船舶综合应用服务平台设计方案

西北工业大学

[1. 系统概述 4](#_Toc393763606)

[1.1. 项目背景 4](#_Toc1587809635)

[1.2. 船舶综合应用服务平台 4](#_Toc1684737823)

[1.3. 面向监控的服务平台 6](#_Toc816705466)

[2. 引用文档 7](#_Toc1800779085)

[3. 需求概述 7](#_Toc1207044424)

[3.1. 船舶综合应用服务平台的总体需求 7](#_Toc1665104606)

[3.1.1. 集成平台一体化 7](#_Toc1553708985)

[3.1.2. 集成应用一体化 7](#_Toc1933247022)

[3.1.3. 服务管理一体化 7](#_Toc655119644)

[3.1.4. 安全管控一体化 7](#_Toc447198539)

[3.2. 船舶综合应用服务平台的具体需求 8](#_Toc2020564120)

[4. 总体设计 8](#_Toc1462254829)

[4.1. 总体架构 9](#_Toc314054735)

[4.1.1. 用户管理 9](#_Toc1950610466)

[4.1.2. 服务集成 9](#_Toc424746960)

[4.1.3. 外部工具链 11](#_Toc486514092)

[4.1.4. 主要业务流程 11](#_Toc1372100115)

[4.2. 部署方案 12](#_Toc1207231319)

[4.2.1. 系统开发态部署方案 12](#_Toc511281577)

[4.2.2. 系统运行态部署方案 14](#_Toc1027392992)

[4.3. 外部接口 14](#_Toc1625494664)

[4.3.1. 接口标识和接口图 14](#_Toc1549344361)

[4.3.2. EGI接口 16](#_Toc1591455452)

[4.3.3. CPI接口 16](#_Toc682958379)

[4.3.4. GEI接口 16](#_Toc181382638)

[4.3.5. GPI接口 16](#_Toc1218701773)

[4.3.6. TFI接口 16](#_Toc21673725)

[5. 模块设计 16](#_Toc1345559732)

[5.1. 用户管理 16](#_Toc1819612814)

[5.1.1. 用户管理 17](#_Toc2065431618)

[5.1.2. 角色管理 18](#_Toc1783533618)

[5.1.3. 权限管理 18](#_Toc1272772900)

[5.1.4. 令牌管理 19](#_Toc409522533)

[5.2. 服务集成 19](#_Toc160123496)

[5.2.1. 服务管理 19](#_Toc398587581)

[5.2.2. 服务网关 29](#_Toc1059978874)

[5.2.3. 服务监控 34](#_Toc1688083453)

[5.3. 外部工具链 40](#_Toc1212134054)

[5.3.1. 组件开发 40](#_Toc1307170136)

[5.3.2. 组件库管理 43](#_Toc850766942)

[5.3.2.2.1. 执行流程 45](#_Toc893872468)

[5.3.3. 组件配置与部署 48](#_Toc1666458911)

[参考文献 51](#_Toc693193003)

# 系统概述

## 项目背景

随着网络中心战、协同作战等概念的逐步实现，我海军C4ISR系统的发展呈现出“陆海空天战场一体化”、“作战指挥扁平化”、“从传感器到武器直接综合”的趋势。为了支持海军以网络为中心的作战，实现我军现代化的跨越式发展和武器装备信息化建设要求，军用船舶中传感器、指控、武器、导航、情报、通信、电子对抗等各种作战装备间的数据交换与信息共享需求越来越突出，信息量也急剧增大。除作战系统外，军用船舶中动力、电站、损管、船舶驾驶等平台设备自动化程度也越来越高，所需状态和控制信息也日趋繁杂。这些变化都要求军用船舶各种信息的传输和相互交流能快速、准确、方便、及时，从而提出军用船舶信息化系统平台发展要综合化、数字化、一体化。

面向服务的体系结构(SOA)是一个组件模型，将服务作为组件，为用户提供定义良好的服务集合[1]。SOA 的核心是服务和共享，基于开放的标准和协议，具有松散耦合、支持应用系统高效整合和业务流程随需应变的特点，为主动信息推送系统的设计开发提供了新的理念和实现途径。SOA 通过网络对松耦合、粗粒度服务进行分布式部署，服务调用者根据不同业务需求进行重组和使用，通过业务驱动服务，服务驱动技术，SOA 让系统变得更有弹性。在SOA的主要实现方式——Web服务技术方面，目前主流的方式有两种，分别是SOAP与REST。SOAP最初是为开发轻量级服务而设计的，但随后SOAP的发展渐渐远离了其初衷，随着协议的完善与扩充，SOAP在技术成熟性和安全性等方面得到了提升，同时也使其愈发沉重，逐渐演变成一个庞大的协议体系。对于指控系统中大数据量的消息传输，SOAP产品的性能开销很大。相比而言，RESTful架构充分体现了互联网的思想，具有结构清晰、通信简洁、通用互操作性强以及可扩展等优势，在各大互联网公司中被广泛使用。

船舶综合应用服务平台是利用服务化的技术，将推进系统、电站及电能管理系统、辅机系统、损管系统、三防系统、综合舰桥系统、综合状态评估系统、实船在线训练系统、全船维护保障系统、电视监视系统等相对独立的分系统纳入到统一的网络中，实现信息集成，形成以信息集成为核心，集监测、控制、管理于一体的网络化、信息化和智能化的综合大系统。其目的旨在为日趋复杂的舰船系统和设备提供高自动化、高可靠性的集成监控平台，并提供一种开放式的、高可扩展性的具备功能动态配置、灵活重组特性的信息集成体系架构，实现功能与设备的分离、信息采集与信息使用的分离、数据与应用的分离，从而解除系统功能与设备紧耦合的绑定关系，消除当前日益严重的舰船系统功能扩展与总体优化间的矛盾，达到减员增效的目标。

## 船舶综合应用服务平台

船舶平台综合管理系统（以下简称为平台）通过软件、传感器、设备和网络的集成，综合船舶平台管理的操作与控制功能，对遍布全舰的各种平台管理系统进行监测与控制，实现舰艇平台管理的信息化和数字化，并为未来的网络中心战提供平台管理资源管理支撑。其体系结构如图1所示：

该体系结构通过将管理系统划分多个层面的网络层次，屏蔽不同的技术细节，实现平台系统的网络化、信息化的有效集成。图中的基础设施层，管理网络和由监控网关接入的各种控制系统组成的设备网络层沿用现行的技术体制和装备，上层的由船舶综合应用服务平台、服务层和应用层组成的信息网络层才是目前平台系统网络化、信息化需要关注的部分。

服务层中的业务服务和监控组件由服务和组件开发人员开发并在船舶综合应用服务平台上集成，而基础服务是由船舶综合应用服务平台提供的，但独立于该平台部署、运行，并与业务服务一样接受船舶综合应用服务平台的管理与监控。管理系统在应用层上表现为由若干子系统组成，如动力监控系统、电力监控系统、损管监控系统、全舰装备保障管理系统、架控系统等。每一个子系统根据其是以监控功能为主还是管理功能为主，分别采用C/S架构和B/S架构进行搭建。

图1 船舶平台综合管理系统体系结构

C/S架构的子系统基于分布式组件技术，分为明显的客户端和服务端。客户端（台位）由部署并运行在界面插件容器中的各种监控插件协同实现，属于监控类子系统的业务展现层。每一个监控插件都有一个显示界面，同一个容器中的多个监控插件所对应的显示界面可以灵活组合，以形成台位所需要的监控界面。服务端属于监控类子系统的业务逻辑层，由部署并运行在业务组件容器中的监控组件和各种服务协同实现。服务包括基础服务和业务服务，有别于监控组件，它们一般均体现为REST服务形式或微服务形式。监控插件和监控组件在专业概念上均为分布式组件。监控插件可以调用监控组件的接口或调用其它服务接口实现它所需要的业务逻辑。一个监控组件可以调用其它监控组件的接口或调用其它服务接口实现自身所需要的业务逻辑。监控插件与监控插件之间、监控插件与监控组件之间、监控组件与监控组件之间均支持相互的数据发布与订阅功能，以高效实现监控类子系统的数据驱动与事件驱动特性。

而管理类子系统采用B/S架构，可以具有近乎零客户端部署与维护能力，并利用标准、开放的SOA（面向服务的体系结构）相关协议和技术，使系统更具标准化和灵活的集成能力。

管理类子系统采用B/S架构，当然，其客户端展现功能统一由浏览器支持并实现。各服务端业务逻辑采用SOA服务封装。建议统一采用微服务理念实现各种服务。如前所述，这些服务可以被监控插件和监控组件调用，但它们不调用监控插件和监控组件提供的接口，也不支持数据发布/订阅机制。

## 面向监控的服务平台

根据合同约定，面向监控的服务平台提供一个面向监控的服务开发、集成与运行环境，实现服务的发布与订阅、服务间协作、服务管理等，支持分布式监控应用的服务化、构件化集成及高级应用快速实现，即为船舶平台综合管理系统中的船舶综合应用服务平台，为行文简洁，后文就简称为服务平台。

该服务平台结合微服务的思想，提供服务集成框架，使现有的系统应用能够以服务的形式进行集成，并支持新的服务的快速开发与部署。总的来说，平台应满足以下功能性和非功能性要求：

1. 平台应实现服务的注册、发现、调用、监控和快速部署。

2. 平台应满足可用性、安全性等非功能性的要求。

因此，研究内容主要如下：

1. 研究SOA架构中的服务的集成方式，具体考虑通过企业服务总线或服务网关等集成方式，研究服务集成中间件的架构和面向服务所提供的接口。

2. 研究在服务集成的中间件上加入数据发布及订阅的机制。

3. 研究服务的可用性和安全性的实现方式。

4. 实现所给平台所需提供的公共服务。

5. 研究服务通信机制的设计。

6. 研究服务部署方式的设计。

7．研究并给出面向SOA的服务平台的软件架构。

8. 对服务平台的体系结构进行研究，明确服务开发及部署的流程。

9. 对服务模型进行研究与建模，并设计服务模型描述语言（SDL）用于对模型的定义。

10. 设计SDL到JSON的映射规则以及服务模型代码框架自动生成技术。

# 引用文档

1. 面向监控的状态对象平台研究技术开发合同
2. 综合应用服务平台技术研究报告

# 需求概述

## 船舶综合应用服务平台的总体需求

### 集成平台一体化

目前国内外厂商受商业化销售策略的影响，其服务集成平台相关核心部件包括企业服务总线、流程管理、服务治理等均自成体系，产品本身存在功能单一、架构各异等诸多缺点。因此，船舶综合应用服务平台应遵循一体化架构设计理念来实现，完全覆盖舰船服务总线、服务生命周期管理、服务治理、服务库管理等功能。

### 集成应用一体化

船舶综合应用服务平台作为船舶所有集成服务的基础交互平台，按照船舶集成的需求和技术规范在平台上实现信息交互，以达到SOA集成的松散耦合、信息资源重用和共享的目标。同时，平台应支持服务开发，形成以面向服务架构为核心、在平台上通过服务组合和编排实现信息集成需求的建设模式，最终搭建以集成、共享、协同为标志的一体化应用体系。

### 服务管理一体化

在面向服务的架构模式下，服务是信息的载体，船舶上的所有服务必须部署在统一的信息集成平台上，船舶综合应用服务平台应提供有效的服务治理能力，为服务的全生命周期管理提供技术支持，完成SOA治理的关键目标。平台应具备服务识别、服务设计、服务开发、服务部署、服务发布等功能，保障舰船不断积累高度复用的服务资产。

船舶综合应用服务平台应提供统一的管控流程和机制，包括服务的自动部署、服务目录的自动同步、服务发现和路由等。同时，针对服务的运行情况，平台应提供服务运行监控、日志审计等功能，以实现对SOA治理的管理支撑和技术支撑，提升各应用系统和舰载信息化建设的成熟度。

### 安全管控一体化

传统方式下安全服务通常定制于各类应用系统中，实施安全服务的硬件、软件为应用系统专用。随着船舶计算环境标准化、综合化、一体化计算框架的构建，互操作性是船舶计算环境的基础，服务用户与服务提供者之间、服务提供者与安全服务之间的接口必须一致，这就要求安全服务要遵循船舶服务平台服务标准和开发要求，为各类应用系统提供标准化的安全服务。

船舶综合应用服务平台应具有统一的身份标识、授权和密码管理设施。在管理设施的统一管理下，各安全服务组件与其它服务组件共同配合建立安全的集成环境。

## 船舶综合应用服务平台的具体需求

根据船舶平台综合管理系统的特点和船舶综合应用服务平台（以下简称平台）的总体需求，平台的功能需求如下：

（1）支持界面插件和后台组件的开发和集成

为了充分利用客户端机器的本地计算资源和界面渲染能力，一般在客户端采用插件(组件)的形式对用户提供服务以及图形界面，以实现快速的业务处理能力和灵活的界面组合能力，而在某些实时性要求较高的应用中该类插件会和运行在服务器端的后台组件采用二进制的协议进行通信，而不是使用传统服务集成中所采用的HTTP协议，因此平台应支持界面插件以及后台组件的开发和集成。

