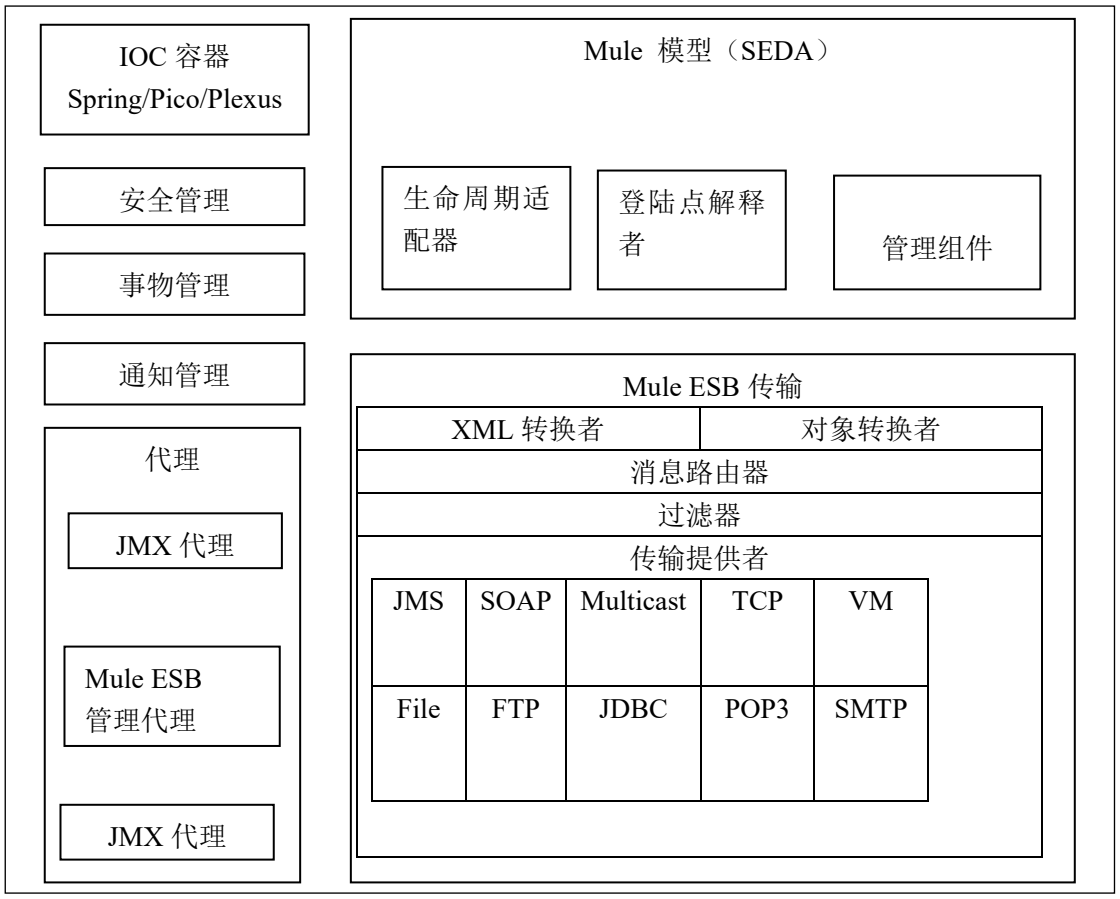


图 3-2 Mule ESB 管理器

3.1.2 Mule ESB 对第三方的支持

为了更加便于开发，Mule 已经将很多技术融入其中。目前已经融入的产品或技术如表 3-1 所示。

表 3-1 Mule 第三方产品或技术支持

产品或技术	说明
Email 提供者（Email Provider）	用来和 POP3 和 IMAP(消息访问协议)邮箱进行连接并利用 javax.mail API 通过 SMTP 协议发送事件。对于每个协议都提供了 SSL 机制。Email 连接者可以通过 Mail 的所有协议以及其它相关的协议发送和接收事件的有效载荷。
Servlet 提供者（Servlet Provider）	Servlet 连接者只面对一个 Servlet 执行，该执行主要是接收请求，然后连接者将请求传送给任何合法的接收者。这个连接者有分发器的概念，它可以被一个请求触发，但有可能不返回响应。
XMPP 提供者（XMPP Provider）	利用 Smack API 通过 XMPP(jabber)实例消息协议来激活它。
WSDL 提供者（WSDL Provider）	通过获取服务的 WSDL 来调用远程的 web 服务。Mule ESB 会为服务创建一个动态的代理并调用它。
UDP 提供者（UDP Provider）	UDP 连接者可以以自带寻址信息的，独立地从数据源行走到终点的数据包的形式发送和接收事件。
Multicast 提供者（Multicast Provider）	Multicast(多点传送)连接者通过多点传送组件发送和接收事件。
SSL 提供者（SSL Provider）	SSL 连接者利用 SSL 或者 TLS 进行可靠安全的 SOCKET 通信。
Stream 提供者（Stream Provider）	允许读写 streaming 数据。这个类似于 java 的 System.out 和 System.in。
SOAP 提供者（SOAP Provider）	通过 Apache 的 Axis 或者 WebMethods 的 Glue 来发布和调用服务。
TCP 提供者（TCP Provider）	通过 TCP 协议发送和接收事件。
EJB 提供者（EJB Provider）	EJB 连接者允许 EJB 状态 beans 作为事件流的一部分被调用。通过给组件一个 EJB 输出的端点，它就可以调用远程对象和返回一个结果。

JMS 提供者（JMS Provider）	这是标准的 JMS1.0.2b 和 1.1 连接者。这个连接者展示了 1.0.2b/1.1 规范的所有特征。
IMAP 提供者（IMAP Provider）	IMAP（消息访问协议）传输提供者接收来自 IMAP 收件箱的消息并可以利用 SSL（IMAPS）和 IMAP 邮箱进行连接。
POP3 提供者（POP3 Provider）	POP3（邮简协议 3）传输提供者接收来自 POP3 收件箱的消息并可以利用 SSL（IMAPS）和 POP3 邮箱进行连接。
File 提供者（File Provider）	通过文件连接器可以读写本地文件系统。可以通过配置文件连接器对读写的文件数据进行过滤。
FTP 提供者（FTP Provider）	通过 FTP 连接器可以读写远程 FTP 服务器上的文件。
HTTP 提供者（HTTP Provider）	HTTP 连接者器通过 HTTP 协议发送和接收事件。它还含有一个 HTTPS 连接器用来进行安全可靠的 HTTP 通信。
Quartz 提供者（Quartz Provider）	提供时序安排特征给 Mule ESB。它基于 OpenSymphony 的 Quartz 调度程序。
VM 提供者（VM Provider）	通过 VM 连接器，组建之间可以进行内部的 VM 通信。它还提供了 VM 短暂的或是长久的队列可选配置。
SMTP 提供者（SMTP Provider）	通过 SMTP 和 SMTPS(安全的)协议发送消息。
JDBC 提供者（JDBC Provider）	通过 JDBC 连接关系数据库。支持简单表的读写操作。
AS400DQ 提供者（AS400DQ Provider）	用来和 IBMAS400DQ 消息服务器进行连接。
RMI 提供者（RMI Provider）	通过 jrmp 发送和接收事件（目前只能进行分发嫌作）。
VFS 提供者（VFS Provider）	使有权使用多种协议例如 WebDav、Samba、Jar/Zip 等等。目前只在 Sandbox 里面可用得到。

3.1.3 Mule ESB 服务器端

Mule ESB 服务端主要有：

(1) 端点(Endpoints)

一个逻辑的，可配置的实体，它绑定在组件或外部网络资源上。用来控制事件的发送者和接收者。它有下面几个属性： Connector：用来连接底层所支持的传输；Filter：对端点接收到的数据进行过滤；Endpoint URI：表示一个资源提供者或者本地或远程接收者的地址。它必须是合法的 URI；Transaction：当一个事件发送或被接收时可以一个事务就可以开始执行；Properties：根据不同端点实例的连接者的需求不同可以对需要的特性进行重新设置。例如，当使用 SMTP 端点的时候你可能会需要重新设定来源地址。

(2) 通用消息对象组件(UMO Component)

UMO 代表 Universal Message Object，它是一个可以接收来自于任何地方的消息的对象。UMO 组件就是你的业务对象。它们是执行引入的事件之上的具体业务逻辑的组件。这些组件是标准的 Java Bean，组件中并没有任何 Mule ESB 特定的代码。Mule ESB 基于你的组件的配置处理所有进出组件的事件的路由和转换。

(3) Mule 管理器(Mule ESB Manager)

Mule ESB Manager 是 Mule ESB Server 实例的中心(也称为一个节点户或者 Mule ESB Node)。其主要的角色是管理各种对象，比如 Mule ESB 实例的端点、连接器和转换器，这些对象然后被用来控制进出你的服务组件的消息流，并且为 Model 和它所管理的组件提供服务。

(4) Mule 模型(Mule ESB Model)

Model 是管理和执行组件的容器，它控制进出组件的管理线程、消息流、生命周期和缓充池。默认的 Mule ESB Model 是基于 SEDA 的，它使用一个有效的基于事件的队列模型来获取的最大的性能和直通性。

(5) Mule ESB 服务器通知(Mule Notification Manager)

Mule ESB 有一个内置的消息机制，通过它可以接进服务器通知信息。例如添加组件，初始化模型或者启动管理者的通知消息。

目前有 10 种服务器通知消息。定制通知，在对象本身定制消息监听器时产生或用来在代理、组件连接器等等上面定制消息；安全通知，请求没有通过安全认证时产生；管理通知(Management)，监控的资源还为实现时产生；管理通知(Admin)，当请求被 Mule ESB 接收到时产生；管理者通知，Mule ESB 管理器状态发生变化时产生，如启动，停止等；连接通知，连接着尝试连接它的资源时产生；消息通知，系统发生或收到事件时产生；模型通知，Mule ESB 模型状态发生变化时产生，如初始化，启动等；组件通知，特殊组件启动或停止等状态变化时产生；空间监控通知，当空间执行如 JavaSpaces 等产生。

(6) Mule ESB 安全管理(Mule Security Manager)

Mule 安全管理有三种方式，端点利用传输的特定细节或普通的认证方法对请求信息进行认证、在 Mule ESB 管理者处定义安全管理者和配置安全过滤。

(7) Mule ESB 事务(Mule Transaction Manager)

Mule 事物有两个特性：对底层事务管理者来说是不可知的；在输入端点处进行配置。

(8) Mule ESB 事件(Mule Event)

Mule ESB 对事件提供三种响应方式：同步，所有的事件在同一个执行线程中执行；异步，许多事件可以在网一时间被同一组件以不同的线程进行处理；请求-响应，发送的请求必须在事先定好的时间内响应。

3.1.4 Mule ESB 客户端

为 JAVA 客户端提供的简单接口，通过它可以从 Mule ESB 服务器端或其他应用系统处接收和发送事件。它提供以下几种功能：利用 Mule ESB 提供的传输和其他应用系统进行通信；发送或接收事件给远程的服务器；发送或接收事件给本地的服务器；在 Mule ESB 服务器端注册或注销组件[2][7]。

3.2 非功能性需求分析

非功能分析主要是对系统的可靠性、易用性、安全性等方面进行分析，主要包括以下几个方面：

(1) 易用性：主要是设计良好的人机交互界面本系统具有良好的简体中文操作界面详细的帮助信息，对 Web 服务的管理和系统的维护都通过操作界面完成提供简单易懂的用户手册，结合页面截图详细展示每个功能操作方法，页面应符合操作简单清晰明确的原则，不至于引起用户的误解。

(2) 可靠性：应保证系统 24 小时正常运行，防止单点失败，提供系统备份和恢复功能，确保每个功能按照需求执行，不出现错误现象为满足系统的可靠性和容灾性要求，系统支持通过增加硬件的方式(包括 CPU 内存硬盘等) 提升系统能力，支持因并发访问用户数增长而对系统的平滑扩容同时本系统采用集群设计，以支持可靠性和容灾性的要求。

(3) 性能要求：根据要求，系统至少需要支持大于 100 个并发用户数；服务调用响应时间，在数据量并发量大的条件下＜10 秒，在数据量较小情况下＜100 毫秒，页面响应时间＜10 秒，简单页面＜1 秒(网络速度不低于 2M 的传输情况下) 为了提高系统性能，系统使用缓存，包括服务信息的缓存和日志的缓存等。

