基于VSOME/IP的汽车E/E架构分布式服务框架设计研究

周辉煌[1]　朱元[1]　毕承鼎[2]　张彪[2]

（1. 同济大学汽车学院，上海 201804；2. 一汽解放汽车有限公司商用车开发院，长春 130011）

【欢迎引用】周辉煌，朱元，毕承鼎，等. 基于VSOME/IP的汽车E/E架构分布式服务框架设计研究[J]. 汽车文摘，2024(4)：10-18.

【Cite this paper】ZHOU H H, ZHU Y, BI C D, et al. Research on Design of Distributed Service Framework for Automotive E/E Architecture Based on VSOME/IP[J]. Automotive Digest (Chinese), 2024(4): 10-18.

【摘要】新型汽车电子电气架构下的车载软件需具备可复用、易扩展、松耦合、兼容互操作等特点。为了将汽车电子控制单元（ECU）上的应用程序抽象为服务，以开源的分布式通信中间件VSOME/IP和远程过程调用框架CommonAPI C++为基础，提出了一种基于VSOME/IP的分布式服务框架。该框架利用Franca IDL服务接口描述语言提高开发人员构建服务效率；通过路由管理器实现了服务框架中服务注册中心组件，为分布式系统提供服务发布、服务订阅、状态同步、消息通知等功能，并采用SOME/IP作为底层通信协议，为系统提供发布订阅式和请求响应式的服务调用方式。

关键词：面向服务架构；VSOME/IP；分布式系统；汽车中间件；服务框架

中图分类号：TP311　　**文献标志码：**A　　**DOI:** 10.19822/j.cnki.1671-6329.20230297

Research on Design of Distributed Service Framework for Automotive E/E Architecture Based on VSOME/IP

Zhou Huihuang[1], Zhu Yuan[1], Bi Chengding[2], Zhang Biao[2]

（1. School of Automotive Engineering, Tongji University, Shanghai 201804; 2. Commercial Vehicle Development Institute, FAW Jiefang Automobile Co., Ltd., Changchun 130011）

【Abstract】In-vehicle software under the new automotive electronic and electrical architecture needs to be reusable, easy to expand, loosely coupled, compatible and interoperable. In order to abstract the application program on automobile ECU into a service, a distributed service framework based on VSOME/IP is proposed based on the open source distributed communication middleware VSOME/IP and the remote procedure calling framework CommonAPI C++. The framework utilizes francadil service interface description language to improve the efficiency of developers in building services; Through the routing manager, the service registration center component in the service framework is realized, which provides services publishing, service subscription, status synchronization, message notification and other functions for the distributed system. SOME/IP serves as the underlying communication protocol to provide the system with publish-subscribe and request-response service invocation methods.

Key words: Service-oriented architecture, VSOME/IP, Distributed systems, Automotive middleware, Service framework

缩略语

SOA	Service-Oriented Architecture
ECU	Electronic Control Unit
AUTOSAR	AUTomotive Open System Architecture
AP	Adaptive Platform
IDL	Interface Description Language
RPC	Remote Procedure Call
OTA	Over-the-Air Technology
ADAS	Advanced Driving Assistance System
IVI	In-Vehicle Infotainment
GPU	Graphics Processing Unit
IO	Input Output
SOME/IP	Scalable service-Oriented MiddlewarE over IP

0 引言

随着基于域控制器的功能集中式汽车电子电气架构的发展[1]，以及为了满足用户对汽车智能化日益变化的需求，面向服务的架构（Service-Oriented Architecture，SOA）的开发方式愈发受到整车制造商和软件供应商关注。SOA的核心是将整车电子控制单元（Electronic Control Unit，ECU）上具有不同功能的应用程序抽象化为独立的服务，并通过服务接口描述语言定义服务，中间件技术在SOA中发挥着至关重要的作用，其可提高服务复用性和互操作性。因此，基于SOA的中间件技术成为车载软件开发的核心要素。