（2）支持服务的集成

平台应支持REST接口风格的服务间的相互调用，平台也应支持上述界面插件和后台组件对服务的调用，以充分的利用服务化的优势，简化界面插件与后台组件的开发、管理和维护。

（3）支持服务和组件的管理

平台应支持服务的管理，包括服务的注册与发现，服务的相关信息的存储与管理，服务生命周期管理等。平台也需支持各类组件的管理，包括组件库的管理，组件的生命周期管理等。

（4）支持数据的发布与订阅

平台应支持监控数据的发布与订阅。由于监控类子系统由插件构成，故应提供插件间的数据发布订阅机制，以高效实现监控类子系统的数据驱动与事件驱动特性。

（5）支持平台上服务和组件的状态监控

平台应支持对平台上集成的服务和组件的运行状态进行监控，包括：

* 对在平台上的各类操作及对服务的调用记录日志以便以后进行审计和管理
* 对异常信息及事件进行监控，方便用户实时准确地掌握了解系统运行情况
* 对服务和组件的运行状态进行监控，方面用户对服务和组件进行集中式的管理
* 对服务各种运行的指标进行监控统计，方便用户进行资源配置，从而增加系统的可用性

（6）提供公共服务

平台应提供一组公共服务，包括安全，身份认证，时间同步，事件管理等。应对舰船基础资源（包括设备、数据等）进行服务化封装，以支持应用的持续创新，并减少不同服务供应商的重复性开发工作量。

（7）支持服务的组合与编排

平台应提供将细粒度的服务组合成更大粒度服务的工具，并实现将组合后的服务和原子服务一并进行管理和监控，对调用方来说，平台上集成的组合后的服务应和原子服务应没有区别。

# 总体设计

## 总体架构

船舶综合应用服务平台的功能模块划分如图2所示，由服务集成、外部工具链和基础数据库组成。

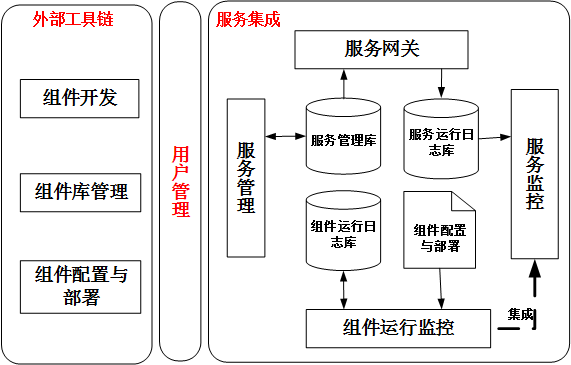


图2 船舶综合应用服务平台的功能模块

### 用户管理

对平台涉及到的用户进行增、删、改、查，这些用户涉及到的角色包括系统管理人员、服务提供（开发）人员、组件开发人员、服务管理人员、服务编排人员、服务集成人员、组件配置与部署人员，等等。

### 服务集成

服务集成部分用于支持服务和组件的管理以及运行支持与监控，具体包括以下功能：

#### 服务管理

这里的服务包括基础服务和业务服务，服务管理包括的功能有：

（1）服务注册/注销：本平台并不提供服务开发平台，服务开发人员（服务提供者）可以利用任何平台和编程语言开发服务，只要其遵循REST或微服务规范即可。开发并测试好的服务完成部署后，需借助本功能向平台注册，注册信息保存于服务管理库中，包括服务的分类信息和描述服务的各种元数据。当然，服务注销是服务注册的反操作，它从服务管理库中删除服务的分类与描述信息。

（2）服务发布/撤回：服务管理人员或服务提供者利用此功能将已注册的服务向外界发布，使其对外部可见。服务撤回是服务发布的反操作。服务管理人员是服务管理系统的超级管理员。

（3）服务查询：系统管理人员可以查询注册和发布的所有服务，其他人员利用此服务只能查询平台已发布的所有服务。查询结果主要是服务的描述信息（元数据）和服务依赖关系。

（4）服务订购与授权：服务集成人员可以对平台中所有已发布的服务进行选择并订购（这里的订购关系指的是服务与人的关系而不是服务与调用服务的软件的关系），服务管理人员对订购行为进行审核，若审核通过，则给予订购者以服务授权。本功能当然也包含取消服务订购与授权的操作。

（5）服务描述文件管理：服务描述文件（SDL）进行管理包含服务描述文件的查询获取、添加、删除、修改等操作。服务消费者可以通过被调用服务的描述信息，从而解析服务调用所需的参数信息。

服务管理的结果存储在服务管理库中。

#### 服务网关

服务网关依据服务管理所产生的信息，对所部署并发布的服务的运行提供支持，所有对所发布的服务的访问均须经过服务网关过滤（根据授权和发布订阅关系）和转接（相当于代理）。主要功能包括：

（1）服务访问控制：依据服务管理所确定的服务订购与授权信息，对每一个发布的服务的接口调用进行权限核查与控制。

（2）服务映射（转接）：依据服务管理所确定的服务订购关系，对通过权限审核的服务访问进行访问转接，即将对所发布的（外部可见的）服务接口的调用，转换为对后台相应服务（一般与外部可见的服务具有不同的接口URL）的调用，并截获内部服务调用结果转发给服务请求者。

（3）负载均衡：如果后台被调用服务启动了多个实例，则在服务映射时，可根据服务运行负载情况进行负载均衡，同时保持服务会话的完整性，且后台服务应该是无状态的。

（4）服务缓存：这是提升服务性能的机制之一。网关依据一定的原则，对后台服务调用的结果进行缓存，对后续到达的相同访问请求，直接利用缓存结果进行响应。

（5）熔断处理：这是处理服务故障并借以提高系统性能的机制。当对后台服务的访问结果进行统计分析后，根据一定的阈值，阻断对它的进一步访问，而直接代理它向服务请求者返回服务不可用信息。

（6）服务运行日志记录：服务网关要负责记录服务的运行情况，包括服务的请求与响应情况、服务启停与恢复情况，将这些信息记录到服务运行日志库中，供服务监控功能使用。

（7）服务控制：与服务监控功能进行协作，实现对服务的运行状态的控制，包括服务的启停与恢复控制。

#### 服务监控

对服务运行日志信息进行统计分析；与服务网关协作，实现对服务的启停与恢复等操作；调用组件运行监控模块对监控插件与监控组件的运行情况进行监控。

#### 组件运行监控

该模块对运行在系统中的组件（监控插件与监控组件）进行运行支持与监控。该项功能依赖组件配置与部署文件（由外部工具链中的组件配置与部署模块产生、提供）进行组件运行状态的集中监控，由组件运行日志库记录组件运行状态。本模块以服务方式与服务监控模块进行集成，并由服务监控模块集中提供组件监控界面，以便通过一个统一的服务入口对整个系统的运行状态进行统一监控。主要包括组件生命周期管理和组件运行状态查询

### 外部工具链

一组相对独立的工具，它们不直接参与系统的运行与监控，包括组件开发工具、组件库管理工具和组件配置与部署工具。

#### 组件开发工具

包括借助于CDL进行组件建模、组件框架代码生成、业务逻辑代码的手工实现、到Visual Studio等集成开发环境中进行构建以生成组件运行代码。通过CDL编译生成的组件代码框架支持对普通服务的调用。

#### 组件库管理工具

实现（非运行态）组件的管理。利用组件开发工具所实现的组件应纳入组件库管理，组件配置与部署工具应从组件库中获得需要的组件信息进行配置和部署。

#### 组件配置与部署工具

该工具用于支持监控子系统规划人员指定组件间的交互关系，配置监控插件中UI界面的显示位置，以实现监控台位界面的灵活组装。产生的结果是组件配置与部署文件。

### 主要业务流程

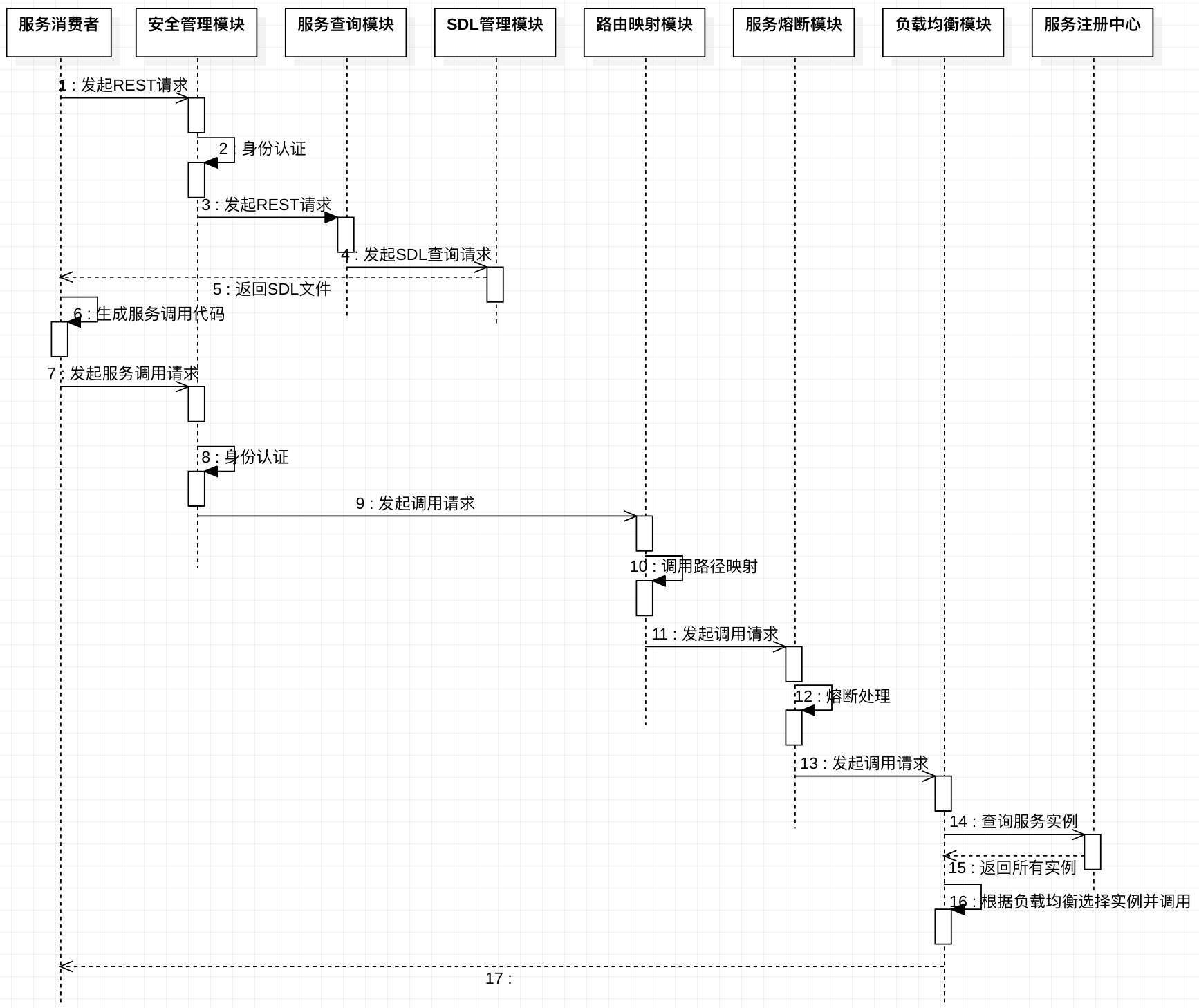


图3 服务调用的一般流程

服务调用的一般流程如下：

1、生成服务调用代码框架

（1）服务消费者向服务网关中的安全管理模块发起调用服务的REST请求，请求参数包括服务名称（服务的唯一标识）；

（2）由安全管理模块对调用者进行认证，检查服务调用者是否有权限调用该服务，认证通过则进入（3），否则返回错误信息；

（3）由服务网关调用服务管理中的SDL管理模块（服务描述文件管理模块），返回服务描述文件；

（4）解析服务描述文件，生成相应的服务调用代码。

2、服务调用

（1）服务消费者向服务网关中的安全管理模块发起调用请求，安全管理模块对调用者进行认证，检查服务调用者是否有权限调用该服务，认证通过则进入（2），否则返回错误信息；

（2）服务网关收到调用请求后会进行路由映射和熔断处理，处理完毕后将请求交由服务网关的负载均衡模块；

（3）负载均衡模块根据负载均衡策略在服务注册中心上查询并选择对应的服务实例进行调用；

（4）负载均衡模块调用完毕后将结果返回至服务消费者。

## 部署方案

部署方案分为系统开发态部署方案和系统运行态部署方案。

### 系统开发态部署方案

平台采用分布式部署方案，在局域网内运行，实施时要保证网络环境畅通。平台的逻辑部署图如4所示。基于平台性能要求以及可扩展性的设计原则，系统主要分为服务网关、服务注册中心、监控中心、数据库系统、服务运行容器和服务开发工具等逻辑节点。各节点可以部署至一台或多台服务器，各节点间通过网络进行通信。