(4) 安全性：支持 HTTPS 的访问方式，保证交互的安全性，通过设置 Tomcat 服务端证书是可以实现的；系统只允许经过授权的用户访问系统；通过角色管理

用户的权限，特定角色用户对应特定的操作行为；并对服务操作的关键活动进行审批。

(5) 可扩展性：基于 SOA 的架构设计理念，使用标准的接口协议，设计过程采用可扩展的设计模式，减少系统之间的耦合性。

3.3 功能性需求分析

(1) 应用管理模块的主要功能：提供 Mule ESB 应用的查询、停止、启动、下线功能，统计应用中各个服务的使用情况，包括正确连接数、错误连接数等相关统计数据。

(2) 服务管理模块的主要功能：提供 Mule ESB 中注册服务的查询、连接、断开功能，服务信息包括服务的基本信息绑定信息安全和日志信息附加文档等服务。

(3) 服务监控管理模块的主要功能： 主要是对服务的调用情况进行统计，包括各服务调用次数成功次数失败次数成功率最大和最小响应时间平均响应时间等，然后将这些统计信息进行展现。

(4) 服务器监控和管理模块的主要功能：监控 Mule ESB 的运行情况以及对 Mule ESB 的相关操作，包括重启 Mule ESB 和关闭 Mule ESB。监控 JVM 内存的使用情况包括堆内存、非堆内存的使用，设置 JVM 内存。监控服务器的资源使用情况，包括 CPU、物理内存、虚拟内存的使用情况。

(5) 日志管理模块的主要功能：主要是对 Mule ESB 运行过程的监控，其中包括 Mule ESB 启动日志，应用服务部署日志以及服务调用情况日志。

(6) 安全性管理模块的主要功能：本模块包括 JMX Mule Agent 以及 Web 控制台安全管理模块。Web 控制台安全管理为权限管理角色管理和用户管理通过权限管理，系统管理员对服务管理系统的功能进行划分，并分配编号，同时提供权限判断接口，各个功能模块在运行时通过调用该接口以判断当前用户是否具备操作某功能的权限通过角色管理，系统管理员创建角色，并为角色赋予权限通过用户管理，系统管理员可为用户分配相应的角色，用户就具备了角色所拥有的权限。JMX Mule Agent 安全性主要是对 Mbean server 中 Mbean 的管理，根据用户设定的安全策略对 Mbean 的访问进行控制，用户可以根据多种策略进行设定。

下图是整个 Web 控制台的功能需求图，如图 3-3 所示。

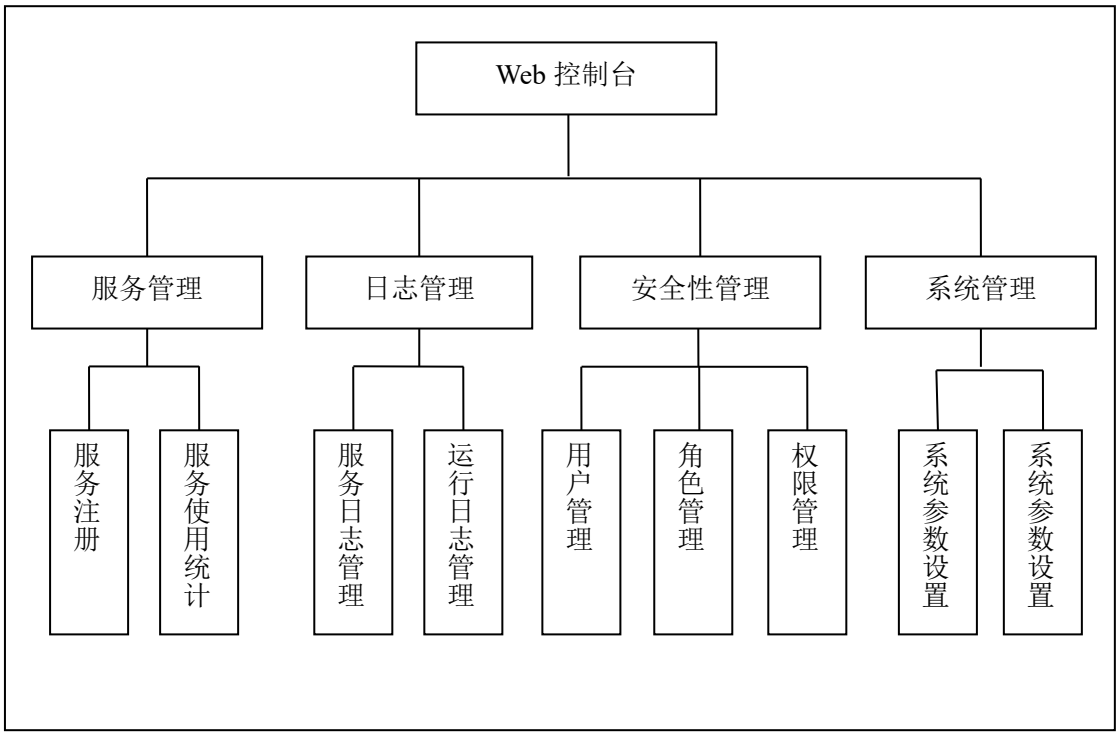


图 3-3 功能需求图

3.4 可行性分析

可行性研究就是在软件设计之前，对软件在技术上、经济上、操作上是否可行进行分析和论证。

技术可行性：是指利用现有的开发技术能否达到系统的需求目标，在规定的期限内能否完成开发工作，软硬件能否满足开发者的需要等。软件方面，本系统采用的是 B/S 模式，对客户端的要求比较低，维护升级较为简单；应用服务器采用 Tomcat，安全稳定；数据库选用 MySQL，它能够很好地处理数据，并保证数据的完整性。因此，系统的软件开发平台已成熟可行。硬件方面，开发人员均配有个人电脑来进行项目开发，公司配备了专有的服务器来进行项目的部署和测试，开发人员可通过内部局域网访问测试服务器。在硬件开发环境上，现有条件完全能够满足系统的开发和测试这两个阶段的工作。

经济可行性：主要分析系统的开发成本是否会超过它所带来的经济效益。现在公安局有很多个单一的业务系统并且他们都是独立运行的，所有的系统都不是由单一的公司单一的技术团队使用相同的技术开发完成的。因此如果投入大量的资金去修改现有的系统来连接各个系统此方法是不可行的。使用 Mule ESB 经济实惠而且不需要开发人员去熟悉现有的系统，只要他们将各自提供的接口集成到 ESB 即可。如果使用了 Mule ESB 那么一个可视的管理控制台就变得尤为重要了。此外，公司的服务器以及现有设备已满足开发需求，不需要额外购置其他设备，

无需投入过多的开发经费。所以，本系统在经济上是可行的。

操作可行性：主要是从用户的角度来考虑本系统是否简单易用。本系统各个模块划分清晰，模块内的功能设计条理清楚，在界面设计上易于服务发布人员和服务使用人员操作使用。该系统采用 B/S 架构，用户无需安装客户端，可在任何连接网络的计算机上访问该系统，非常便于用户使用。所以，本系统在操作上也是可行的。

综上所述，本系统的开发目标非常明确，在技术、经济及操作方面都具有可行性，并且投入少，见效快。因此，本系统的开发是完全可行的。

3.5 本章小结

通过分析 Mule ESB 的实现架构为设计为实现 Mule Agent 打下了坚实的理论基础。

需求分析的主要目标是构建现实世界的模型。同时，需求分析是获取用户需求的主要途径，是系统分析与设计的桥梁。需求分析不清晰会导致后续的开发设计也偏离现实世界的模型，导致系统所提供的功能无法或难以满足需求，从而导致系统重构，更严重的导致项目失败。本章从功能性需求、非功能需求已经可行性方面分析了 Web 控制台，为下面设计与实现架好了桥梁。

第四章　基于 ESB 的服务发现和管理的设计

4.1 Web 控制台设计

由于 Mule ESB 已经实现了 JMX 内核架构，因此我们只需要部署一个 JMX 代理应用到 Mule ESB 的项目中，就可以通过开发 JMX 代理客户端来监控和控制 Mule ESB 的某些行为。另外，为了可以在管理控制台方便地查阅 Mule ESB 的日志信息，需要改写 Mule ESB 的日志行为，并将其运行日志存储到指定的数据库中。

管理控制台使用 REST 风格设计。根据 MVC 分层思想，管理控制台分为视图层、控制层和模型层三层。其中，视图层是展示和操作的用户接口；控制层负责信息的转发；模型层可以进一步分为两部分：第一部分是 JMX 代理客户端模型，该模型通过远程调用 Mule ESB 的 JMX 内核实现，用以规格化 JMX 的 MBean，第二部分 Mule ESB 自动将 ESB 所有的日志信息写入数据库，然后通过访问数据库获得 Mule ESB 的日志信息。

管理控制台的体系结构如图 4-1 所示。

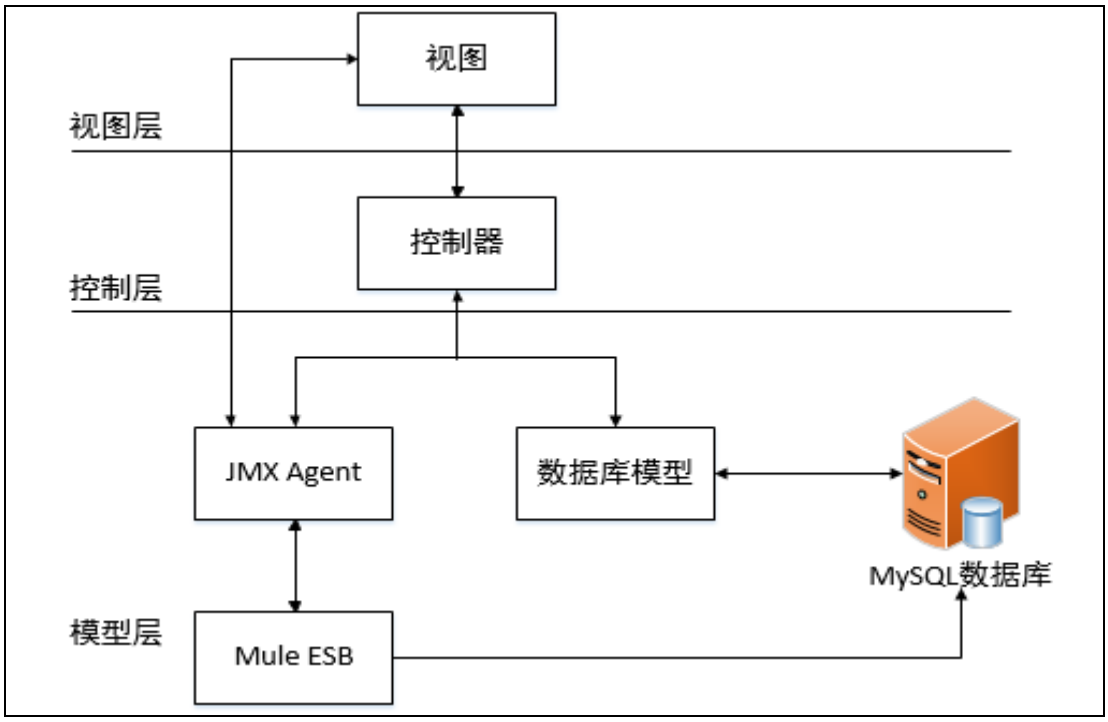


图 4-1 管理控制台分层结构

(1) 视图层：整个页面框架使用 BootStrap 框架技术搭建，采用响应式设计，

支持移动端、桌面端浏览。所有的页面内容都通过 Ajax 异步的方式加载。

(2) 控制层：使用 Spring 技术，将请求映射根据业务的需求分类。该层是视图和模型的适配中介，可将模型与具体的业务结合起来。使用 AOP 对用户权限进行控制。

(3) 模型层：可以分为数据库模型和 JMX 代理客户端模型两大类。数据库模型，主要负责管理用户权限信息、Mule ESB 使用 Log4j 自动写日志以及从日志数据库获取数据，使用 Hibernate 来实现对数据库的操作；JMX 代理客户端模型，提供了对 Mule ESB 服务实时信息的获取和对 Mule ESB 服务操作的调用功能。

基于 Web 的动态服务管理前端平台是一个集合服务注册，服务发布查询，服务管理的多功能平台，为 Web 服务的发布者和 Web 服务的使用者都提供了一个便捷和界面友好的环境。

服务注册和服务查询：Web 服务的发布者可以登录页面，注册发布自己的 Web 服务，在友好的界面的引导下，填写注册信息，提交后转入后台的服务注册模块进行逻辑处理，动态的接入 ESB 总线系统。Web 服务的使用者可以使用服务查询功能，浏览已经实现注册的 Web 服务条目，确定 ESB 系统，列表显示 Web 服务名称，特性和服务限制条件等等信息，用户可以根据自身情况决定是否有需要的服务功能个体。表 4-1，图 4-2 分别为服务注册和查询模块用例表和图。