中间件是一类介于应用软件和操作系统之间的软件，其主要功能是对操作系统提供的系统调用接口进行封装整合，同时为应用软件提供应用层的调用接口，以达到复用度高、易于维护的目的。汽车开放系统架构（AUTomotive Open System Architecture，AUTOSAR）组织针对目前车载应用新需求发布了AUTOSAR自适应平台（Adaptive Platform，AP），其符合SOA设计的模式受到多家国内外汽车相关企业认可[2]。AP为应用程序提供了运行环境，其基础功能模块为应用程序提供接口，服务功能模块为应用程序提供服务。现在，国内整车制造商虽然计划在新型汽车电子电气架构的转变下尝试SOA开发方式，但是成熟的AP方案来自于国外且架构昂贵，导致开发成本增加，限制国内智能汽车软件行业的发展[3]。许多软件供应商针对AP标准提出的解决方案，目前比较成熟且可量产的方案有以下4种：

（1）Elektrobit公司推出的EB corbos AdaptiveCore，该平台与基于POSIX的操作系统兼容，在系统性能上达到最高的汽车安全等级[4]。

（2）VECTOR公司推出的Adaptive MICROSAR，提供完整的工具链，包括AP各模块源码，支持AP系统建模、集成验证、代码生成、编码开发等功能[5]。

（3）ETAS联合BOSCH公司共同推出的RTA-VRTE，可选择QNX作为操作系统，提高车载软件的安全性[6]。

（4）国内东软睿驰公司推出自主研发基于AP标准的NeuSAR产品，基于该中间件能快速构建自动驾驶域控制器以及车联网控制器。

由于AP标准发布时间较短并且商用解决方案不对外公开，因此在学术领域相关研究较少。Guissouma等[7]介绍了现代电气电子架构的模型UPDATER（UPDateable Automotive Test dEmonstratoR），展示了如何以高效和敏捷的方式实现空中下载技术（Over-the-Air Technology，OTA）。Kenjić等[8]提出了一种经过验证的自动化解决方案，用于通过车载以太网协议（Scalable service-Oriented MiddlewarE over IP，SOME/IP）在高级驾驶员辅助系统（Advanced Driving Assistance System，ADAS）和车载信息娱乐（In-Vehicle Infotainment，IVI）域之间进行数据交换。Ioana等[9]提出了一种VSOME/IP-OPC UA网关概念解决方案，以确保中间件接口的安全性。Kato等[10]提出了一种在AP平台通过可视化对软件进行功能验证的方法，需要优化的事件通过可视化软件进行展示。扩展了开源的自动驾驶框架（Autoware）软件功能，提供了一套完整的自动驾驶模块，包括定位、检测、预测、规划和控制，并评估了Autoware在基于ARM的嵌入式处理核心和基于Tegra嵌入式图形处理单元（Graphics Processing Unit，GPU）上的性能。Staron[11]等结合了学术研究和行业经验中的理论和实践问题，AUTOSAR软件开发的量产例子、Simulink、构架权衡分析方法（Architecture Tradeoff Analysis Method，ATAM）和ISO 26262: 2018《道路车辆—功能安全》（Road Vehicles-Functional Safety）等重要标准和方法[12]。

国内汽车产业迫切需要具有我国知识产权的中间件技术解决方案。本文提出了一种基于VSOME/IP的分布式服务框架，符合智能汽车开发标准和测试评估标准的发展特点和趋势，能够提供合理的服务质量，具备针对不同功能域提供差异性服务策略的能力[13]。根据网络带宽、网络时延、安全性指标的敏感程度，为功能域提供动态灵活的服务质量配置，使服务能满足智能汽车对于性能和可靠性的要求。构建基于SOME/IP通信协议的分布式服务框架应该包含服务建模、服务发布订阅机制、服务调用、服务路由等核心功能。本文将围绕这些核心功能模块结合CommonAPI C++和VSOME/IP进行设计研究[14]。在二者已有的功能上进行封装改良，提出可以提高系统性能以及兼容性的设计方案，最终实现符合车载端需求的基于VSOME/IP的分布式服务框架：系统显示与查询工具（SOME/IP based Distributed Service Framework，SDSF）。