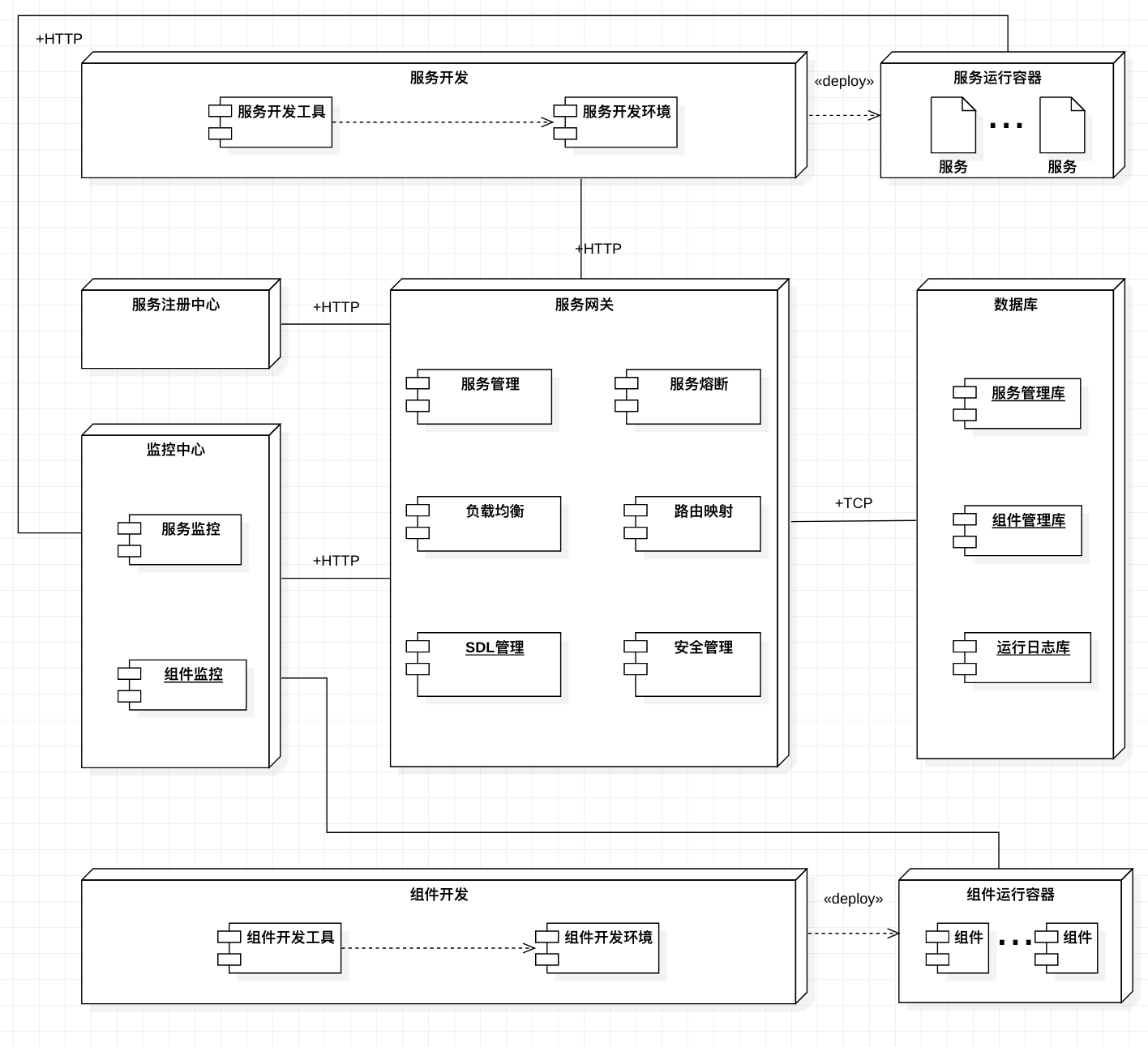


图4 系统开发态逻辑部署图

1. 服务网关。主要包括服务管理、服务熔断、负载均衡、路由映射、SDL管理、安全管理等逻辑组件，主要对外提供了统一的服务接口。与服务注册中心、监控中心和服务开发工具通过HTTP协议进行通信，并与数据库系统通过TCP协议进行通信。
2. 监控中心。包括服务监控和组件监控等逻辑组件，与服务运行容器和组件运行容器进行通信来监控服务和组件的运行状态，并与服务网关通信对外提供监控状态的查询。
3. 服务注册中心。主要提供服务的注册与发现功能，与服务网关通过HTTP协议进行通信，可独立部署，也可与服务网关部署至同一台物理设备。
4. 数据库系统。主要包括服务管理库、组件管理库和运行日志库等逻辑组件，是一个为平台提供数据的软件系统，通过TCP协议与服务网关进行通信。
5. 服务开发工具。主要是服务开发人员所使用的开发工具，开发人员可根据需要进行选用。
6. 服务运行容器。是指服务部署时所需的中间件服务器，可独立部署。
7. 组件开发工具。是组件开发人员借助于CDL进行组件建模、组件框架代码生成等的集成开发环境。
8. 组件运行容器。为组件的运行提供支撑的中间件容器，可独立部署。

### 系统运行态部署方案

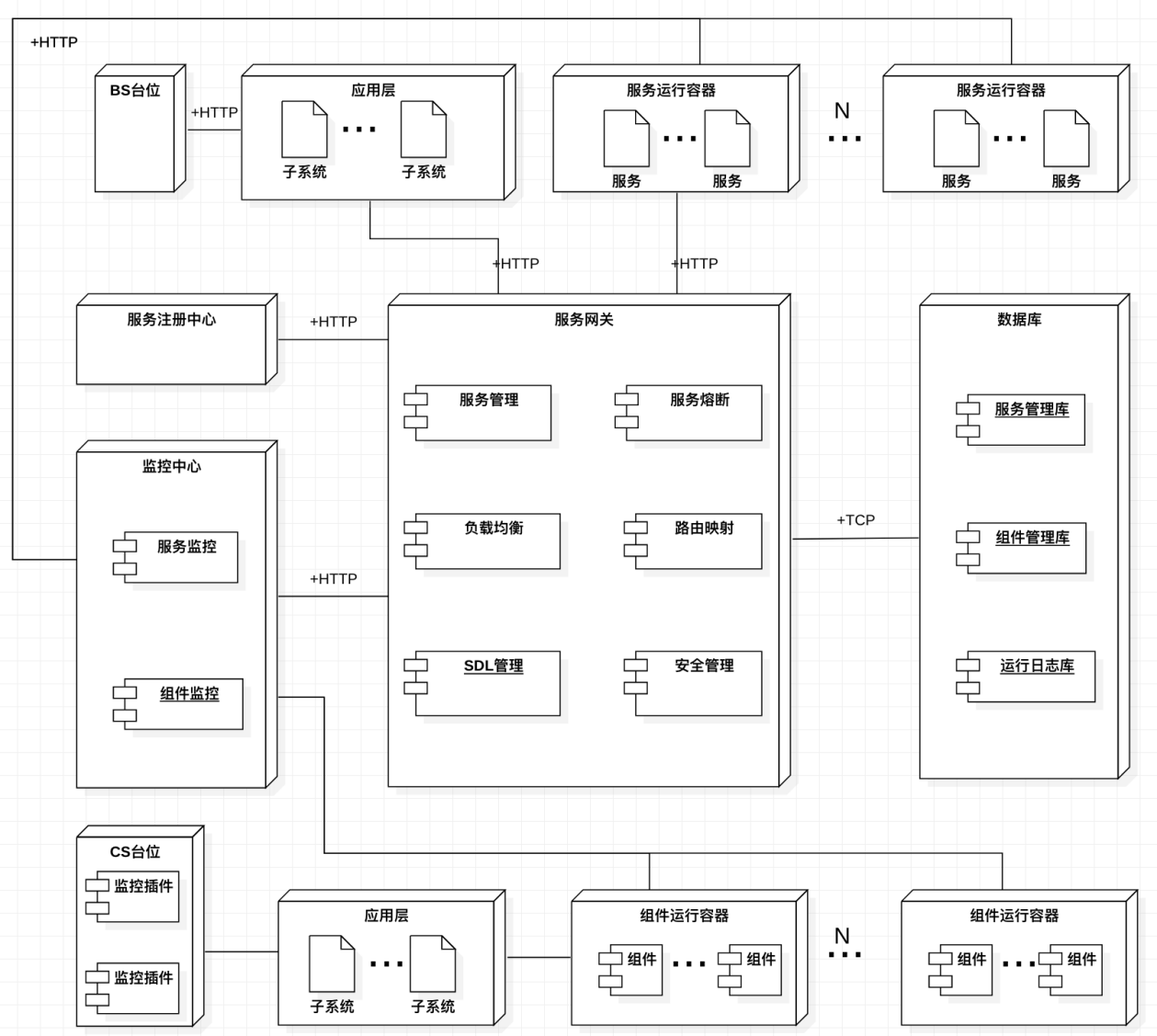


图5 系统开发态逻辑部署图

1. B/S台位。是指管理类子系统的管护端，展示功能统一由浏览器支持并实现。与应用层的子系统采用HTTP协议进行通信。
2. C/S台位。由部署并运行在界面插件容器中的各种监控插件协同实现，属于监控类子系统的业务展现层，与应用层各子系统进行通信。
3. 应用层。包括各子系统，监控类子系统的业务逻辑层，由部署并运行在组件运行容器中的监控组件和各种服务协同实现。服务包括基础服务和业务服务，体现为微服务的方式，以HTTP的方式进行通信。

## 外部接口

### 接口标识和接口图



图6 APPPL&&DP接口图

APPPL&&DP的主要任务是基于Eclipse的GEF框架，对导入的构件进行可视化的应用规划。APPPL&&DP工具是以用户导入的本地CDL文件为输入，解析CDL文件，将文件中包含的构件类型以图元的形式展示在调色板中。用户首先拖取应用图元设置应用编辑区域，并用连线方式在高层规划应用之间数据的发布订阅关系以及接口的调用关系，应用之间连线完成后拖取构件图元在应用编辑区域表达应用与构件之间的隶属关系，通过用带有箭头的连接线表达端口与端口之间数据发布与订阅的关系以及接口与接口之间调用的关系。应用与应用，应用与构件，构件与构件之间的关系的变化反映到图形编辑区域，最终根据该可视化编辑文件中的结果生成应用规划配置文件。通过采用图形化方式编辑应用规划配置文件的方式，使得应用开发人员能高效率快速地编写配置文件，解决了手工编写配置文件效率低下，易出错，不易维护等问题。

APPPL&&DP接口图如图6所示，接口图中展示了5个接口，分别为EGI（Edit Graphic Interface）、CPI（CDL Parse Interface）、GEI（Generate Edit File Interface）、GPI（ Generate Plan File Interface）以及TFI（Transport File Interface）。

### EGI接口

EGI接口定义了可视化编辑应用规划信息的核心操作方法集，包括创建、修改、删除应用编辑区域以及构件实例，表达构件之间端口、接口连接关系等操作。EGI用于根据用户的输入请求调用应用可视化规划子工具中的实现方法，从而对用户的请求做出响应，达到与用户交互的目的。EGI的接口实体中的实现方法将与应用规划模型信息产生数据交换，从而实现改变应用规划信息的功能。EGI的接口实体中的实现方法还会与应用规划视图产生信息交换，从而刷新视图反映当前应用规划信息。

### CPI接口

CPI接口定义了提取指定的CDL文件内信息并利用提取的构件模型信息生成可视化编辑文件.apf文件中调色板构件图元选项的方法集。用户可以通过拖取构件以达到规划应用的目的。

### GEI接口

GEI接口定义了将图形编辑区域以及应用编辑区域上的应用规划信息存入可视化编辑文件以及读取可视化编辑文件中有效信息的方法集合。应用规划文件生成子工具可以调用该接口，读取可视化编辑文件中的信息。

### GPI接口

GPI接口定义了将可视化文件解析并提取必要信息生成应用规划配置文件的方法集。GPI接口方法与应用规划信息进行数据交互；GPI中的配置文件生成方法，以解析可视化文件获得的其中的应用规划信息为输入，组织成应用规划配置文件。

### TFI接口

TFI接口定义了加载并读取应用规划配置文件，提取有效信息，分发配置文件和应用所包含的构件动态链接库至应用所在的主机的方法集。方法可以解析配置文件，获得应用所在的主机IP，构件类型，构件实例名，构件动态链接库所在的文件夹位置等信息，将配置文件连同动态链接库文件发送至构件所属应用所在的主机，这些文件供构件集成框架使用。

# 模块设计

## 用户管理

用户管理部分主要使用基于角色的访问控制模型(RBAC)对用户和权限进行管理。主要包括用户管理，角色管理，权限管理以及认证授权管理和令牌管理这几个模块。由于增加了角色，授权会更加灵活方便。