表 4-1 服务注册和查询模块用例表

服务名	使用者	用例描述
服务注册	服务发布者	用户将打包好的项目直接上传，后台发动态部署后用户即可以访问。
服务查询	服务使用者	提供服务查询的功能，由已经服务注册的内容动态生成，方便用户查找相关功能的服务。

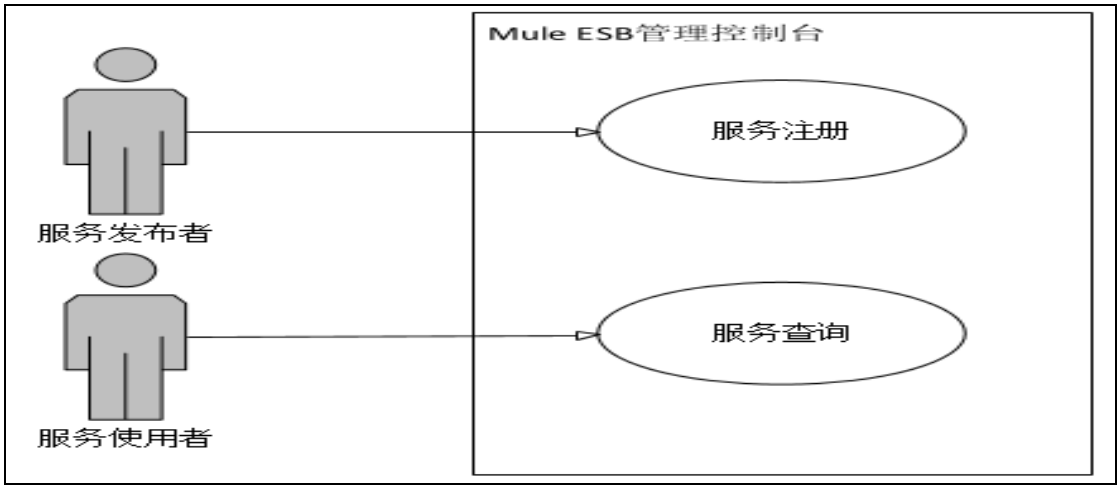


图 4-2 服务注册和查询模块用例图

服务管理和监控：用户可以通过该模块对已经实现注册的 Web 服务实现远

程监控和管理操作。其中包括了服务发布管理，服务运行状态管理和监控，运行实例 管理等多个功能。同时，用户可以通过该模块修改已经注册的服务信息，动态加载更新服务。表 4-2 为服务注册和查询模块用例表，图 4-3 为对应的监控模块用例图。

表 4-2 服务管理和监控模块用例表

服务名	使用者	用例描述
查看已经发布服务列表	服务发布者	服务监控和管理模块提供已经发布服务列表功能供用户使用。
服务监控和管理	服务发布者	服务监控管理模块提供服务状态监控和管理功能供用户使用。用户能够监控已经发布在 ESB 中的服务，并能管理已经发布的服务。
服务注销或注册	服务发布者	服务监控和管理模块提供了流程定义停用的功能供租户使用。
服务实例开启	服务发布者	服务监控和管理模块提供服务实例开启功能，用户能开启已经发布的服务实例。
服务实例停用	服务发布者	服务监控和管理模块提供服务实例停用功能，用户能停用已经发布的服务实例。
服务实例激活	服务发布者	服务监控和管理模块提供服务实例激活功能，用户能激活已经发布的服务实例。
服务实例终止	服务发布者	服务监控和管理模块提供服务实例终止功能，用户能终止已经发布的服务实例。

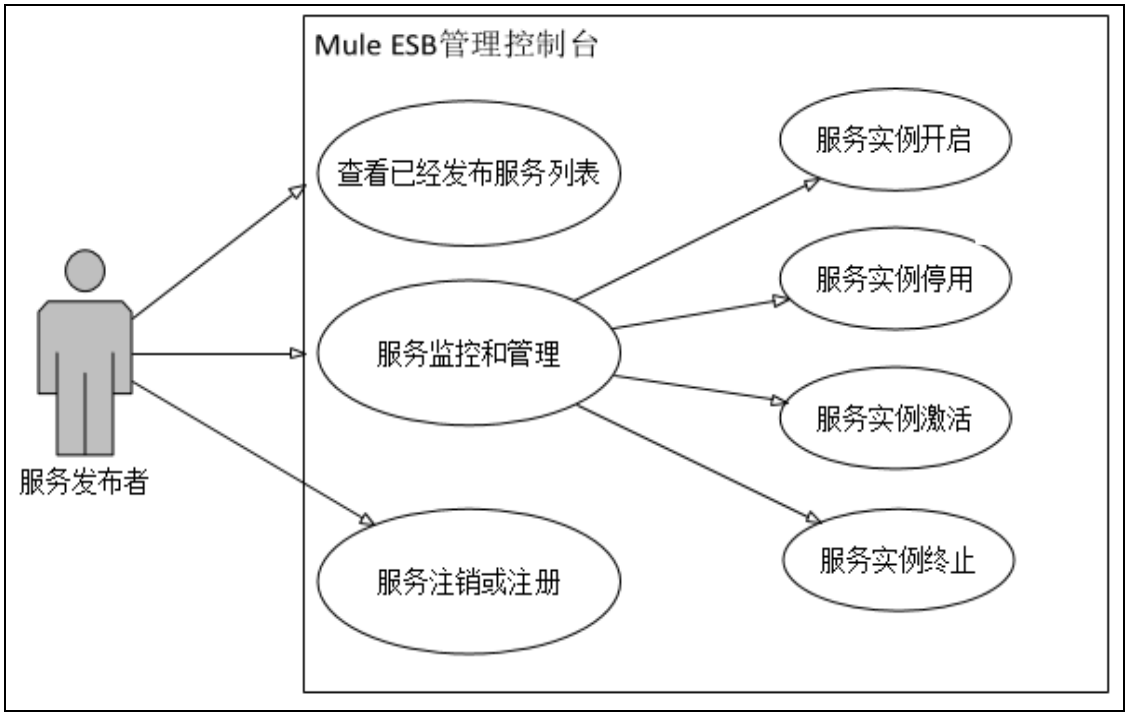


图 4-3 监控模块用例图

4.2 Web 控制台权限控制设计

Web 控制台权限控制模块用于在管理用户登录后该用户可使用的系统功能模块。该模块采用经典 RBAC（Role-Based Access Control，基于角色的访问控制）模型设计，通过建立用户、角色、资源之间的关联关系，来控制用户可进行的操作。资源是通过 URL 的方式来获取的。用户与角色之间，角色与权限之间，是多对多的关系。Web 控制台 RBAC 权限模型如下图 4-4 所示：

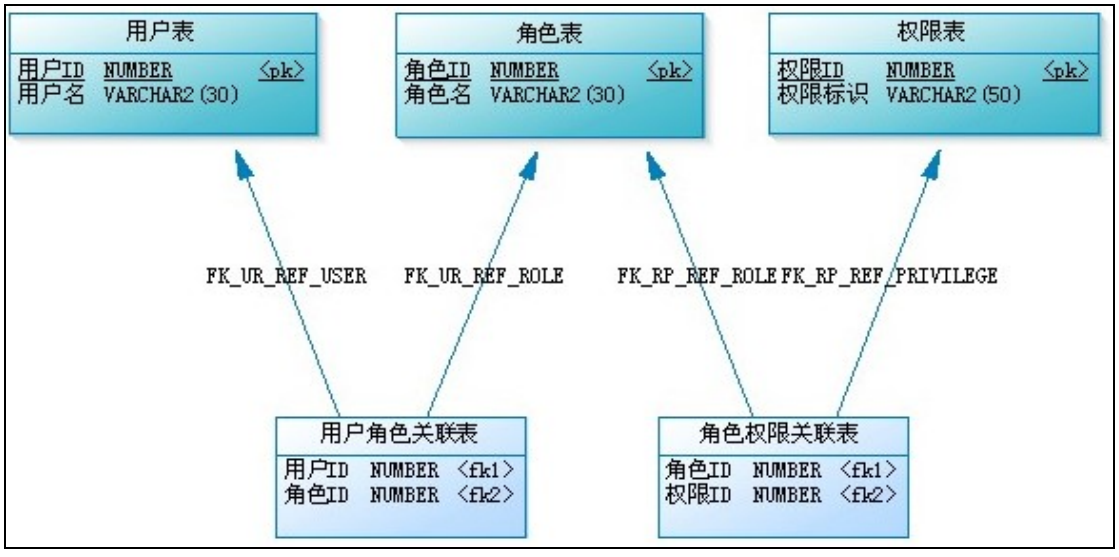


图 4-4 RBAC 权限模型图

使用 RBAC 权限控制具有两大优势：

(1) 由于角色/权限之间的变化比角色/用户关系之间的变化相对要慢得多，减小了授权管理的复杂性，降低管理开销。

(2) 灵活地支持企业的安全策略，并对企业的变化有很大的伸缩性。下面是 Web 控制台使用 RBAC 权限控制的主要步骤：

首先，用户是管理用户，用户可通过添加用户功能创建。

其次是角色，系统初步划分的角色有系统管理员、服务发布者、服务监控者、服务器管理员、服务使用者。

再次是资源，在本系统中每个系统功能模块作为一个资源，并且为每个资源分配一个资源编号。系统功能模块有：角色管理、用户管理、权限管理、服务器监控和管理、服务注册模块、服务查询模块、服务使用统计模块、日志管理模块等资源。

最后，将用户与角色以及角色与资源关联起来，以达到权限控制的目的。

对于一个用户来说，可以成为多个不同的系统角色，每个系统角色又可拥有多个不同的系统资源。

系统初始的角色资源分配如表 4-3 所示。

表 4-3 系统角色资源分配表

角色	资源
系统管理权	角色管理、用户管理、权限管理、服务器监控和管理、服务注册模块、服务查询模块、服务使用统计模块、日志管理模块等系统所有其他功能。
服务发布者	服务在线注册。
服务监控者	监控所有应用以及所有服务的使用情况。
服务器管理员	对 Mule ESB 服务器进行管理，对 JVM 进行管理（包括 JVM 参数设置等）。
服务使用者	查看已经发布的应用以及服务。

该模块提供的最主要的功能是在用户登陆 Web 控制台时提供系统权限查询查询流程如下：

(1) 用户登陆 Web 管理控制台。

(2) 查询登录用户所具有的角色。

(3) 查询用户所具有角色所拥有的资源。

(4) 删除重复项。

(5) 返回用户的资源。

4.3 JMX 代理设计

JMX 代理设计模型如图 4-5 所示。

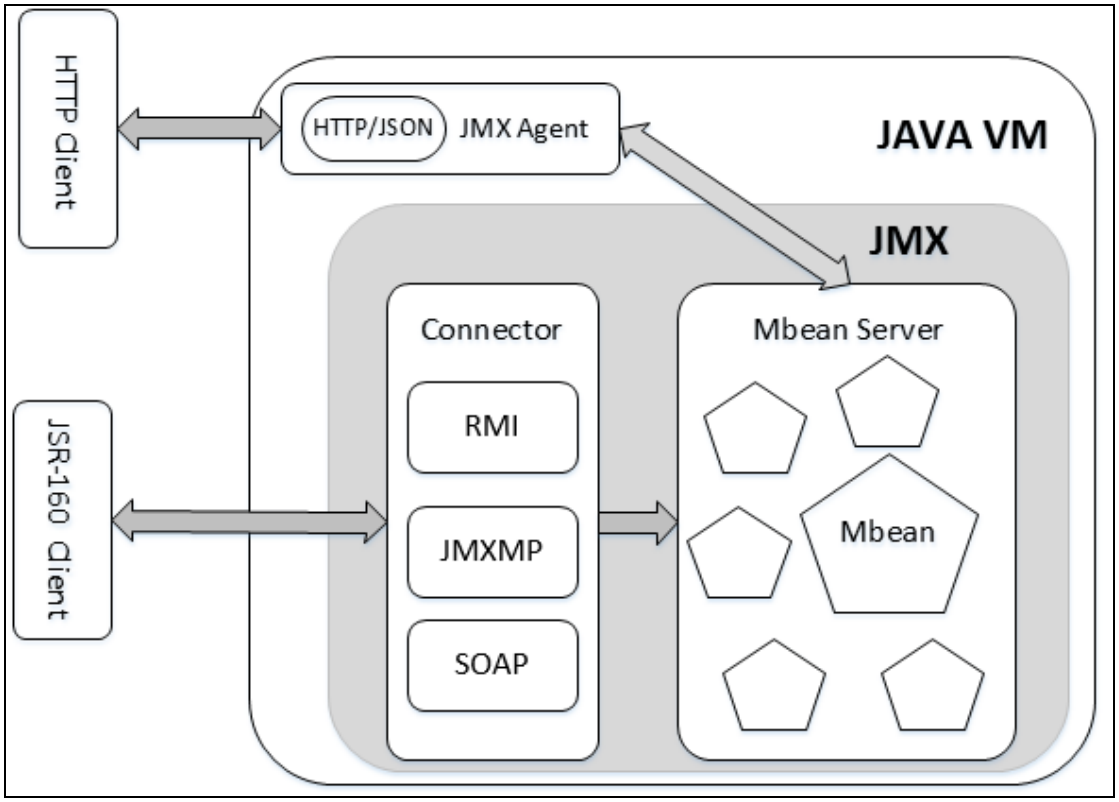


图 4-5 JMX Agent 模型

JMX Agent 是注册在 Mule ESB 中的一个 Mule Agent，它内嵌了 Jetty 服务器。它和 Mule ESB 有一样的生命周期，因此它可以在 Mule ESB 启动或销毁的时候初始化或者销毁内嵌的 Jetty 服务器。Mule ESB 提供了一组管理 API 给注册的 Mule Agent。

上图说明了 JMX Agent 工作的环境。它的前端是一个基于 HTTP 访问的提供 JSON 数据响应的协议，它可以访问到本地的 JMX MBeans。它运行于 JSR-160 空间之外，因此它需要不同的设置。目前有很多技术可以通过 HTTP 对外提供服务。JMX Agent 最突出的特点是它被放在了一个 Servlet 容器里面-Jetty 服务器。用户在浏览器输入管理员在 Mule ESB 注册好的的 JMX Agent 地址请求数据，HTTP 先是访问的 JMX Agent 中的 Servlet，Servlet 再根据用户的输入参数发送请求到 Mule ESB 中的 Mbean Server，Mbean server 将用户请求的 Mbean 数据在通过 JSON 格式的形式传到 Servlet 然后再返回给用户。

JMX Agent 的架构和 JSR-160 连接器是不一样的。最明显的不同就是 JMX Agent 无类型的方法。JSR-160 发布于 2003 年，它设计的目标和 JMX Agent 是不一样的。它是一个客户端可以透明地调用 MBean 规范，无论 MBean 驻留在本地

或远程的 MBean Server[1]。它提供了一个很好的，便于 JAVA 访问的 API，但它同时也很危险，因为它隐藏了 JMX 的隔离要求。有几个微妙的问题，性能是其中之一。一个调用是本地的还是远程的是一个问题。调用者至少应该知道会发生什么，什么后果。另外，也有消息传递模式，其中包括远程明确，这样调用者知道，从她是调用一个潜在的昂贵的远程调用编程模型。这可能是最主要的原因，RMI（JSR-160 连接器的默认协议栈）失去的市场份额，作为更明确的远程协议。

JSR-160 的另一个问题是它在 RMI 隐含的依赖和它的一个完整的（Java）的对象序列化机制，通过管理信息通过线路的要求。这将导致其他语言（非 JAVA）无法使用它。JMX Agent 采用的是无类型的方法，使用一些轻量级的序列化到 JSON（在两个方向，但有点不对称在其能力）。当然，这种方法也有一些缺点，但也颇有些优势。至少它是在 JMX 世界上独一无二的。

在 JSR-160 中远程安全是或有或无的。要么所有的 MBean，或者没有 MBean 可访问（除了当你的应用服务器使用安全管理器，但这并不是经常发生的情况）。不同的是，JMX Agent 允许在 XML 安全策略文件中定义细粒度的安全性。它允许根据特定的 Mbean 名字（或者特定种类的 Mbean）、属性、操作、源 IP 地址（或者子网）以及操作的类型限制访问，下一节将详细介绍。

4.4 JMX 代理安全设计

上一节中简单提到了 JMX 代理的安全设置，这一节将详细介绍 JMX 代理安全的设计。由于 JMX 要么暴漏全部 Mbean 要么不暴漏，因此针对 Mbean 的安全控制变得尤为重要了。可以通过 Mule ESB 对 XML 文件的动态加载机制进行权限控制，同时 XML 文件能够很好的和 JAVA 代码相结合，用户只需经过简单的 XML 文件编写工作，即可以对 JMX Agent 进行安全设定，并且设定后 Mule 也能进行动态加载，实现了用户对安全的实时控制，并且不影响服务的使用。下面是几种安全控制方法的详细设计：

(1) 控制特定 Mbean 的访问：可以通过 XML 标签进行设定特定的 Mbean 是否能够被 JMX Agent 访问，因为有很多 Mbean 涉及到了软件的重要信息，因此管理员是不想暴漏这些接口给外界的。

(2) 基于对 Mbean 的操作进行选择性控制：Mbean 可以分为三类，分别是只读的，可读写的，可操作的。管理员可以根据这三类进行选择性控制。

(3) 基于 IP 地址控制对 Mbean 的访问：目前很多应用安全设置中是具有这一项的，不管什么软件如果放在外网了，那么他就不是很安全了，因为有很多因素可能导致软件数据泄漏，因此基于 IP 地址访问控制就变得尤为重要了。可以通过 XML 文件定义某些 IP 能够访问 Mbean 或者某类 IP 可以访问。

(4) 基于用户 HTTP 请求的类型行进控制：在 Web 开发中，开发人员也会进行这个控制，例如当用户删除，添加，修改数据的时候使用 POST 请求，查看数据的时候使用 GET 请求。在操作敏感性数据的时候限制使用 POST 请求访问比使用 GET 请求访问更加安全。

4.5 本章小结

本章设计了一种远程监控和管理 Mule ESB 服务的解决方案。该解决方案采用 JMX 方式进行连接和操作 Mule ESB。作为 Java 的标准 API，JMX 具有跨平台的优势，而且其性能及其稳定性都能得到保证。通过和 JSR-160 实现原理的比较阐述了 JMX Agent 实现方案的优点。这一章为实现 Web 控制台打下了结实基础。

第五章　基于 ESB 的服务发现和管理的实现与结果

5.1 Web 控制台实现

Mule ESB 日志处理，创建日志记录表 mule_log，日志记录表主要记录 ESB 的运行日志以及服务运行日志，各个字段如表 5-1 所示：

表 5-1 日志记录表(mule_log)

字段名	描述
id	int，主键，自增
log_location	varchar(100)，执行的方法，完整类名+方法
log_thread	varchar(100)，执行的线程
log_create_time	varchar(50)，日志记录的时间，格式：1992-09-10 00:00:00
log_level	varchar(10)，日志的优先级，有 INFO, WARN, ERROR
log_msg	varchar(255)，日志输出消息
log_time	int，显示从程序启动时到记录该条日志时已经经过的毫秒数

使用 Mule ESB 中的 Log4j 记录日志，需要修改 conf 文件下的 log4j.properties 文件，添加的具体内容如下：

```
#定义写日志的类，这里使用的是经过改写的写数据库的类。
log4j.appender.db=org.apache.log4j.jdbc.MyJDBCAppender
#定义什么级别的错误将写入到数据库中
log4j.appender.db.Threshold=INFO
#设置缓存大小，就是当有 5 条日志信息是数据库才插入一次，提高效率
log4j.appender.db.BufferSize=5
#设置要将日志插入到数据库的驱动,连接方式，用户名，密码
#这里使用的是 MySQL 数据库
log4j.appender.db.driver=com.mysql.jdbc.Driver
log4j.appender.db.URL=jdbc:mysql://localhost:3306/mule_log?useUnicode=true&c
haracterEncoding=UTF-8
log4j.appender.db.user=root
log4j.appender.db.password=root
#定义使用的插入 SQL 语句
log4j.appender.db.sql=insert into sys_log (log_location, log_thread, log_create_time,
log_level, log_msg, log_time) values ('%C. %M', '%t', '%d{yyyy-MM-dd
HH:mm:ss}', '%p', '%m', '%r')
log4j.appender.db.layout=org.apache.log4j.PatternLayout
```

由于字段 log_msg 中可能存在单引号，因此在使用 Log4j 自带的数据库操作类写数据时会发生错误，因此需要定义自己的数据库操作类，将字段中存在的单

引号(‘)替换为转义字符加单引号(\‘)，经过研究发现需要先新建一个类 MyLoggingEvent 并且继承 LoggingEvent 类，然后重写 getThreadName(获取操作线程类)和 getRenderedMessage(获取消息)方法，因为 ThreadName 和 Message 中可能存在单引号。最后继承 JDBCAppender 类然后覆盖 getLogStatement 方法并且调用 MyLoggingEvent 对字符串进行替换操作即可。部分代码如下：

```
public class MyLoggingEvent extends LoggingEvent {
    private static final long serialVersionUID = 8925816852575301213L;
    public MyLoggingEvent(String fqnOfCategoryClass, Category logger,
            Priority level, Object message, Throwable throwable) {
        super(fqnOfCategoryClass, logger, level, message, throwable);
    }
    public String getThreadName() {
        String thrdName = super.getThreadName();
        if (thrdName.indexOf("'") != -1) {
            thrdName = thrdName.replaceAll("'", "''");
        }
        return thrdName;
    }
    public String getRenderedMessage() {
        String msg = super.getRenderedMessage();
        if (msg.indexOf("'") != -1) {
            msg = msg.replaceAll("'", "''");
        }
        return msg;
    }
}
public class MyJDBCAppender extends JDBCAppender {
    protected String getLogStatement(LoggingEvent event) {
        String fqnOfCategoryClass = event.fqnOfCategoryClass;
        Category logger = Category.getRoot();
        Priority level = event.getLevel();
        Object message = event.getMessage();
        Throwable throwable = null;
        MyLoggingEvent bEvent = new
        MyLoggingEvent(fqnOfCategoryClass,logger,level, message, throwable);
        return super.getLogStatement(bEvent);
    }
}
```

整个控制台的实现使用的是 Spring MVC 框架实现的具体的结构图如下图 5-1 所示：

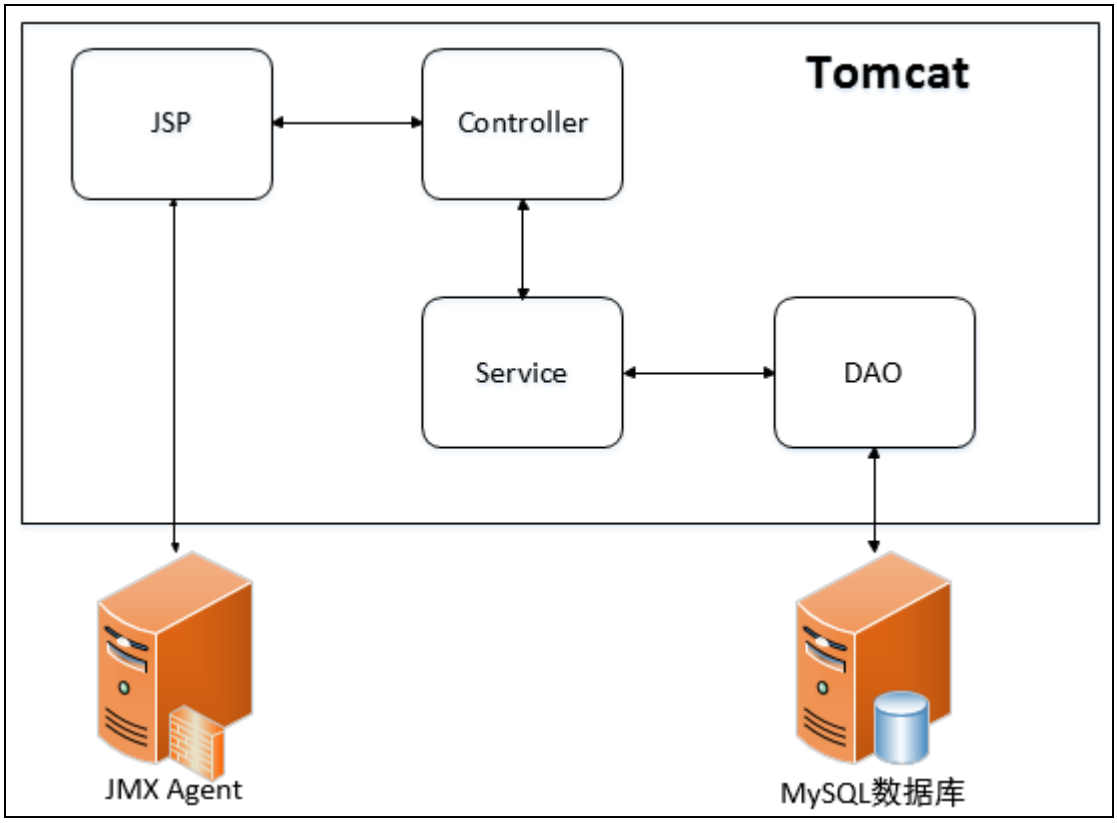


图 5-1 控制台实现结构图

获取数据库中的数据时，前台发送一个请求到 Controller，Controller 再通过 Model 调用数据库操作层 DAO 层，DAO 将获取的数据封装成 Model 返回到 Controller，Controller 最后将结果返回 JSP 页面显示给用户。获取 Mbean Server 中数据时，用户直接访问 JSP 页面，JSP 通过 Ajax 异步发送请求到 JMX Agent 中的 Servlet，Servlet 再将 Mbean 的数据以 JSONP 的格式返回 JS，JS 通过 JS 模版引擎渲染页面元素，将信息以各种方式显示在页面上。