图7 用户管理的功能模块

### 用户管理

#### 1、用户注册

（1）输入用户名，将该用户名与数据库用户表进行比对。

（2）若该用户名已经存在，表示该用户名已经被注册过，返回错误信息，否则进入（3）。

（3）输入用户信息。

（4）将用户信息存入数据库。

2、删除用户

（1）输入用户信息。

（2）将该用户信息与数据库角色表进行比对，验证用户身份类型。

（3）若该用户为普通用户，返回无权限进行删除操作信息；若该用户为管理员，进入（4）。

（4）进入数据库，找到要删除的用户，将其删除。

3、修改用户

（1）输入用户信息。

（2）将该用户信息与数据库角色表进行比对，验证该用户是否存在。

（3）若该用户不存在，返回错误信息，否则进入（4）。

（4）进入数据库，对需要修改的信息进行修改。

4、查找用户

（1）输入用户信息。

（2）将该用户信息与数据库角色表进行比对，验证该用户是否存在。

（3）若该用户不存在，返回错误信息，否则进入（4）。

（4）进入数据库查找所需要信息。

5、用户登录

（1）输入用户信息。

（2）将该用户信息与数据库角色表进行比对，验证该用户是否存在。

（3）若用户信息输入无误，显示登陆成功，否则返回错误信息。

### 角色管理

1、增加角色

（2）权限管理员输入所要增加的角色信息。

（2）若该角色信息已经存在，返回错误信息，否则将角色信息增加到数据库中。

2、删除角色

（1）权限管理员在列表中找到所要删除的角色条目。

（2）从数据库中将角色删除。

3、修改角色

（1）权限管理员在列表中找到所要修改的角色。

（2）修改需要修改的角色信息并保存至数据库。

### 权限管理

1、增加权限

（1）权限管理员执行增加权限操作。

（2）若已经存在该权限，返回错误信息，否则将权限增加至数据库。

2、删除权限

（1）权限管理员在列表中找到需要删除的权限。

（2）将对应的权限从数据库中删除。

3、权限验证

（1）通过自定义一个过滤器，来配置可以访问的url。

（2）若该请求没有权限访问，则返回错误信息。

4、修改权限

（1）权限管理员在列表中找到需要修改权限的用户。

（2）修改相应的权限并将其保存。

5、查找权限

（1）查找用户。

（2）查找相应的权限。

6、权限缓存

（1）用户发送一个请求。

（2） 若需要的权限存在缓存中，我们从缓存中进行加载。

3. 若需要的权限不存在缓存中，我们则从数据库中进行加载并将其保存在缓存中。

5.1.4 认证授权管理

用户授权验证的一般流程：

（1）用户进行登录。

（2）请求被登录验证的拦截器所拦截，验证用户是否存在。

（3）若用户验证通过，则进入（4），否则返回错误信息。

（4）根据用户信息的查询角色信息。

（5）根据角色信息查询功能信息（资源信息），并返回。

（6）根据返回结果进行授权。

### 令牌管理

#### 业务流程

#### 1、生成令牌

（1）用户发起获取token的请求。

（2）过滤器验证url是否是认证的请求，并通过客户端id查询生成一个认证对象。

（3）通过用户名和生成的认证对象生成一个用户信息对象，并检查用户是否存在。

（4）验证是否是refreshToken请求。

（5）如果是token，通过AbstractTokenGranter中的getAccessToken方法，调用tokenStore中的getAccessToken方法获取token。

（6）将token对象包装进响应流返回。

2、刷新令牌

（1）用户发起获取token的请求。

（2）过滤器验证url是否是认证的请求，并通过客户端id查询生成一个认证对象。

（3）通过用户名和生成的认证对象生成一个用户信息对象，并检查用户是否存在。

（4）验证是否是refreshToken请求。

（5）如果是refreshToken，通过RefreshTokenGranter中的getAccessToken方法，使用tokenStore中的refreshAccessToken方法获取token。。

（6）将token对象包装进响应流返回。

2、验证令牌

（1）用户携带token发起请求。

（2）过滤器验证url是否是认证的请求，并通过客户端id查询生成一个认证对象。

（4）通过用户名和生成的认证对象生成一个用户信息对象，并检查用户是否存在。

（4）如果用户存在，则验证通过。

3、获取用户信息

（1）用户携带token发起请求。

（2）如果通过验证，则可以通过调用get方法来获取用户的信息。

#### REST接口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 操作 | HTTP 动作 | 输入 | 返回值 |
| 用户登陆生成令牌 | POST /auth/login | JSON 用户对象 | JSON 用户角色、令牌 |
| 刷新令牌 | POST /auth/login | JSON 令牌 | JSON 令牌 |
| 验证令牌 | POST /auth/validateToken | JSON 令牌、用户对象 | JSON 验证结果 |
| 根据令牌获取用户信息 | POST /auth/getUserByToken | JSON 令牌 | JSON 用户对象 |

#### 文件清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 控制类 | | 操作 | | |
| com.kenji.web.AuthController | | createAuthenticationToken：用户登陆  refreshAuthenticationToken：刷新令牌  validateToken：验证令牌  getUserByToken：根据令牌获取用户信息 | | |
| 业务类 | | | | |
| 类名 | AuthServiceImpl | | | |
| 包名 | com.kenji.service.impl | | | |
| 父类 | 无 | | | |
| 接口 | AuthService | | | |
| 方法 | 参数 | | 返回值 | 描述 |
| login | User 用户信息 | | String 令牌 | 用户登陆生成令牌 |
| refresh | String 令牌 | | String 令牌 | 根据旧令牌刷新令牌 |
| validateToken | String 令牌  User 用户信息 | | Boolean 验证结果 | 根据令牌及用户信息验证令牌 |
| getUserByToken | String 令牌 | | User 用户信息 | 根据令牌返回用户信息 |

## 服务集成

服务集成部分用于支持服务和组件的管理以及监控，具体包括以下功能：

服务网关依据服务管理所产生的信息，对服务的访问进行过滤（根据授权和发布订阅关系）和转接（相当于代理）。主要功能包括：

### 服务管理

图7 服务管理的功能模块

服务管理

数据库

服务注册

服务发布

服务查询

服务注销

服务撤回

服务订购

服务运行日志库

服务管理库

服务描述文件管理

SDL查询

SDL增加

SDL修改

SDL删除

服务管理主要是对服务本身的管理和对服务描述文件的管理。包括服务的注册与注销、服务的发布与撤回、服务的查询获取、服务的订购与授权和服务描述文件的管理。

#### 服务注册/注销

本平台并不提供服务开发平台，服务开发人员（服务提供者）可以利用任何平台和编程语言开发服务，只要其遵循REST或微服务规范即可。

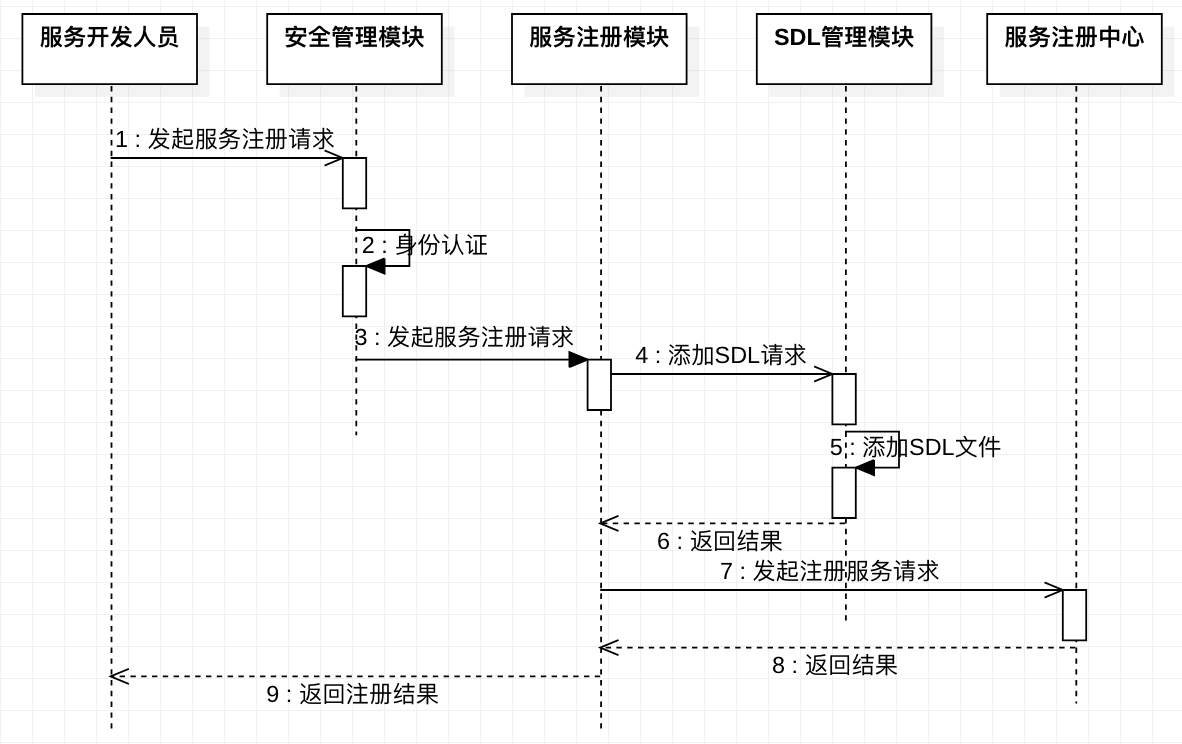


图9 服务注册模块的工作过程

服务注册模块的工作过程

1. 开发并测试好的服务完成部署后，首先向服务管理模块发起服务注册的REST请求；
2. 由服务网关中的安全管理模块验证身份，验证成功后进入（3），否则返回错误信息；
3. 由服务管理中的服务注册模块调用SDL管理模块将注册信息及服务描述文件保存于服务管理库中，包括服务的分类信息和描述服务的各种元数据。
4. 由服务管理模块向服务注册中心发起服务注册的REST请求，将服务注册至服务注册中心中，并返回结果。