5.2 Web 控制台权限控制实现

Web 控制台权限控制，需要首先创建用户表(mule_user)、角色表(mule_role)、资源表(mule_resource)，三个表以及各个字段如下所示：

表 5-2 角色表(mule_role)

字段名	描述
id	int，主键，自增
role_name	varchar(100)，角色名，非空
create_time	varchar(50)，创建时间，格式 1992-09-10 00:00:00
update_time	varchar(50)，更新时间，格式 1992-09-10 00:00:00
user_id	int，用户 id，外键，与用户表多对一

表 5-3 用户表(mule_user)

字段名	描述
id	int，主键，自增
username	varchar(100)，用户名，唯一，非空
password	varchar(100)，密码，非空
create_time	varchar(50)，创建时间，格式 1992-09-10 00:00:00
update_time	varchar(50)，更新时间，格式 1992-09-10 00:00:00

表 5-4 资源表(mule_resource)

字段名	描述
id	int，主键，自增
resource_name	varchar(100)，资源名，唯一，非空
resource_url	varchar(255)，资源地址，非空
create_time	varchar(50)，创建时间，格式 1992-09-10 00:00:00
update_time	varchar(50)，更新时间，格式 1992-09-10 00:00:00
role_id	int，角色 id，外键，与角色表多对一

下图为权限管理物理模型图，如图 5-2 所示。

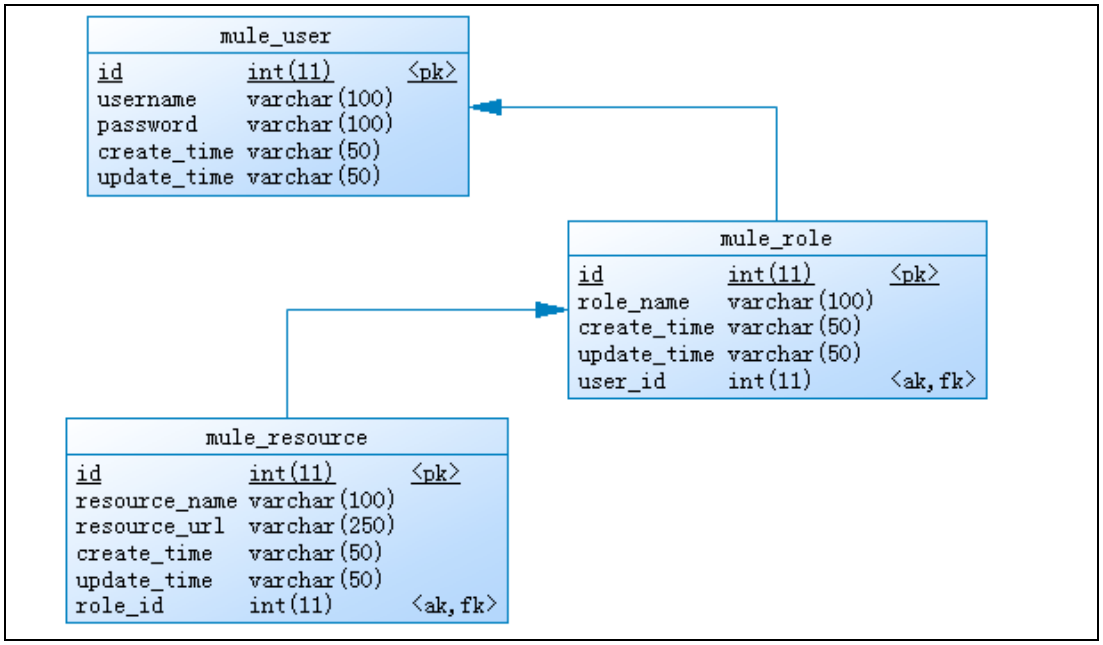


图 5-2 权限管理物理模型图

使用 Spring MVC 中的拦截器技术拦截 Controller 进行权限控制。为了实现 Spring 拦截器我们首先需要了解 Spring 的工作原理，如下：

(1) 用户向服务器发送请求，请求被 Spring 前端控制 DispatcherServlet 捕获。

(2) 前端控制 DispatcherServlet 对请求 URL 进行解析，得到请求资源标识符(URI)然后根据该 URI，调用 HandlerMapping 获得该 Handler 配置的所有相关的

对象，最后以 HandlerExecutionChain 对象的形式返回。

(3) 前端控制 DispatcherServlet 根据获得的 Handler，选择一个合适的 HandlerAdapter（附注：如果成功获得 HandlerAdapter 后，此时将开始执行拦截器的 preHandler 方法）。

(4) 提取 Request 中的模型数据，填充 Handler 入参，开始执行 Handler。

(5) 等 Handler 执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象。

(6) 根据返回的 ModelAndView，选择一个适合的 ViewResolver(必须是已经注册到 Spring 容器中的 ViewResolver)返回给 DispatcherServlet。

(7) 视图解析 ViewResolver 结合 Model 和 View，来渲染视图，并将渲染结果返回给客户端。

不难看出在步骤(3)的时候 Spring 便开始执行拦截器，对请求进行拦截。为了实现拦截器，首先我们需要创建一个类 MyHandlerInterceptor 并且继承 HandlerInterceptorAdapter 类重写 preHandler 方法，所有的请求都会经过这个方法，我们只要在这里获取登陆成功后的与用户，然后再根据与角色表关联，角色表再与资源表管理，即可得到用户所有的可用资源，这里把用户资源存在 session 当中，用户登陆成功后即可以访问自己的资源服务。MyHandlerInterceptor 是需要注册到 Spring 的容器中的下面是注册代码：

```
<!-- 启动 Spring MVC 的注解功能，完成请求和注解 POJO 的映射，添加拦截器，类级别的处理器映射 -->
<bean
class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
        <property name="interceptors">
            <list>
                <bean class="com.mule.interceptor.MyHandlerInterceptor" />
            </list>
        </property>
</bean>
```

5.3 JMX 代理实现

实现访问 Mule 中注册的服务需要先创建 Mule ESB Application，下面为主要实现步骤。

首先需要在 Mule ESB 中配置注册 JMX Mule Agent，注册的方法是在 Mule ESB 软件 apps 文件夹中创建一个 default（名字可以任意命名）文件夹，再在 default 文件夹中创建 mule-config.xml 文件，然后就是编辑 mule-config.xml 中的内容，具体内容如下：

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:management="http://www.mulesoft.org/schema/mule/management"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule/core
      http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
      http://www.mulesoft.org/schema/mule/management
      http://www.mulesoft.org/schema/mule/management/3.1/mule-management.xs
      d">
    <!—class 创建好的 JMX Mule Agent 类名-->
    <custom-agent name="jmx-agent" class="org.jmx.mule.JMXMuleAgent">
        <!—定义访问的端口号，默认 8888-->
        <spring:property name="port" value="8899"/>
        <!—定义访问的用户名和密码，可以不填写-->
       <spring:property name="username" value="admin"/>
        <spring:property name="password" value="123"/>
    </custom-agent>
</mule>
```

JMX Mule Agent 创建类图如图 5-3 所示，MuleAgentHttpServerFactory 工厂类负责调用 MortbayMuleAgentHttpServer 创建内嵌 Jetty 服务器，并且保持单例，现在支持的是 Jetty8 服务器，代码如下：

```
public final class MuleAgentHttpServerFactory {
    static String CLAZZ_NAME = "org.mortbay.jetty.Server";
     protected MuleAgentHttpServerFactory() {};
    public static MuleAgentHttpServer create(Agent pParent, MuleAgentConfig
    pConfig){};
}
```

MortbayMuleAgentHttpServer 负责创建内嵌 Jetty 服务器并将访问 Mbean Server 的代理 Servlet AgentServlet 加载到 Jetty 容器里面，主要代码如下：

```
public class MortbayMuleAgentHttpServer implements MuleAgentHttpServer {
      private Agent parent;
      private Server server;
      MortbayMuleAgentHttpServer(Agent pParent, MuleAgentConfig pConfig){}
```

```
    public void start(){}
    public void stop(){}
    private Server getServer(MuleAgentConfig pConfig){}
    private ServletHolder getServletHolder(MuleAgentConfig pConfig){}
    private Context getContext(HandlerContainer pContainer, MuleAgentConfig
    pConfig){}
    private SecurityHandler getSecurityHandler(String pUser, String pPassword,
    String pRole){}
}
```

JMXMuleAgent 继承 Mule ESB 的 AbstractAgent，使程序和 Mule ESB 连接在一起，它和 Mule ESB 具有相同的生命周期，包括初始化、启动、停止、注册等阶段。JMXMuleAgent 实现了 MuleAgentConfig 接口用来获取 AbstractAgent 提供的一些属性，主要代码如下：

```
public class JMXMuleAgent extends AbstractAgent implements
MuleAgentConfig{
    protected MuleAgentHttpServer server;
    public void start(){}
    public void stop(){}
    public void dispose() {}
    public void registered() {}
    public void unregistered() {}
    public void initialise(){}
}
```

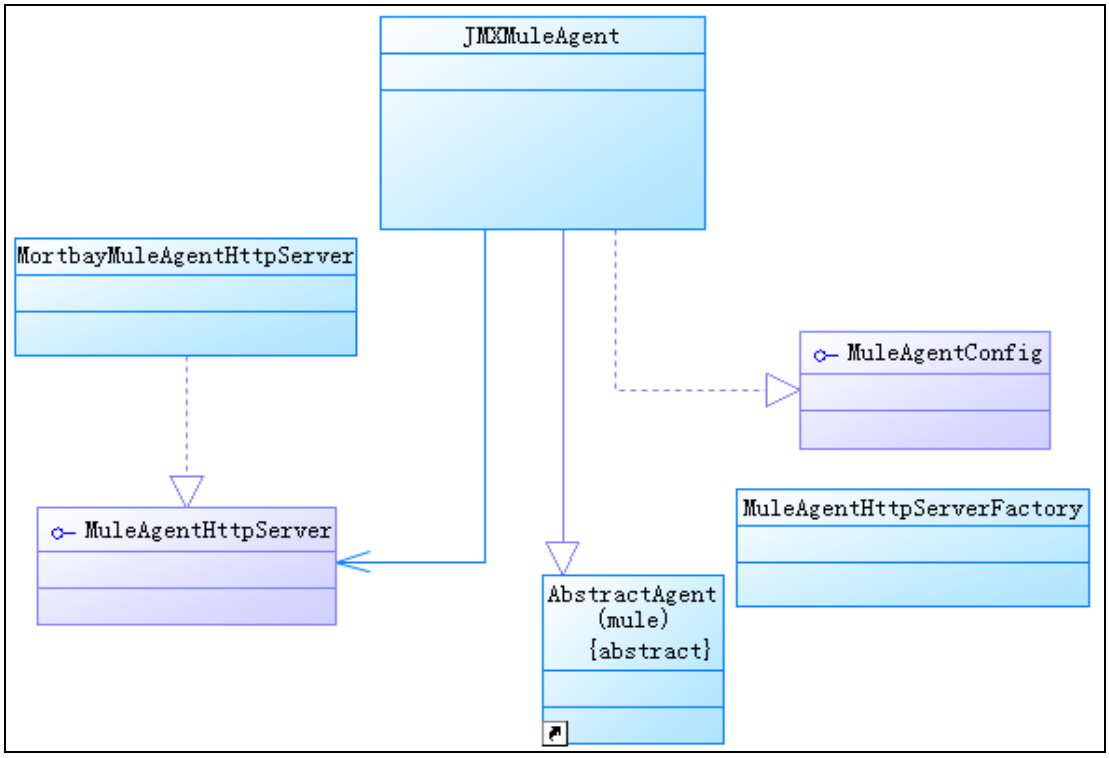


图 5-3　JMX　Mule Agent 创建类关系图

5.4　JMX 代理安全实现

JMX 代理的安全是通过读取 XML 策略文件 jmx-mule-access.xml 定义来实现的，jmx-mule-access.xml XML 文件可以通过配置多种参数来限制 Mbean 的访问，具体方法如下：

(1) 基于 IP 的限制

整体访问可以被授予基于一个或多个 HTTP 客户端的 IP 地址。这些限制是在<remote>部分中指定的，其中包含一个或多个<host>元素。<host>可以是一个 IP，也可以是一个网址。XML 文件内容如下，表示只允许本地访问。