当然，服务注销是服务注册的反操作，它从服务管理库中删除服务的分类与描述信息，并将已注册至服务注册中心的服务注销。

#### 服务发布/撤回

服务管理人员或服务提供者利用此功能将已注册的服务向外界发布，使其对外部可见。服务撤回是服务发布的反操作。

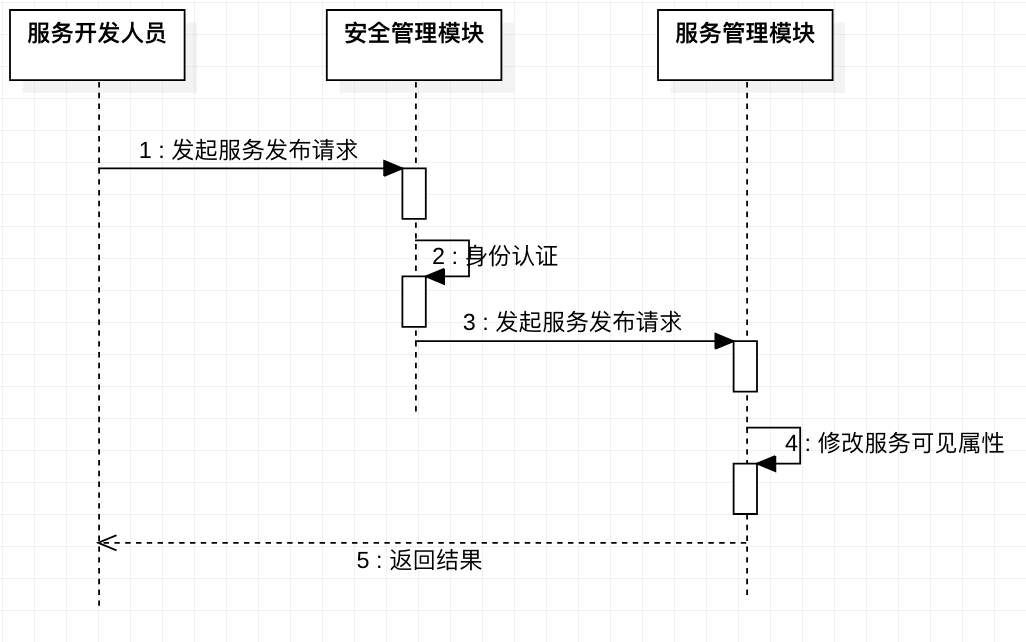


图10 服务发布模块工作过程

1、已经注册完成的服务，首先向服务管理模块发起服务发布的REST请求；

2、由服务网关中的安全管理模块验证身份，验证成功后进入（3），否则返回错误信息；

3、由服务管理中的服务发布模块修改服务管理库中的服务状态为发布状态。

4、由服务管理模块返回结果。

#### 服务查询

系统管理人员可以查询注册和发布的所有服务，其他人员利用此服务只能查询平台已发布的所有服务。

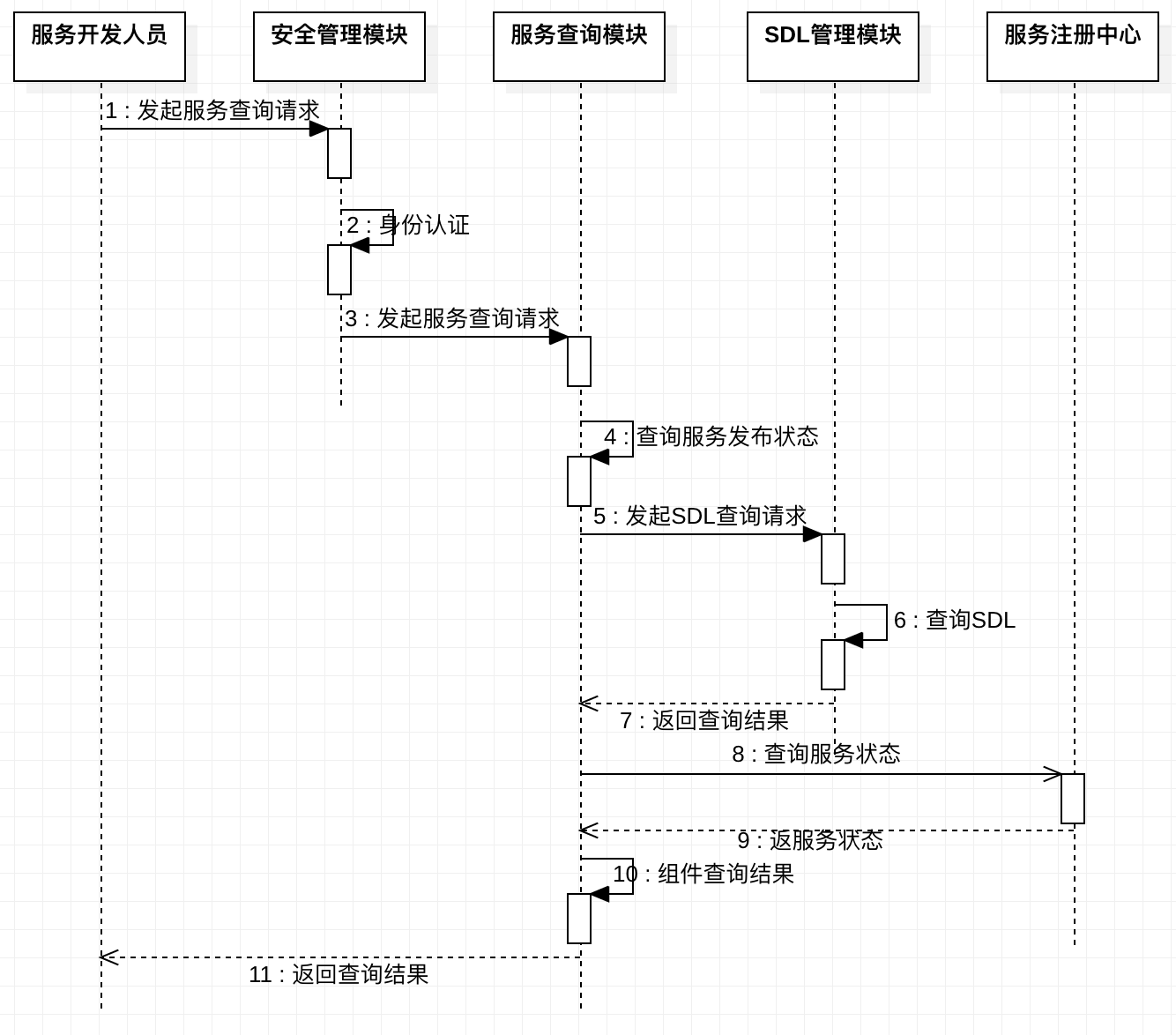


图11 服务查询模块的工作过程

1. 服务查询首先向服务网关发起服务查询的REST请求；
2. 由服务网关中的安全管理模块验证身份，验证成功后进入（3），否则返回错误信息；
3. 服务查询模块调用SDL管理模块通过服务管理库查询服务描述文件及注册信息；
4. 服务查询模块向服务注册中心发起服务查询的REST请求，查询服务的状态信息；
5. 由服务查询模块将查询结果合并后返回。

#### 服务订购与授权

可以对平台中所有已发布的服务进行选择并订购，服务管理人员对订购行为进行审核，若审核通过，则给予订购者以服务授权。服务授权的结果是给订购者颁发一个token，在服务运行阶段，所有对服务的访问都依据该token进行访问控制。



图12 服务订购模块的工作过程

服务消费者获取token的过程，其中服务消费者（浏览器、程序、组件及服务），安全管理模块可以是由服务管理人员来进行审核。当服务消费者使用用户提供的用户名和密码来直接请求token，认证服务器验证通过后，会返回一个token response并将token自身也保留下来。服务消费者提供以下参数请求安全管理模块：

|  |  |  |
| --- | --- | --- |
| 参数 | 是否可选 | 备注 |
| grant\_type | 必选 | 值固定为“password” |
| username | 必选 | 用户登陆账号 |
| passward | 必选 | 用户登陆密码 |
| scope | 可选 | 授权范围 |

表1 请求参数

|  |
| --- |
| POST /token HTTP/1.1  Host: server.example.com  Content-Type: application/x-www-form-urlencoded  grant\_type=password&username=xxx&password=123 |

图8 请求参数示例

授权服务器会返回如下信息：

|  |  |
| --- | --- |
| 参数 | 备注 |
| access\_token | 访问令牌 |
| refresh\_token | 刷新令牌 |
| expires\_in | 过期时间 |

表2 返回参数

|  |
| --- |
| HTTP/1.1 200 OK  Content-Type: application/json;charset=UTF-8  {  "access\_token":"2YotnFZFEjr1zCsicMWpAA",  "token\_type":"example",  "expires\_in":3600,  "refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA",  "example\_parameter":"example\_value"  } |

图13 返回结果示例

刷新令牌：在上述得到访问令牌（access\_token）时，一般会提供一个过期时间和刷新令牌。以便在访问令牌过期失效的时候可以由客户端自动获取新的访问令牌，而不是让用户再次登陆授权。如下是刷新令牌的收客户端需要提供给Authorization Server的参数：

|  |  |  |
| --- | --- | --- |
| 参数 | 是否可选 | 备注 |
| grant\_type | 必选 | 固定值“refresh\_token” |
| refresh\_token | 必选 | 客户端得到access\_token的同时拿到的刷新令牌 |

表3 刷新令牌参数

|  |
| --- |
| POST /token HTTP/1.1  Host: server.example.com  grant\_type=refresh\_token&refresh\_token=tGzv3JOkF0XG5Qx2TlKWIA |

图14 刷新令牌示例

#### 服务描述文件管理

服务描述文件（SDL）进行管理包含服务描述文件的查询获取、添加、删除、修改等操作。服务描述文件SDL是通过使用XML的方式对于服务名称、服务的输入输出参数、端口号等参数信息（详细参数信息见表4）进行了描述。服务消费者可以通过被调用服务的描述信息，从而解析服务调用所需的参数信息。

|  |  |
| --- | --- |
| 参数名称 | 描述 |
| serviceName | 服务的名称 |
| hostName | 主机名（域名） |
| ipAddr | IP地址 |
| portNumber | 端口号 |
| status | 服务状态 |
| overriddenstatus | 服务实例的覆盖状态 |
| managementPort | 监控端口 |
| healthCheckUrl | 健康检查接口 |
| statusPageUrl | 信息查询接口 |
| serviceInputParameter | 服务输入参数 |
| serviceOutputParameter | 服务输出参数 |
| dataCenterInfo | 使用的数据中心 |
| metadata | 主页 |
| lastDirtyTimestamp | 最后修改时间 |

表4 服务描述文件参数列表

服务描述文件管理可以分为几个功能模块，分别为：服务描述文件的查询获取功能模块、服务描述文件的添加功能模块、服务描述文件的修改功能模块、服务描述文件的删除功能模块这四个功能模块。如图15所示：

服务描述

文件添加

服务描述

文件查询

服务描述

文件删除

服务描述

文件修改

服务描述文件查询功能模块

图15 服务描述文件管理的功能模块

服务描述文件（SDL）查询获取过程如下：



图16 服务描述文件查询模块的工作过程

1. 服务消费者向服务查询模块发起调用请求；
2. 服务查询模块向SDL查询模块发起查询请求；
3. SDL查询模块向服务管理库请求查询操作，SDL查询模块的查询数据库中与被调用服务的服务名相同的元组，如果存在，获得服务实例的描述文件并返回；如果不存在，则返回失败信息。

服务描述文件（SDL）的其它操作：增加、删除和修改操作，其也需先经过上述查找过程，然后进行服务描述文件相应的操作。

1. 服务描述文件（SDL）添加流程如下：

图17 服务描述文件添加模块的工作过程

1、服务开发人员向SDL增加模块发送添加命令和欲添加服务的参数；

2、SDL增加模块对服务管理库进行查找，如果存在，则返回失败信息，如果不存在，则进入（3）；

3、解析SDL文件，并将对应服务描述信息添加至服务管理库中，并返回结果。

（2）服务描述文件（SDL）修改流程如下：

图18 服务描述文件修改模块的工作过程

1、服务开发人员向SDL修改模块发送修改命令和欲修改服务的参数；

2、SDL修改模块对服务管理库进行查找，如果不存在，则返回失败信息；如果存在，则进入（3）；

3、SDL修改模块把服务的欲修改的信息参数发送给服务管理库进行修改，并返回结果。

（3）服务描述文件（SDL）删除流程如下：

图19 服务描述文件删除模块的工作过程

1、服务开发人员向SDL删除模块发送删除命令和欲删除服务的参数；

2、然后SDL删除模块就对服务管理库进行查找，如果不存在，则返回失败信息；如果存在，则进入（3）；

3、SDL删除模块删除服务管理库中相应的元组，并返回结果。

#### 服务注册中心

包含了服务器端和客户端组件。服务器端即服务注册中心，用于提供服务的注册与发现。客户端组件包含服务消费者与服务提供者。在应用程序运行时，客户端向注册中心注册自身提供的服务并周期性的发送心跳来更新它的服务租约。同时也可以从服务端查询当前注册的服务信息并把他们缓存到本地并周期性的刷新服务状态。

### 服务网关

#### 负载均衡

如果后台被调用服务启动了多个实例，则在服务映射时，可根据服务运行负载情况进行负载均衡，同时保持服务会话的完整性，且后台服务应该是无状态的。

负载均衡模块

服务注册中心

服务实例1

服务实例3

负载均衡请求

注册

注册

可用

服务

列表

服务实例2

注册

图20 负载均衡模块的工作过程

服务实例的负载均衡主要通过负载均衡模块来实现，负载均衡模块会定期从服务注册中心更新并过滤服务实例列表，在服务注册时可指定服务的负载均衡策略，这些策略包括：轮询、随机、根据响应时间加权等。在服务调用时负载均衡模块会按以下步骤执行：

（1）优先选择在同一个Zone（可理解为同一机房中）中且负载较少的服务注册中心；

（2）根据网关查询到的服务的负载均衡策略，在从服务注册中心取到的服务注册列表中选择一个实例的地址；

（3）进行服务调用，并返回调用结果。

其中，负载均衡模块机制如下



图21 负载均衡模块

RetryRule:重试，在一定时间内，一直重试轮询。在定义RetryRule时可传入，默认500毫秒。

##### REST接口描述

#### 文件清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 控制类 | | 操作 | | |
| com.kenji.web.AuthController | | createAuthenticationToken：用户登陆  refreshAuthenticationToken：刷新令牌  validateToken：验证令牌  getUserByToken：根据令牌获取用户信息 | | |
| 业务类 | | | | |
| 类名 | AuthServiceImpl | | | |
| 包名 | com.kenji.service.impl | | | |
| 父类 | 无 | | | |
| 接口 | AuthService | | | |
| 方法 | 参数 | | 返回值 | 描述 |
| login | User 用户信息 | | String 令牌 | 用户登陆生成令牌 |
| refresh | String 令牌 | | String 令牌 | 根据旧令牌刷新令牌 |
| validateToken | String 令牌  User 用户信息 | | Boolean 验证结果 | 根据令牌及用户信息验证令牌 |
| getUserByToken | String 令牌 | | User 用户信息 | 根据令牌返回用户信息 |

#### 服务熔断

服务熔断是处理服务故障并借以提高系统性能的机制。其一般是指软件系统中，由于某些原因使得服务出现了过载现象，为防止造成整个系统故障，从而采用的一种保护措施，所以很多地方把熔断亦称为过载保护。主要包括两个方面：

1. 服务熔断：当这个服务出现故障的时候，来访问我的请求我会直接响应他一个Fallback（即降级操作）而不会让他一直等待请求；
2. 服务降级：当服务器压力剧增的情况下，根据当前业务情况及流量对一些服务和页面有策略的降级，以此释放服务器资源以保证核心任务的正常运行。

服务熔断集成了服务降级、熔断机制、隔离机制。

（1）资源隔离机制：包括线程池隔离和信号量隔离，其作用限制调用分布式的资源使用，某一个调用的服务出现问题不会影响其他的服务调用。

线程池隔离模式：使用一个线程池来存储当前请求，线程池对请求作处理，设置任务返回处理超时时间，堆积的请求先入线程池队列。这种方式要为每个依赖服务申请线程池，有一定的资源消耗，好处是可以应对突发流量（流量洪峰来临时，处理不完可将数据存储到线程池队里慢慢处理）。

信号量隔离模式：使用一个原子计数器（或信号量）记录当前有多少个线程在运行，请求来先判断计数器的数值，若超过设置的最大线程个数则丢弃该类型的新请求，若不超过则执行计数操作请求来计数器+1，请求返回计数器-1。这种方式是严格的控制线程且立即返回模式，无法应对突发流量（流量洪峰来临时，处理的线程超过数量，其他的请求会直接返回，不继续去请求依赖的服务）。

（2）服务降级：在服务和请求之间增加线程池，用户的请求将不再直接访问服务，而是通过线程池中的空闲线程来访问服务，如果线程池已满，则会进行降级处理，用户的请求不会被阻塞，至少可以看到一个执行结果，而不是无休止的等待或者看到系统崩溃。

（3）熔断机制：为系统中如果存在某个服务，失败率过高时，将开启熔断器，对于后续的调用，不在继续请求服务，而是进行Fallback操作。

服务熔断和服务降级的实现过程如下：

断路器针对服务存在三种状态，分别为：关闭状态（Closed）、半熔断状态（Half-Open）、熔断状态（Open）。正常情况下，断路器处于关闭状态（Closed），如果调用持续出错或者超时，电路被打开进入熔断状态（Open），后续一段时间内的所有调用都会被拒绝（Fail Fast），一段时间以后，保护器会尝试进入半熔断状态（Half-Open），允许少量请求进行尝试，如果调用仍然失败，则回到熔断状态；如果调用成功，则回到电路闭合状态。

Calls failing fast

Open

Success Trip Breaker Attempt Reset

Trip Breaker

Closed

Half-Open

Reset Breaker

图22 熔断器的状态转换

服务降级：如果服务失败，则通过fallback进行降级，返回静态值。具体过程为首先创建一个服务熔断中处理降级类的对象实例来向其组件发出操作请求，再通过调该服务降级类实例对象中的run()方法，如果线程池已满，即通过fallback进行降级。如下图所示：

getFallback()

服务降级类对象实例

Run()

图23 服务降级

#### 路由映射

服务映射主要由路由映射模块完成。路由映射模块会代理所有注册到服务注册中心的服务，并将所发布的（外部可见的）服务接口的调用，转换为对后台相应服务（一般与外部可见的服务具有不同的接口URL）的调用，并截获内部服务调用结果转发给服务请求者。

过滤器（Filter）是路由映射的核心，用来实现对外服务的控制。所以路由映射主要包含三大模块：Core 模块、过滤器加载模块、过滤器管理模块。路由映射的模块划分如图24所示：

Http Request

Http Response

Core

过滤器加载

Servlet

Filter Loader

Filter Runner

Filter File Manager

Request

Context

Post

Routing

Filter

Routing

Filter

Pre Routing

Filter

过滤器管理

Filter Poller

Filter Persister

Filter Publisher

图24 路由映射的功能模块

1、核心模块：对应于上面架构图的Core。用来接收请求然后通过 pre、route、post 三个主要类型的过滤器。其中还有另外两个上图中没有展现，就是 error 和 custom 类型。error 是处理异常时的处理动作，custom 是自定义处理请求的过滤流程。上面各个过滤器之间是通过 RequestContext来做上下文传递的。FilterRunner 组织了各种类型的过滤器的执行逻辑。

Filter是路由映射的核心，用来实现对外服务的控制。路由映射大部分功能都是通过过滤器来实现的，这些过滤器类型对应于请求的典型生命周期。具体如下：

* PRE： 这种过滤器在请求被路由之前调用。我们可利用这种过滤器实现身份验证、在集群中选择请求的微服务、记录调试信息等。
* ROUTING：这种过滤器将请求路由到微服务。这种过滤器用于构建发送给微服务的请求，并使用HTTP客户端或负载均衡组件请求微服务。
* POST：这种过滤器在路由到微服务以后执行。这种过滤器可用来为响应添加标准的HTTP Header、收集统计信息和指标、将响应从微服务发送给客户端等。
* ERROR：在其他阶段发生错误时执行该过滤器。 除了默认的过滤器类型，路由映射还允许我们创建自定义的过滤器类型。例如，我们可以定制一种STATIC类型的过滤器，直接在路由映射中生成响应，而不将请求转发到后端的微服务。

2、过滤器加载模块：这个模块主要是给Core 模块的FilterRunner 提供服务的，也就是提供编译好的Filter组件。其中的FilterFileManager 是以轮询的方式，不断的从包含groovy 脚本的目录中加载文件，提供给 FilterLoader 编译和管理。

3、过滤器管理模块：这个模块需要对其做定制开发，从而适应不同的业务需求。

#### 安全管理

服务访问控制是在用户进行服务订购操作的基础上才能进行。当用户对服务进行订购之后会获取到一个token，当用户发出请求的时候，将token以参数形式拼接至请求URL后。当拦截器接受到请求之后，将订购时保存下来的token并与接受到的参数做比对，如果比对成功则通过网关调用服务，否则返回失败页面。

调用服务

调用服务

服务网关

拦

截

器

请求（包含token）

服务

拥有权限，调用服务

服务消费者

图25 服务访问控制的工作过程

### 服务监控

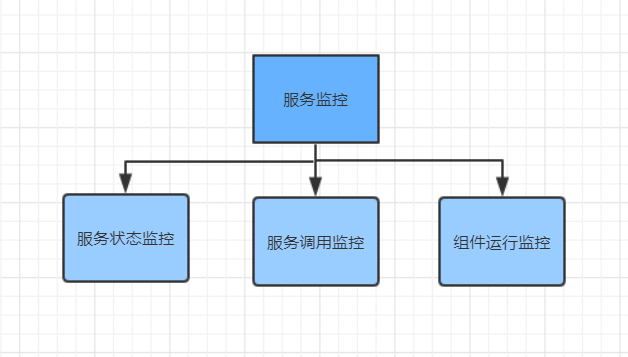


图26 服务监控的功能模块

服务监控主要包括服务状态监控、服务调用监控以及组件运行监控。

#### 服务状态监控

通过周期性的发送心跳来监测服务的状态，并将服务的状态信息保存并显示在服务网关中。

1.基于平台开发的服务

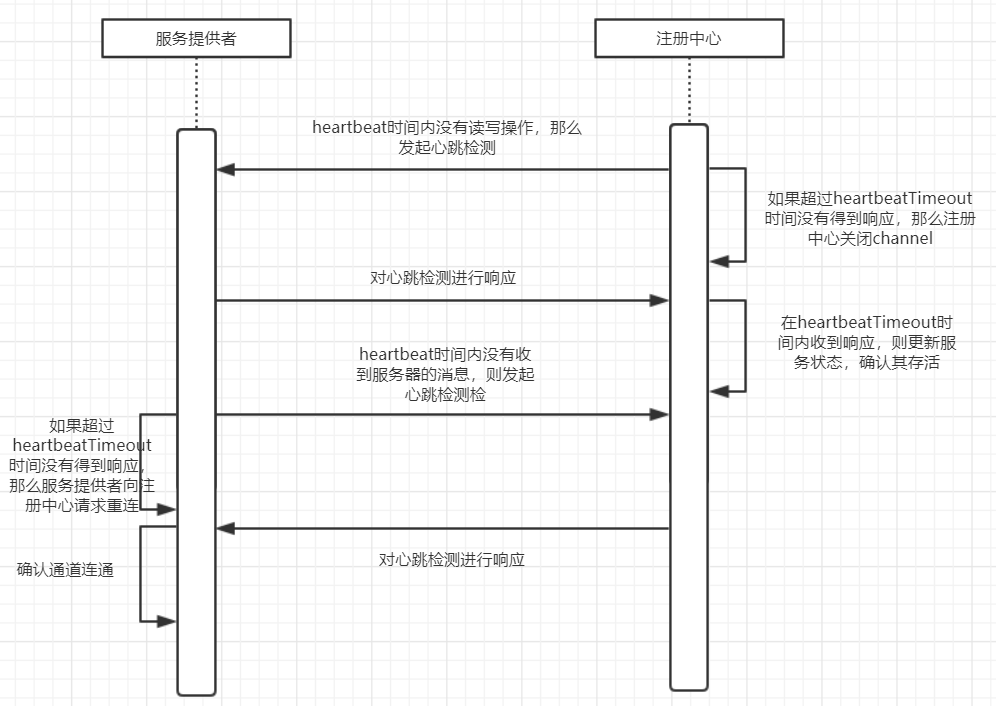


图27 服务续约的工作过程

1. 服务提供者在规定的heartbeat时间内没有进行任何的读写操作，服务注册中心就会向服务提供者发起心跳检测，检测服务提供者的健康状态，是否正常运行。

（2）规定heartbeatTimeout = 3\*heartbeat，如果在heartbeatTimeout的时间内注册中心没有收到服务提供者的响应，那么就认为服务提供者没有正常运行，则服务注册中心关闭与服务提供者的通道并更新服务的状态信息同步至其它注册中心。如果服务注册中心在heartbeatTimeout时间内收到了服务提供者的心跳检测响应，那么服务此时正常运行。

（3）服务提供者在规定的heartbeat的时间内没有收到服务注册中心的任何消息，那么服务提供者就会向服务注册中心发起心跳检测。

（4）如果在heartbeatTimeout时间内服务提供者没有收到服务注册中心的任何响应，那么就认为通道断开连接，服务提供者会进行重新连接。如果在heartbeatTimeout时间内收到注册中心的响应，那么一切正常。

2.非平台开发的服务

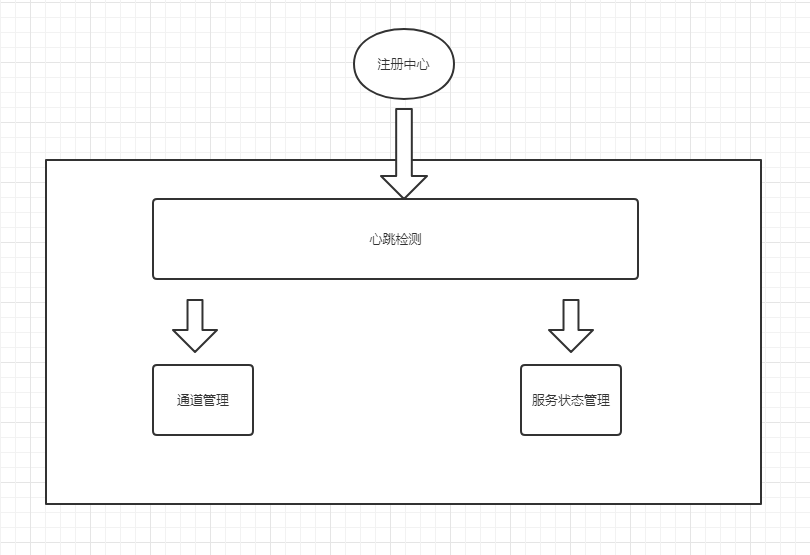


图28 服务状态监控的功能模块

1、心跳检测模块：当注册中心和服务提供者在满足一定条件时会去调用心跳检测模块发送心跳检测请求，并根据心跳响应做出反应。

2、通道管理模块：当注册中心没有得到服务提供者的心跳响应，那么注册中心认为服务提供者服务不可用，则关闭注册中心与服务提供者之间的通道。

3、服务状态管理模块：当注册中心没有得到服务提供者的心跳响应，那么注册中心认为服务提供者服务不可用，注册中心更新该服务的服务状态为DOWN并向其他注册中心同步该服务的注册信息。

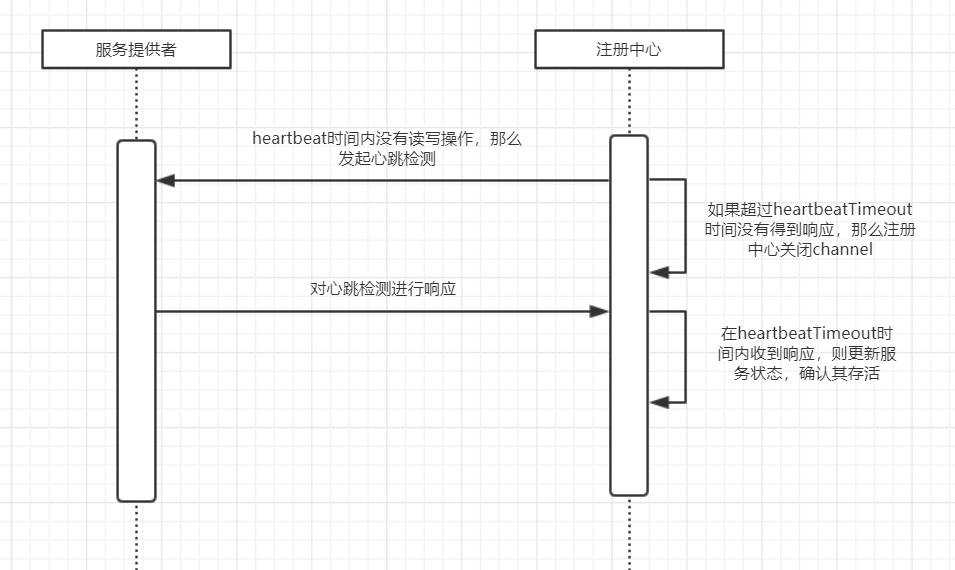


图29 服务状态监控的工作过程

1. 对于非平台开发的服务，当该服务向注册中心注册后，注册中心如果在heartbeat时间内没有收到服务的任何读写请求，那么注册中心就会向服务发起心跳检测，检测服务提供者的健康状态，是否正常运行。