```
<remote>
   <host>localhost</host>
</remote>
```

(2) 根据命令来，JMX 中操作分为三种，分别是 read,读取 Mbeans 属性；write，设置 Mbeans 属性；exec，执行 JMX 操作。在 JMX Mule Agent 中有添加了另外两个操作分别是，list 列出所有可见的 Mbeans 属性以及操作；search，根据 url 中的值查找相关的 Mbeans。xml 文件内容如下:

```
<commands>
   <command>read</command>
   <command>list</command>
```

```
</commands>
```

(3) 限制访问特定的 Mbeans

通过标签<allow>定义可以访问的 Mbeans，<deny>限制访问的 Mbeans，针对当前定义的 Mbeans 不用管（2）中定义的相关操作限制。这两个标签中都可以包含多个<mbean>。<mbean> xml 定义文件如下：

```
<mbean>
  <name>java.lang:type=Memory</name>
  <attribute>*Memory*</attribute>
  <attribute mode="read">Verbose</attribute>
  <operation>gc</operation>
</mbean>
```

<name>,<attribute>,<operation>都支持通配符格式的字符，<attribute>　元素的属性 mode="read"表示当前 Mbean 是只读的。

(4) HTTP 请求方法限制

最后，访问可以通过 HTTP 请求方法进行限制，例如为了安全 Mbeans 只能通过 POST 请求访问，xml 文件定义如下：

```
<http>
  <method>post</method>
</http>
```

xml 安全策略文件 jmx-mule-access.xml 是放在项目 classpath 下面的，完整版 jmx-mule-access.xml 例子如下:

```
<?xml version="1.0" encoding="utf-8"?>
<restrict>
    <remote>
        <host>127.0.0.1</host>
    </remote>
    <http>
        <method>post</method>
    </http>
    <commands>
        <command>read</command>
    </commands>
    <allow>
        <mbean>
            <name>java.lang:type=Memory</name>
```

```
            <operation>gc</operation>
        </mbean>
    </allow>
    <deny>
        <mbean>
            <name>com.mchange.v2.c3p0:type=PooledDataSource,*</name>
            <attribute>properties</attribute>
        </mbean>
    </deny>
</restrict>
```

上面的 XML 安全策略文件定义了当前 JMX Mule Agent 的权限：只能通过本地访问，只能使用 POST 方式读取 Mbean 数据，可以操作 gc 命令提醒 JVM 释放内存，禁止访问 C3P0 的 Mbean。

所有安全策略的实现都是通过 restrictor 包下的类实现的，图 5-4 为安全策略实现类图。

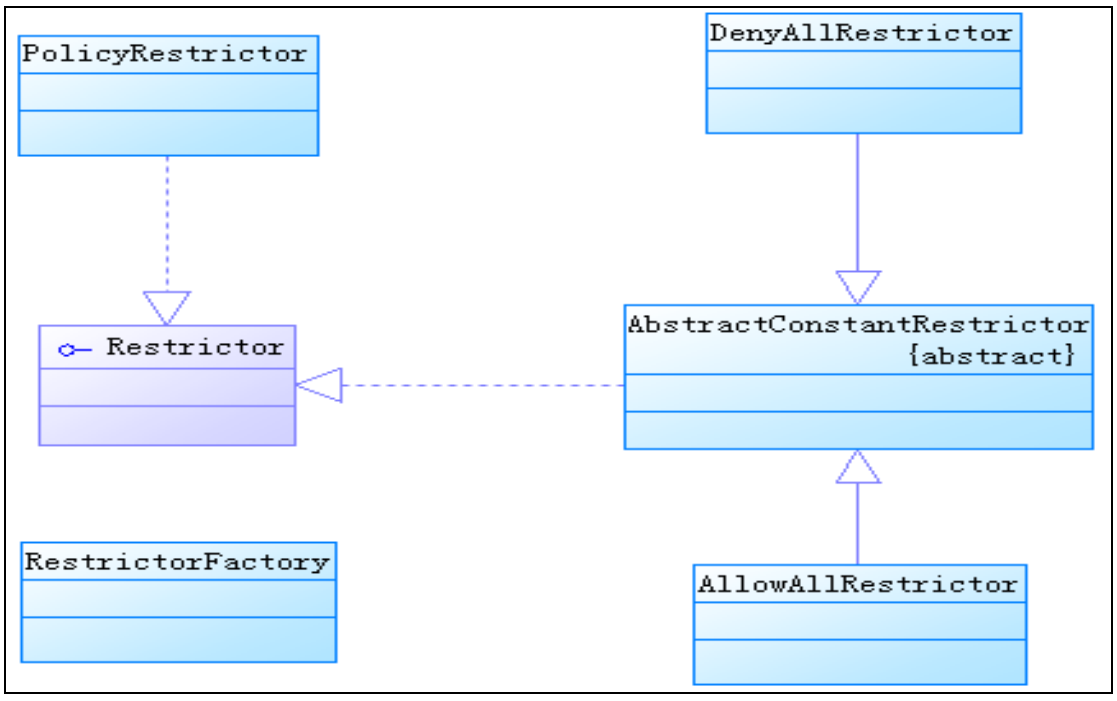


图 5-4 安全策略实现类图

在 Mule ESB 启动时，Agent 会读取 jmx-mule-access.xml 文件，如果文件不存在则所有的操作都是允许的，如果存在则读取文件内容进行相关权限控制。RestrictorFactory 为调用 PolicyRestrictor 获取 jmx-mule-access.xml 中值的工厂类，保持单例状态，主要代码如下：

```
public final class RestrictorFactory{
    private RestrictorFactory() {}
```

```
    public static PolicyRestrictor lookupPolicyRestrictor(String pLocation){}
}
```

PolicyRestrictor 实现 Restrictor 接口，根据策略文件中内容分别在 IP、特定 Mbean、HTTP 请求方式以及对 Mbean 的操作方式方面判断当前 Agent 访问权限，主要代码如下：

```
public class PolicyRestrictor implements Restrictor{
      private HttpMethodChecker httpChecker;
      private RequestTypeChecker requestTypeChecker;
      private NetworkChecker networkChecker;
      private MBeanAccessChecker mbeanAccessChecker;
      public boolean isHttpMethodAllowed(HttpMethod method){}
      public boolean isTypeAllowed(RequestType pType){}
      public boolean isRemoteAccessAllowed(String ... pHostOrAddress){}
      public boolean isAttributeReadAllowed(ObjectName pName, String
      pAttribute){}
     public boolean isAttributeWriteAllowed(ObjectName pName, String
     pAttribute) {}
     public boolean isOperationAllowed(ObjectName pName, String
     pOperation){}
}
```

AllowAllRestrictor 继承抽象类 AbstractConstantRestrictor，当没有安全策略定义文件时允许所有操作，DenyAllRestrictor 继承抽象类 AbstractConstantRestrictor，禁止所有访问，工厂类 AbstractConstantRestrictor 实现 Restrictor 接口，在创建的时候返回常数值。

5.5 结果与分析

程序分为两部分一个是注册在 Mule ESB 中的 JMX Agent，一个是部署在 Tomcat 的应用程序，Tomcat 中的应用程序需要通过 JMX Agent 访问 Mule ESB。因此，首先我们应该先要启动 Mule ESB，然后是启动 Tomcat 服务器。

图 5-5 显示的是 Mule ESB 启动图，从图中我们可以看出已经注册的 JMX Agent Triman Agent，我们可以通过 http://liang-PC:8899/triman 访问 Mule ESB 中注册好的 Mbean。

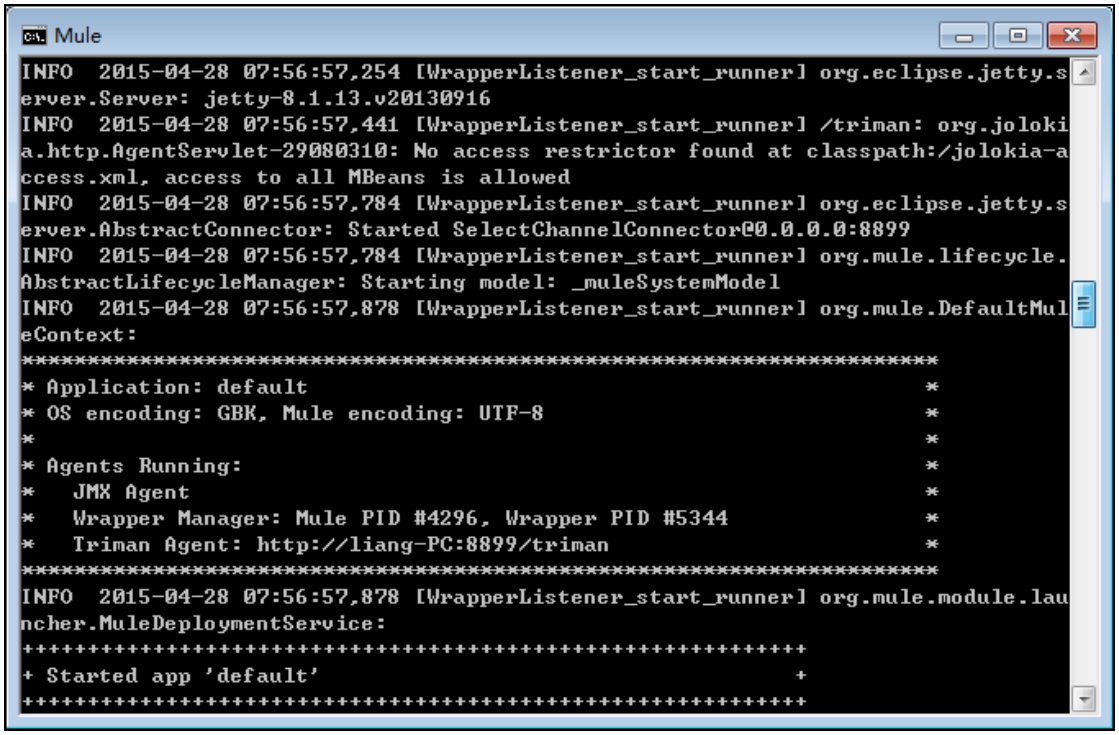


图 5-5 Mule 启动图

基于 JMX 的 Mule ESB 控制台可分为四个部分：服务器状态实时展示、应用的查看和管理、服务的查看和管理以及 ESB 运行日志的查看。下面的图将分别展示控制台的各个运行界面。

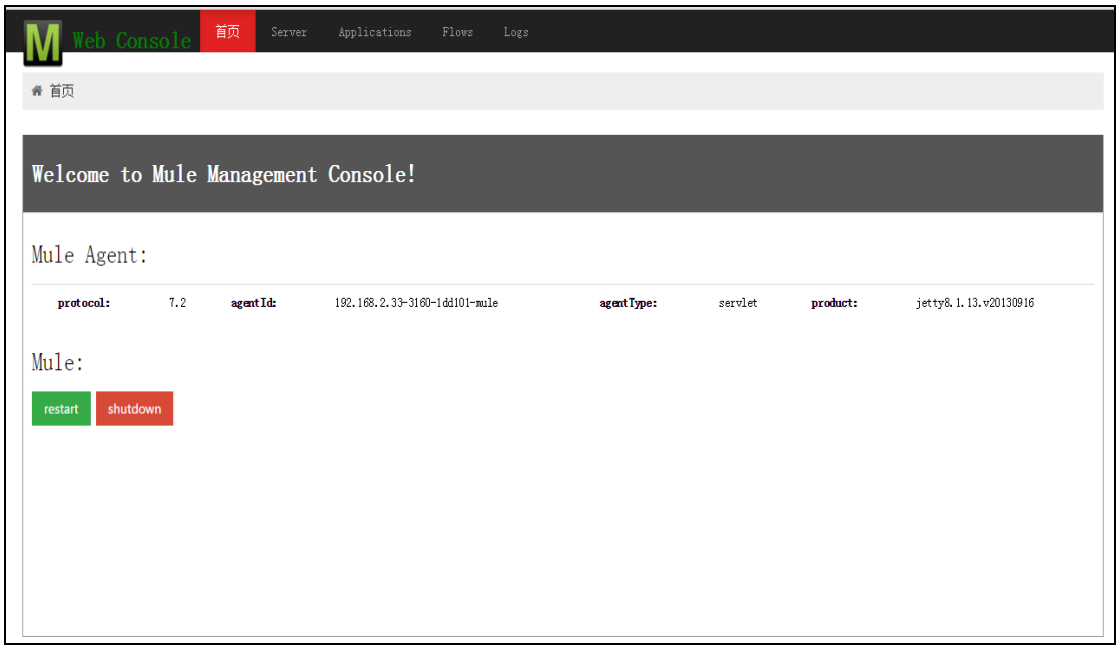


图 5-6 首页

图 5-6 首页，在这里可以查看 Mule Agent 基本信息以及 Mule ESB 启动状态，当需要关闭时，可以进行关闭，当出现应用部署失败等问题时，可以进行重启操作。

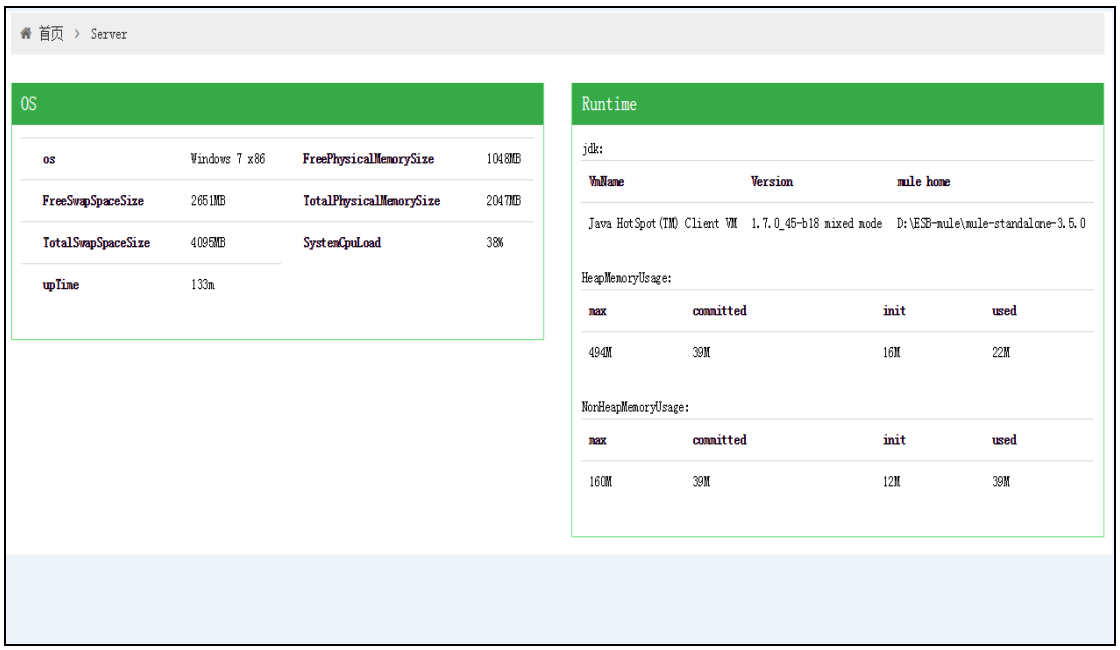
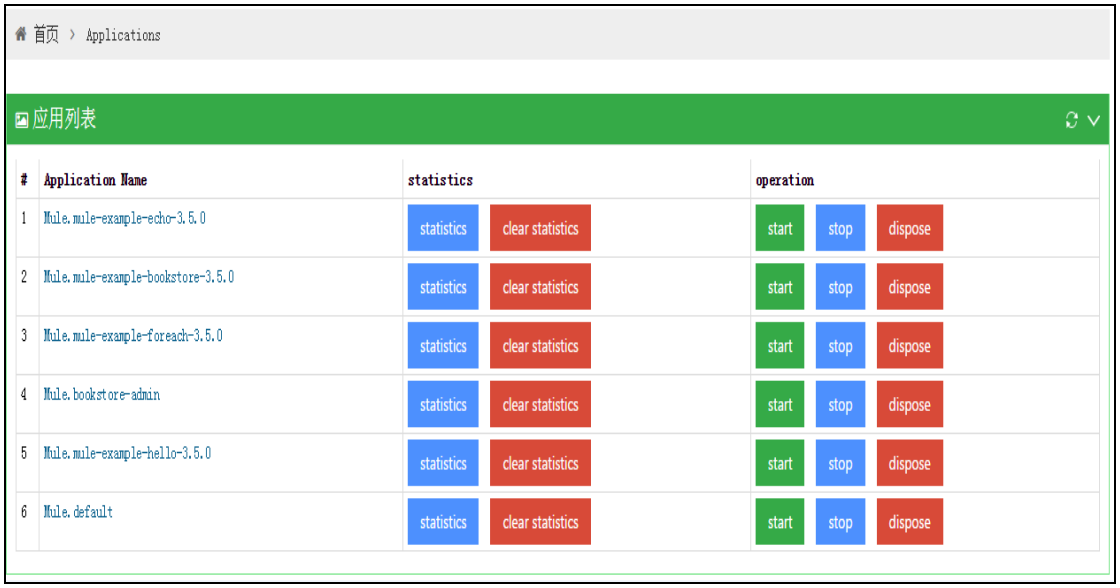


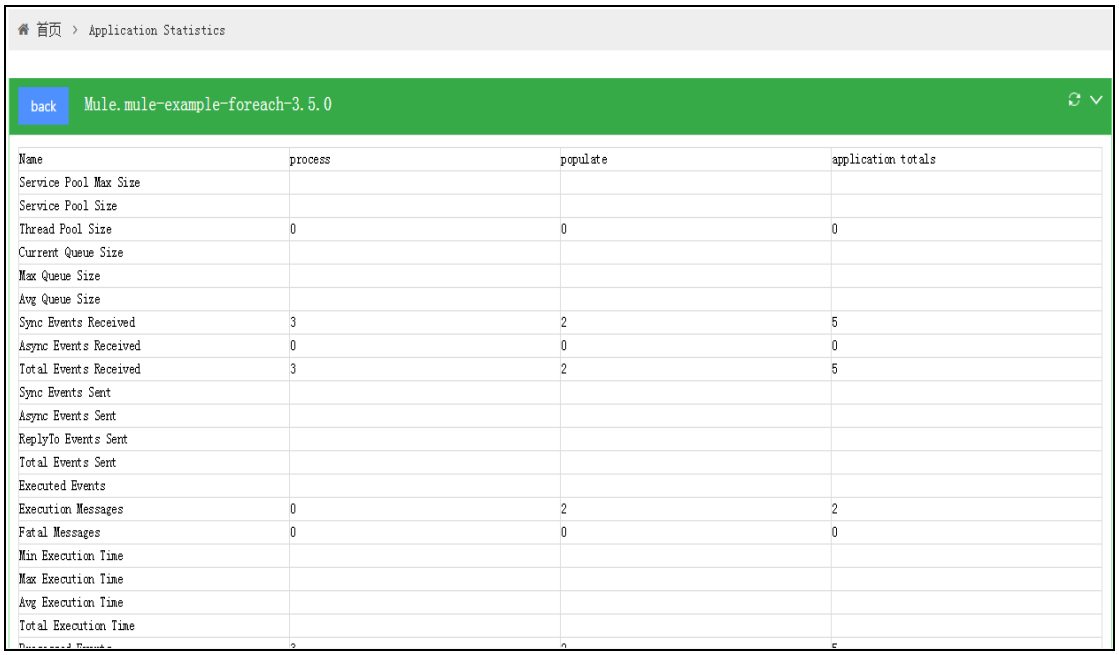
图 5-7 服务器监控图

图 5-7 服务器监控图，左边的表格显示的是操作系统相关信息，可以对系统运行负载进行查看，包括物理内存、虚拟内存使用情况，以及 CPU 使用率。右边的表格显示的主要是 java 虚拟机的运行状况，可以查看 JDK 的版本信息，堆内存、非堆内存的申请使用情况。



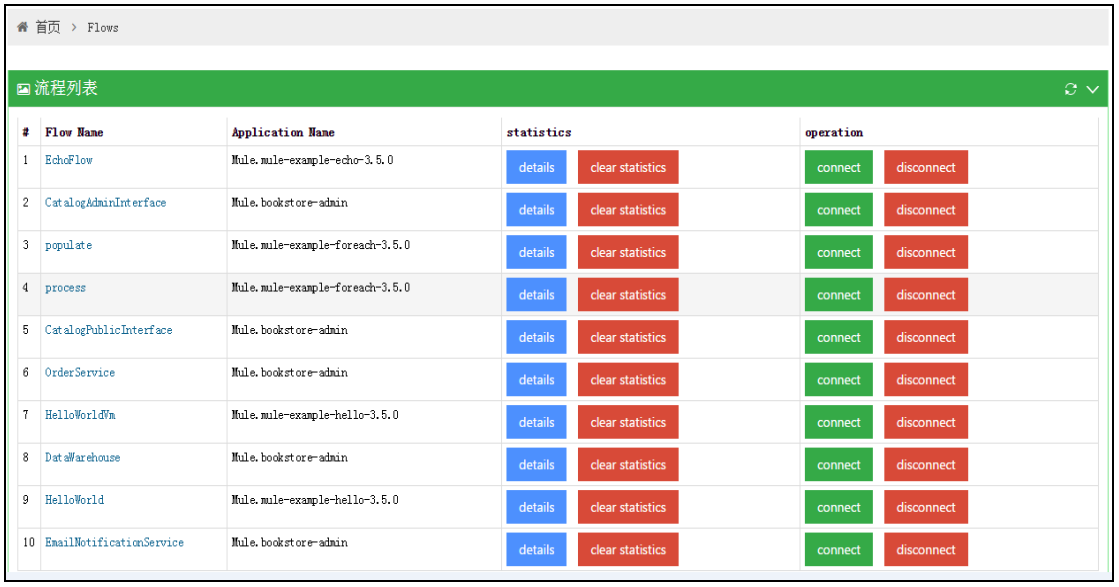
5-8 应用列表图

5-8 应用列表图，可以查看部署在 Mule ESB 中的应用，可以查看应用中服务使用统计如图 5-9 所示，可以清除应用中服务的使用统计数据。也可以停止应用对外提供服务，如果应用不想使用了，可以将应用下线，如果还想要该应用上线，可以重启服务器即可。



Name	process	populate	application totals
Service Pool Max Size			
Service Pool Size			
Thread Pool Size	0	0	0
Current Queue Size			
Max Queue Size			
Avg Queue Size			
Sync Events Received	3	2	5
Async Events Received	0	0	0
Total Events Received	3	2	5
Sync Events Sent			
Async Events Sent			
ReplyTo Events Sent			
Total Events Sent			
Executed Events			
Execution Messages	0	2	2
Fatal Messages	0	0	0
Min Execution Time			
Max Execution Time			
Avg Execution Time			
Total Execution Time			

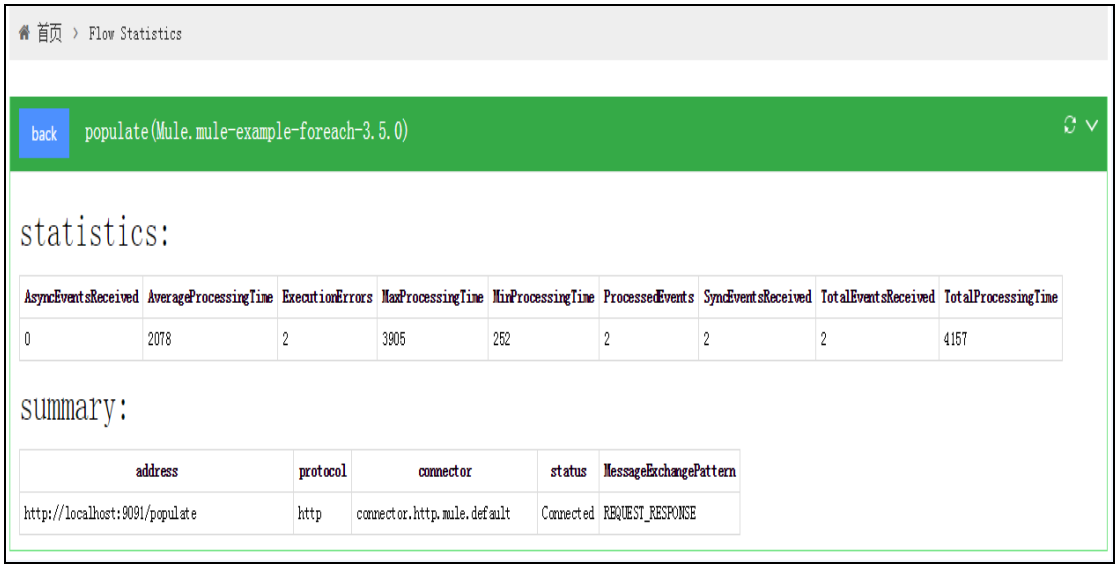
5-9 应用服务使用统计图



#	Flow Name	Application Name	statistics		operation	
1	EchoFlow	Mule.mule-example-echo-3.5.0	details	clear statistics	connect	disconnect
2	CatalogAdminInterface	Mule.bookstore-admin	details	clear statistics	connect	disconnect
3	populate	Mule.mule-example-foreach-3.5.0	details	clear statistics	connect	disconnect
4	process	Mule.mule-example-foreach-3.5.0	details	clear statistics	connect	disconnect
5	CatalogPublicInterface	Mule.bookstore-admin	details	clear statistics	connect	disconnect
6	OrderService	Mule.bookstore-admin	details	clear statistics	connect	disconnect
7	HelloWorldVm	Mule.mule-example-hello-3.5.0	details	clear statistics	connect	disconnect
8	DataWarehouse	Mule.bookstore-admin	details	clear statistics	connect	disconnect
9	HelloWorld	Mule.mule-example-hello-3.5.0	details	clear statistics	connect	disconnect
10	EmailNotificationService	Mule.bookstore-admin	details	clear statistics	connect	disconnect

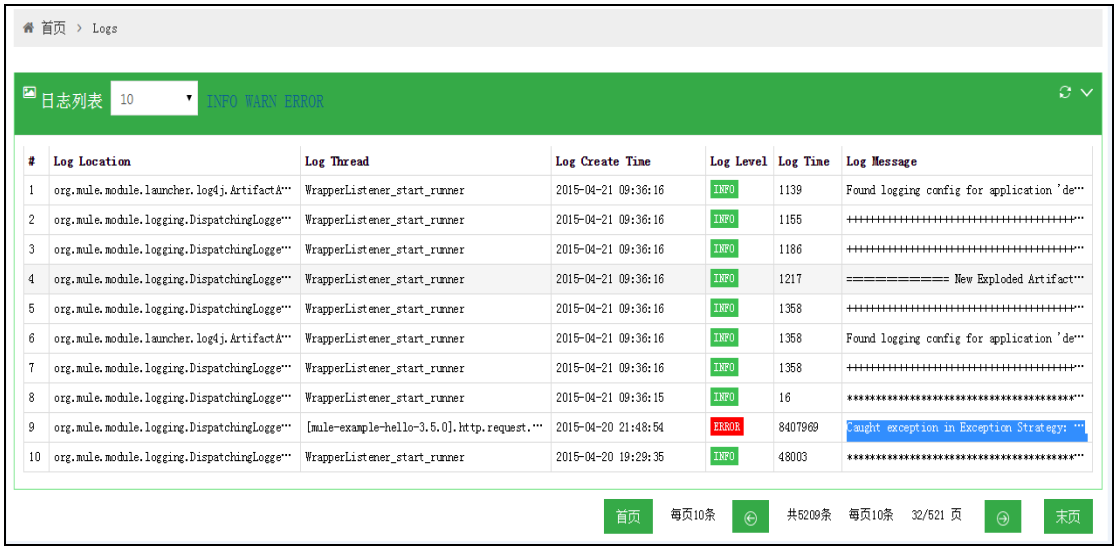
5-10 流程列表图

图 5-10 流程列表图显示的是 Mule ESB 中注册的对外提供的服务。你可以断开服务，也可以上线已断开的服务，对服务使用的统计数据可以清除，点击 details 按钮可以查看服务的统计数据以及服务的基本信息，包括地址、协议等，如下图 5-11 所示。