规定heartbeatTimeout = 3\*heartbeat，如果在heartbeatTimeout的时间内注册中心没有收到服务提供者的响应，那么就认为服务提供者没有正常运行，则服务注册中心关闭与服务提供者的通道并更新服务的状态信息同步至其它注册中心。如果服务注册中心在heartbeatTimeout时间内收到了服务提供者的心跳检测响应，那么服务此时正常运行。

#### 服务调用监控

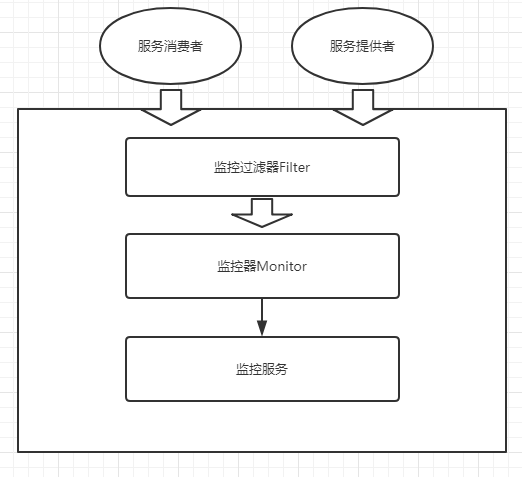


图30 服务调用监控的功能模块

1、监控过滤器模块：服务消费者的请求以及服务提供者的响应经过过滤器，过滤器会获取计算数据，并将数据发送给监控器。

2、监控器模块：监控器拿到过滤器的数据后进行聚合数据并暂存，随后定时调用监控服务模块进行数据持久化。

3、监控服务模块：将数据写入文件。

服务统计信息，对服务的注册数量、调用次数和可用服务的数量等进行统计并显示在服务网关中，一般流程如下：

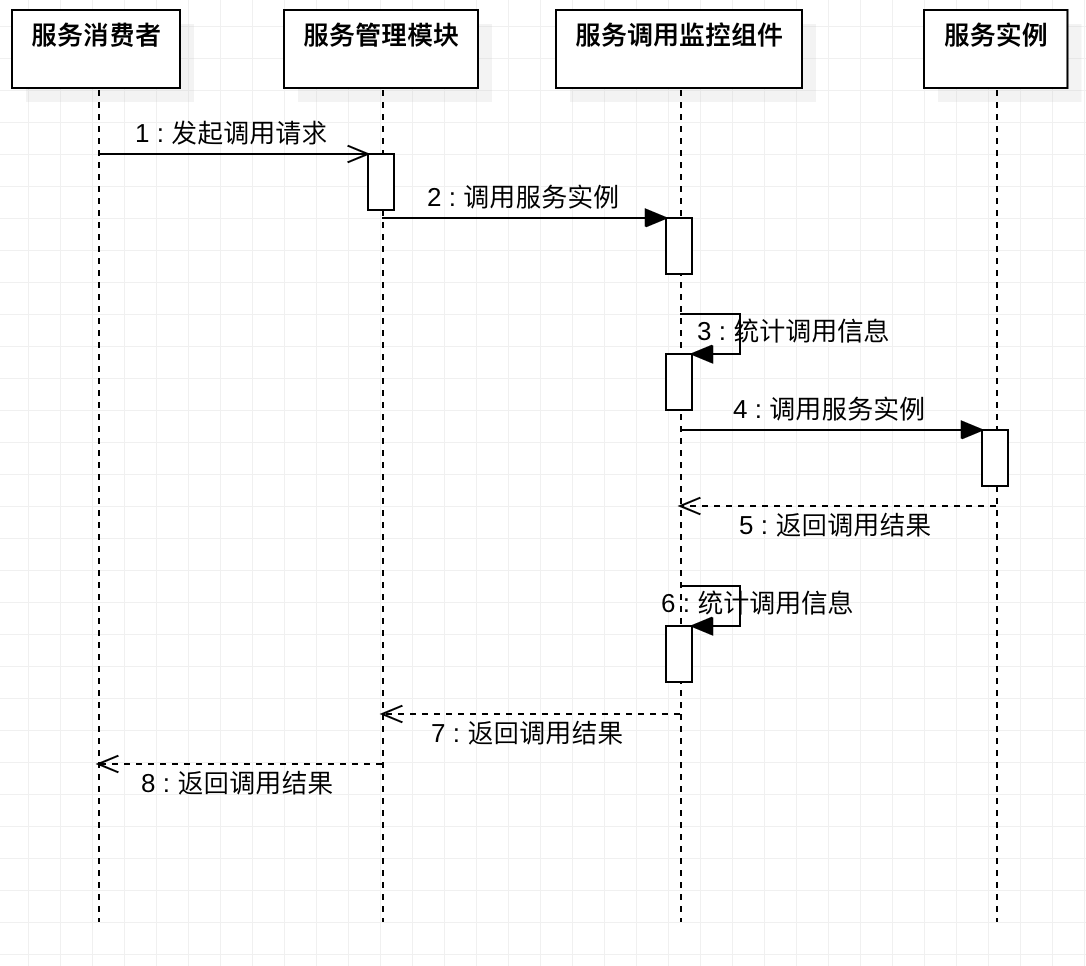


图31 服务调用监控模块的工作过程

1、服务消费者向服务管理模块发起调用请求，由服务调用监控模块记录调用情况；

2、服务实例返回响应信息，由服务监控模块对响应情况进行记录保存。

#### 组件运行监控

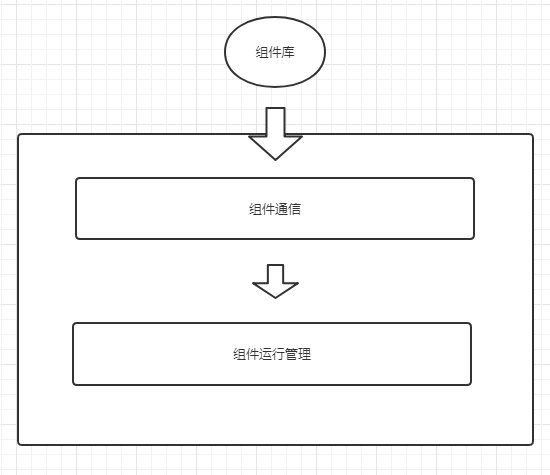


图32 组件运行监控的功能模块

1、组件通信模块：组件库定时会向组件运行容器发送组件运行监控的请求，组件运行容器在接收到请求后，组件运行容器中的组件通信模块把通过SendMsgEvent方法和OnMsgEvent方法来供服务组件和界面组件调用，进行通信。如果能和组件通信成功说明组件当前处于运行状态，如果某一组件无法通信成功，则将该组件的通信状态发送至

2、组件运行管理模块：如果通过组建通信模块，组件间的通信成功则说明组件当前处于运行状态，如果某一组件通信失败，则将该组件的通信状态更新为DOWN并将更新信息返回给组件库，组件库依据信息进行组件管理。

## 外部工具链

一组相对独立的工具，它们不直接参与系统的运行与监控，包括组件开发工具、组件库管理工具和组件配置与部署工具。

### 组件开发

1. 组件的定义

服务的本质是一个对外部开放的接口，主要对外提供服务，而组件的本质是一个能够被复用的封装体，主要用于复用。组件作为一种特殊化的结构化类，具有类的封装性、继承性和多态性。但是组件更强调其重用性，而重用性取决于组件是如何定义、如何实现以及如何使用的。定义一个组件的行为是要确定其供口和需口，供口确定了可以向外部提供什么服务，服务对象可以是最终用户、本地开发者或者把该组件暴露向远端提供服务；需口确定了它需要其他服务生产者或环境所提供的服务。组件的内部实现是自包含的，就是说组件具备理解自身所需要的全部信息，而不需要额外信息。一个组件是系统中的一个可替换单位，替换应基于接口兼容性而提供等同功能。替换可能发生在设计时刻，也可能发生在运行时刻，具体来说，一个组件的供口和应连接的需口具有相同类型或子类型，而且该组件的所有需口都以相同规则连接到其他组件，该组件方可替换。这里所说的组件主要是指监控插件与监控组件。

1. 组件调用服务

组件开发工具的核心是服务描述语言（SDL）解析器，通过解析标准的服务描述语言，将其解析为服务调用框架供服务消费者使用。服务描述语言旨在按照XML Schema规范设计一种合适的服务描述语言规范（Services Description Language SDL），此规范以XML Schema的形式体现。服务提供者依据XML Schema编写符合规范的服务描述文件的XML文件。服务描述语言是一种XML Application，它将[Web](https://baike.baidu.com/item/Web/150564" \t "https://baike.baidu.com/item/%E6%8F%8F%E8%BF%B0%E8%AF%AD%E8%A8%80/_blank)服务描述定义为一组服务访问点，[客户端](https://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF" \t "https://baike.baidu.com/item/%E6%8F%8F%E8%BF%B0%E8%AF%AD%E8%A8%80/_blank)可以通过这些服务访问点对包含面向文档信息或面向过程调用的服务进行访问。

SDL首先对访问的操作和访问时使用的请求/响应消息进行抽象描述，然后将其绑定到具体的传输协议和消息格式上，以最终定义具体部署的服务访问点。相关的具体部署的服务访问点通过组合就成为组合后的服务。

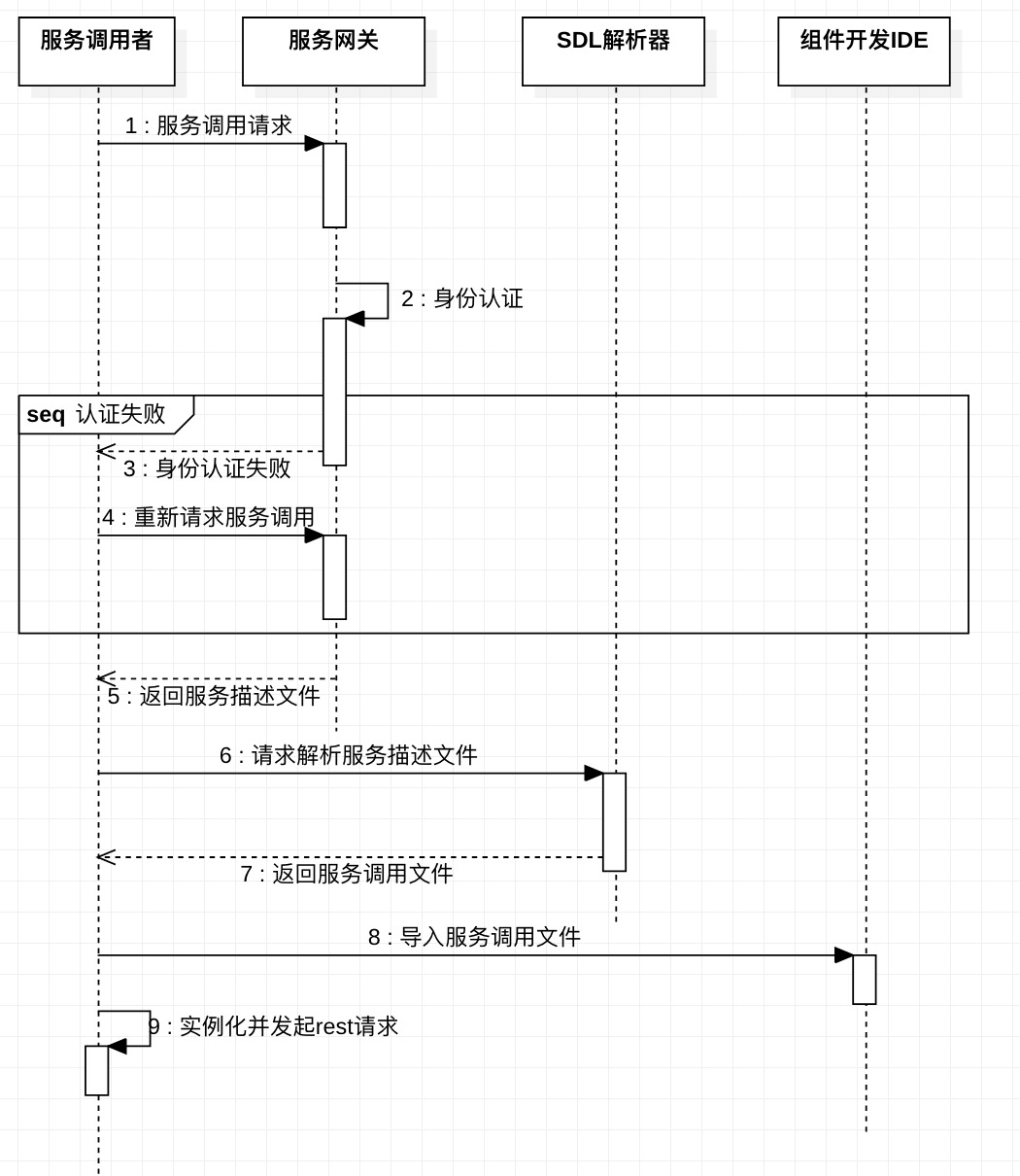


图33 组件调用服务的工作过程

为组件生成服务调用代码的一般过程如下：

1、组件开发人员向服务网关发起调用请求，并由服务网关进行身份认证，认证成功则进入（2），否则则返回错误信息；

2、服务网关查询服务服务描述文件(SDL)并返回，服务描述文件包括对服务名、服务输入输出参数格式、服务的IP地址和端口号以及服务的具体功能描述；

3、由可视化的服务描述语言解析器将其解析为目标语言类库（C++，java等），该类库包括头文件（.h）和源文件（.cpp）以及一个可选的主文件构成。

服务消费者在类库生成阶段即可选择将目标文件直接建立在调用该服务的项目文件中，也可以在类库生成后导入到对应项目文件中。服务消费者通过该类库实例化对象，调用该对象的服务调用方法。