5-11 流程服务详情图

图 5-12 是 Mule ESB 的运行日志图，包含 Mule ESB 运行日志以及服务调用日志。



5-12 Mule ESB 运行日志图

5.6 本章小结

本章基于 ESB 的服务发现和管理的模块实现部分的重点章节。针对上一章节的详细设计阐述了实现 Web 服务发现和管理的系统的多个功能模块的方法。包括对 Log4j 日志处理中间件的使用，以及对出现的问题对源代码做出的改进；对 Mule Agent 的设计，以及针对 Mule Agent 访问 Mbean 的安全问题也做出了几个可行的解决办法。最后给出了 Web 服务发现和管理前端控制平台的演示截图和部分实例的运行界面。

第六章　结束语

6.1 总结

企业服务总线 ESB 是传统中间件技术与 XML、Web 服务等技术结合的产物。ESB 提供了网络中最基本的连接中枢，是构筑企业神经系统的必要元素。是实现 SOA 的有效途径。它是整合应用和服务的一个灵活的基础架构。服务总线位于 SOA 的中心，并通过减少接口的数量，大小和复杂度使得 SOA 更为强大。ESB 的出现改变了传统的软件架构，可以提供比传统中间件产品更为廉价的解决方案，同时它还可以消除不同应用之间的技术差异，让不同的应用服务器协调运作，实现了不同服务之间的通信与整合。

本文通过对目前业界主流的 ESB 系统和国内外 ESB 技术发展的现状进行了深入的技术研究和总结之后。得出许多关于通用 ESB 实践技术和框架设计方法的结论和研究成果，针对 目前主流开源 ESB 系统不提供管理控制台的问题，提出了自己的解决方案，基于开源软件 Mule ESB 系统的分析成果，目的是实现 ESB 可视化管理。利用最新技术 JMX 并最终实现了设计提出的功能模块。

6.2 展望

虽然基本功能已基本完成，但是限于笔者的知识储备，同时对 ESB 的研究还不够深入，还有许多地方需要改进。同时在实现主体功能后很多新的问题又出来了，下面是还存在的一些主要问题还需研究：

(1) 当前的服务控制台只是针对单机 Mule ESB 开发的，Mule ESB 是支持分布式部署的，下一步需要针对分布式环境对控制台进行改进优化。

(2) 当前开发只是针对 ESB 开发的 JMX Agent，或许可以让控制台支持监控、管理更多开源软件，目前很多开源软件内部都实现了 JMX 规范，例如 Servlet 容器 Tomcat、数据库连接池 C3P0 等。

(3) 前台展示数据需要以更加直观的方式展现，可以利用动态图展示数据，目前可以选的方案有 Flash，几乎兼容所有的浏览器；HTML5 目前发展趋势，可以很方便的在浏览器绘制各种图表。

参考文献

[1] Oracle JMX官方网站. http://docs.oracle.com/javase/tutorial/jmx/index.html.

[2] Mule官方网站. http://www.mulesoft.org/.

[3] ESB介绍. http://baike.baidu.com/view/1224042.htm?fr=aladdin.

[4] 面向服务的体系结构. http://baike.baidu.com/subview/21305/5033544.htm.

[5] 黄安安等. 基于ESB的企业应用集成研究[J]. 微计算机应用, 2007, (9).

[6] 丁昭华, 李建华. 企业服务总线在企业应用集成中的研究与应用[J]. 计算机应用与软件, 2008 (9).

[7] David Dossot, John D'Emic, Victor Romero. Mule in Action. Manning Publications, 2014.

[8] Ryan Carter. Getting Started with Mule Cloud Connect. Manning Publications, 2012(11).

[9] 王益祥, 薛霄等. 基于SOA架构的企业应用集成技术研究[J]. 微计算机信息, 2010, 5-3:64-65.

[10] 胡圣明, 褚华主编. 软件设计师教程(第二版). 清华大学出版, 2009(08).

[11] 林怀恭. 基于ESB的共享数据中心的研究与实现. 计算机应用与软件, 2010(05).

[12] (美)康诺利著, 宁洪等译. 数据库系统设计、实现与管理(第三版). 电子工业出版社, 2004(01).

[13] 付更丽, 曹宝香. SOA-SSH分层架构的设计与应用[J]. 计算机技术与发展, 2010, 20(1):74-77.

[14] 李振, 韩国强. SOA在大型数据采集平台中的应用[J]. 微计算机信息, 2009, 25(6-1):89-91.

[15] 彭政, 聂瑞华, 李飞. 基于ServiceMix的SOA架构的研究与实现[J]. 计算机工程与科学, 2009, 31(4):153-158.

[16] 刘贤梅, 刘茜, 徐锋. 基于SOA的企业应用集成模型的研究[J]. 计算机工程与设计, 2009, 30(16):3790-3793.

[17] 伍轶明, 宋伟奇, 郭源源. 基于SOA的社区信息综合服务平台设计研究[J]. 广西工学院学报, 2009, 20(3):60-65.

[18] 黄序鑫, 聂瑞华, 罗辉琼等. 基于SOA的数据同步技术研究与实现[J]. 计算机工程与设计, 2009, 30(14):3338-3340.

[19] 温欣. 基于SOA的物流平台设计与实现[J]. 科技资讯, 2009, 6(16):243-244.

[20] 张启文, 徐琪. 基于SOA和ESB的供应链快速响应系统集成研究[J]. 计算机应用, 2009, 29(9):2523-2526.

[21] 林怀恭, 聂瑞华, 罗辉琼. 基于SOA架构的服务集成技术的研究[J].计算机技术与发展, 2009, 19(7):141-148.

[22] 张乾. 论SOA在系统集成中的应用[J]. 中国教育信息化, 2009, 17:81-84.

[23] 秦鼎, 贺智明. 基于ESB的服务中介框架的设计与实现[J].江西理工大学学报, 2009, 30(3): 49-52.

[24] 边小凡, 代艳红, 马兵会. 基于ESB的企业内部构件与Web服务的集成[J]. 计算机工程与设计, 2009, 30(4):918-924.

[25] 黄彦琦, 王志朋, 崔勇杰. 基于ESB的面向构件的开发模式研究[J]. 软件导刊, 2009, 8(8):10-12.

[26] 冯培培, 王辉. 基于ESB技术的系统集成框架的研究[J]. 通信技术, 2010, 01(43):195-197.

[27] 孔样瑞, 郑洪源. 基于企业服务总线的业务集成方法[J]. 计算机工程, 2009, 35(16):280-28.

致 谢

首先，我要感谢我敬爱的导师——燕彩蓉老师。在研究生两年的学习生活中，燕老师以她严谨的学风，深刻的思想，敏捷的思维，坚实的理论功底，以及高尚的品格，为我树立了人生的楷模，使我受益匪浅。更重要的是她不断探索的科学精神和一丝不苟、兢兢业业的工作风格是我永远学习的榜样。在我为学业和生活感到困难和迷惑的时候，是她给予了我前进的动力和奋斗的方向。在两年的研究生生涯中，很幸运得到老师的辛勤培养和悉心指导。此刻，似乎无法用言语来形容我的感激之情，惟愿师生情谊一生延续。

其次感谢实验室黄老师，很高兴能和您在同一个实验室。感谢您给我们提供实践项目的机会，使我们熟能生巧，锻炼自己。您平时严格要求我们，闲暇时也会给我们组织各种活动，使我们有了一个丰富多彩的研究生生活。

最后一年我们是在企业中实习度过的，感谢企业导师王总以及指导老师王战英老师。在企业当中我们得到了锻炼自己的机会。王老师技术功底扎实，工作态度认真，我非常钦佩，您是我以后工作学习的榜样。

其次，衷心感谢答辩委员会的各位老师们，你们百忙之中抽出时间来审阅论文，为我指点迷津。同时感谢我培育了我两年的大连海事大学，在这里，我度过两年美好的研究生时光，学习到了我将受益终生的知识，也学会了如何做人。

在两年的学习生活期间，得到了师姐师兄、师弟师妹、室友以及同班同学们的关心和帮助，在此表示深深的感谢，愿同窗之间的友谊永远长存，感谢有你。

最后，向所有给予我关怀和帮助的老师和同学们表示最衷心的感谢。