服务描述语言（SDL）解析器使用QT框架进行界面搭建，并使用QT中提供的QDomDocument以及相关类库进行服务描述语言解析。解析生成的目标类的类名应与服务提供者提供的服务同名，类中声名的函数为Public类型并与服务生产者提供的远程方法同名，一个服务生产者对应一个具体的类，服务生产者的每个调用方法对应类的方法，在方法内提供发起HTTP请求，将所需要调用的服务以及参数发送至服务网关，服务网关通过服务查询，路由选择，负载均衡等一系列操作后，将请求发送至服务提供者，服务提供者将服务结果返回服务网关，再通过服务网关将服务结果返回服务消费者。

### 组件库管理

#### 输入输出

5.3.2.1.1. 用户管理

组件库面向多种用户角色，因此用户管理模块很重要。用户管理模块包括用户的添加、登录，以及用户角色的查询、修改和删除等操作。

用户管理需要与用户交互，用户管理模块所接收的输入和将产生的输出有：

1.用户登录

接收的输入：用户名、密码以及登录请求。

对应的输出：

（1）如果登录成功，则转入系统操作界面。

（1）如果登录失败，则提示用户名或密码错误信息。

2.用户添加

接收的输入：用户名、密码、真实姓名、性别等信息，以及添加请求。

对应的输出：

（1）如果注册成功，提示注册成功。

（2）如果有信息填写有误，则提示注册失败。

3.用户查询

接收的输入：用户名和查询请求

对应的输出：如果系统中存在相同用户名的用户，则列表显示；如果没有存在相同用户名的用户，则显示空列表。

4.用户删除

接收的输入：系统管理员选择的要删除的用户以及删除请求。

对应的输出：删除被管理员选中的用户，提示删除成功。

5.用户修改

接收的输入：需要修改的用户的信息以及用户修改请求。

对应的输出：

1. 输入的用户信息无误，提示修成功并显示修改后的用户信息。
2. 输入的用户信息有误，对相应错误输入做出提示。

5.3.2.1.2 组件上传

组件库管理人员可以通过组件库系统客户端上传自己开发的组件，但是组件的上传必须遵循一定的规则。一个完整的组件需包含组件的CDL文件，组件的实现，以及组件生成的DLL文件。其中组件的实现需打包压缩后上传。

接收的输入：上传组件的请求，组件名，组件关键字，组件描述信息，组件的CDL文件，组件的实现文件（压缩包），组件的DLL文件，组件版本。

对应的输出：

（1）若输入信息正确，则上传组件到服务端。

（2）若输入项有误，则提示相应项错误。

5.3.2.1.3. 组件下载

在组件库管理人员通过组件库系统客户端提交组件下载请求之后启动组件下载服务，组件库系统服务将组件库管理员期望下载的组件传输到目标机器的存储介质上。

接收的输入：组件库管理人员要下载的组件的选择，以及组件下载的请求。

对应的输出：传输组件库管理员所选择下载的组件到本地。

5.3.2.1.4. 组件删除

当组件已经不再被需要的时候，管理人员可以删除组件。

接收的输入：选定的要删除的组件、删除指令以及确认指令。

对应的输出：从系统中删除被选中的组件，并列表显示还存在的组件。

5.3.2.1.5. 组件检索

组件库管理员可以使用系统提供的两种组件模式：基于导航式的组件检索和基于关键字的检索模式。基于导航式的组件检索模式是将根据组件的特点对组件进行分类并以分类结构图的形式展现给用户，用户根据需要可以按照分类结构找到自己想要的组件。基于关键字的检索是根据用户的输入在服务器端进行匹配，反馈匹配成功的组件列表。

1.基于分类的组件检索

接收的输入：用户选择的分类。

对应的输出：分类下所有的组件。

2.基于关键字的组件检索

接收的输入：组件关键字，以及检索请求。

对应的输出：符合要求的组件列表。

5.3.2.2. 功能结构

组件库按照功能分为用户管理、组件上传、组件下载、组件删除、和组件检索五大功能。其功能结构图如图34所示。



图34 组件库功能结构图

5.3.2.2.1. 执行流程

1.用户管理

组件库面向组件库系统管理员，组件库系统管理员在对组件库进行操作的时候需要获得合法的权限，所以对用户的合法身份进行验证是至关重要的。

组件库提供针对用户和权限的四种服务：新用户注册服务，非法用户删除服务，用户检索服务和用户权限管理服务。用户管理流程图如图35所示。



图35 用户管理流程图

(1)用户添加服务：系统管理员是维护系统正常运行的专职人员。系统管理员可以添加用户，并且赋予该用户合理的权限。

(2)用户删除服务：为了维护组件库的正常运行，系统管理员需要删除不合法的用户。

(3)用户检索服务：用户检索服务是帮助系统管理员管理注册用户的辅助服务。在注册用户量很大的情况下，通过用户检索服务可以快速定位目标用户。

2.组件上传

组件库管理员可以通过组件库客户端上传组件，但是组件的上传必须遵循某些规则。



图36 组件上传流程图

如图23组件上传流程图所属，在组件库管理员提交组件上传请求之后启动组件上传服务，将用户需要上传的组件通过web服务传输到组件库服务端的数据库中，组件上传应该遵循如下规则：

1. 上传的组件格式。一个完整的组件包含组件的CDL定义、组件实现的程序代码、编译后生成的DLL文件。
2. 组件库管理人员在上传组件时需要填写组件描述字段并提供组件关键字(关键字可缺省)，以便下载组件时参考。为了帮助其他人员下载组件的时候有准确的参考，组件描述信息的准确性和完整性是很重要的。在组件上传时由组件库管理人员提供组件描述信息和准确的关键字可以保证组件检索时候的准确性。
3. 组件库管理员上传组件时需要为组件选择所属分类。组件库管理员可以更改组件的分类结构，包括分类结构的添加，删除和修改。

3.组件下载

组件下载流程图如图37所示。



图37 组件下载流程图

组件库系统管理人员需要登录之后才能下载组件。首先检索自己需要的组件，然后选择所需要的组件版本，下载到本地存储介质上使用。

4.组件删除

组件删除流程图如图38所示：



图38 组件删除流程图

组件库系统管理人员可以删除自己上传的组件。首先需要登录系统，然后选择需要删除的组件，点击删除并确认。

5.组件检索

组件检索流程图如图39所示。



图39 组件检索流程图

用户登录系统后可以使用组件检索功能。分为关键字检索和导航式检索。如果用户选择关键字检索，需要输入要检索的组件的关键字信息，系统会与组件库中组件上传时填写的关键字进行匹配，匹配成功则返回已匹配的组件信息列表供管理人员下载。

导航式检索是以分类结构图的形式展现给客户端用户的，用户根据需要按照分类结构找到自己想要的组建并下载。

### 组件配置与部署

#### 组件运行容器

支持组件开发人员从组件库中下载的组件放置到组件容器的相应的位置，并且组件规划人员以规划好应用，生成的XML文件也放置在组件容器的相应位置。组件容器加载服务组件和界面组件来形成实际的应用。组件容器支持查看组件的注册和反注册信息，调试组件的运行。

输入：

组件文件(DLL)和相关应用规划的配置文件（XML）

输出：

组件运行容器加载组件

组件运行容器的目录组织结构如图40所示：

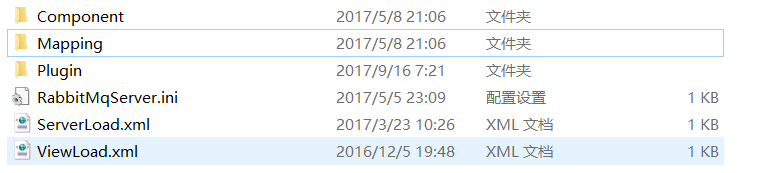


图40 行容器的目录组织结构图

1.Component文件夹存放服务组件的DLL文件。

2.Mapping文件夹存放每一个消息配置文件。

3.Plugin文件夹存放界面组件的DLL文件。

4.RabbitMqServer.ini设置RabbitMq的服务器IP。

5.ServerLoad.xml本台位需加载的服务组件。

6.ViewLoad.xml本台位需加载的界面组件。

#### 功能结构

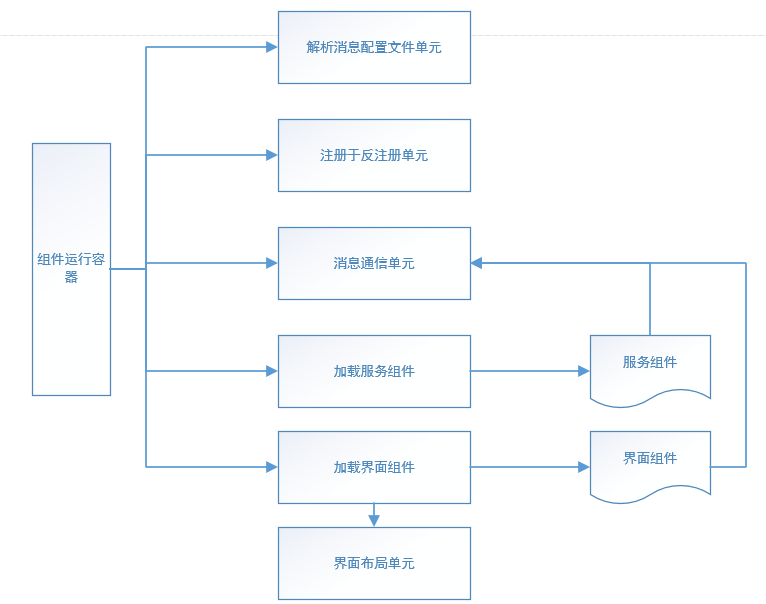


图41 组件运行容器静态调用关系图

组件运行容器调用RabbitMq消息中间件进行组件之间的通信，能够调试组件的运行，包括接口的调用，端口的发布和订阅功能，支持组件的注册和反注册。组件运行容器主要包含软件单元：消息通信单元，组件的注册和反注册单元，界面组件加载单元，服务组件加载单元，解析消息配置文件单元，界面布局单元。软件单元及期间静态调用关系。

1. 解析消息配置文件单元

解析ViewLoad.xml，SerLoad.xml和Mapping文件夹中所有的XML文件，把解析的结果保存到结构体中，解析结果，消息通信单元，界面布局单元都会用到。

2.注册于反注册单元

在组件运行容器界面显示组件的注册和反注册信息。

3.消息通信单元

消息通信单元把RabbitMq的通信封装成两个方法， SendMsgEvent方法和OnMsgEvent方法来供服务组件和界面组件调用，进行通信。

4.加载服务组件

加载服务组件单元动态加载服务组件并进行初始化组件，并告知组件容器已注册该服务组件。

5.加载界面组件

加载界面组件单元动态加载见面组件并进行初始化组件和界面布局，并告知组件容器已初测该服务组件。

6.界面布局单元

界面布局单元调用配置文件得到布局信息来布局该界面组件的位置。

参考文献

1. T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design[M]. Prentice Hall PTR, 2005.
2. 李贞昊. 微服务架构的发展与影响分析[J]. 信息系统工程, 2017(1):154-155.
3. James Lewis, Martin Fowler.Microservices a definition of this new architectural term，2014
4. 刘为. 微服务架构及相应云平台解析[J]. 科教导刊, 2017(1x):27-28.
5. Namiot D, Sneps-Sneppe M. On Micro-services Architecture[J]. International Journal of Open Information Technologies, 2014, 2(9):24-27.