1 服务模型的建立

1.1 服务模型特征

根据面向服务的特性，服务应该具备以下4个特征：

（1）软件程序：服务是服务提供者所在的机器或

虚拟机中运行的软件程序，可对某个业务逻辑进行抽象封装，为其实现有意义的功能并向服务消费者提供服务接口。

（2）服务标准契约：为了服务提供者和消费者之间能够正常交互，二者之间必须遵守某个标准契约，该契约包含服务接口的描述、服务绑定信息、服务寿命等相关约束。接口定义语言（Interface Description Language，IDL）文件作为服务契约常用的一种形式，通过其定义的服务具有兼容性和互操作性。

（3）可组合复用：多个服务可以组合在一起实现新的服务，将已有的逻辑重新编排形成新的功能，促进了服务的复用。

（4）可发现：服务提供者发布服务后，服务消费者能感知服务存在，通过服务实现彼此之间松耦合关系。因此服务遵循标准化服务契约，具备模块化、可重复性，可被感知发现并调用。标准化的服务契约是构建服务模型的关键，而服务通常以服务接口表示，如 Google 的 Protobuf、Facebook 的 Thrift 以及 GENIVI 的 CommonAPI C++等框架，均采用自定义的 IDL 文件用来描述服务接口。其中 GENIVI 的 CommonAPI C++是基于 VSOME/IP 实现的远程过程调用（Remote Procedure Call，RPC）框架，与 VSOME/IP 之间存在良好的连接性。据此，本文选择 CommonAPI C++作为研究的主要框架。服务模型设计如图 1 所示。

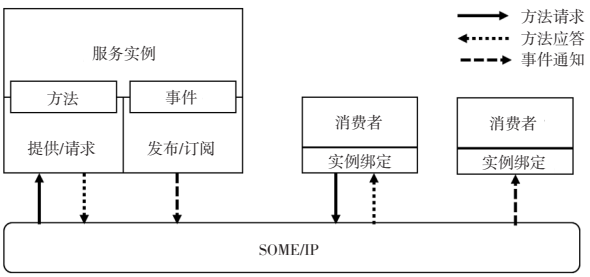

图 1　服务模型设计

1.2　服务模型的设计

在汽车内部复杂的分布式系统中，应用软件除了要满足基于事件的一对多通信方式，以此实现类似汽车传统总线进行广播传递消息的功能，也要提供基于 RPC 的一对一的通信方式，实现软件程序的复用，避免 ECU 功能冗余，有效降低系统耦合性，提高系统稳定性。因此，车载端服务应具备发布/订阅式的通信行为以此满足实时通信需要，并且还能支持请求/响应式的通信行为，使整个系统的算力分配更合理。

服务模型依赖于服务接口设计，一个服务仅能通过一个服务接口定义。根据 fidl 描述语言，该框架的服务接口包含以下 3 种抽象行为和其他描述字段：

（1）方法（Methods）：需确定方法的输入、输出参数数量和类型，服务提供者需要对该接口方法进行实现。服务消费者请求方法调用时，提供者调用本地接口方法并返回结果，实现远程过程调用。

（2）广播（Broadcast）：需确定广播输出的数据类型，服务消费者能订阅广播。服务提供者在相应事件发生后向订阅者发布消息，服务提供者可以同时向多个服务订阅者发送消息，实现事件通知类型的广播通信。

（3）属性（Attribute）：需确定属性的数据类型，属性是方法和广播的结合体。服务提供者需要对属性的接口方法进行实现，服务消费者利用接口方法实现对属性内容的读写访问。属性还能提供与广播类似的事件通知（Notifier），在消费者订阅后并且属性内容变化时，主动通知消费者，实现状态同步控制。

（4）包名（Package）：确定该服务接口的域空间，便于区分管理，对应于 C++程序中的命名空间。

（5）接口名（Interface）：在同一域空间，通过接口名来区分不同的服务，一个接口名下可以包含多个方法、广播、属性。

（6）接口版本（Version）：接口版本与包名组成域空间，接口版本分别由 Major 和 Minor 2 个字段来确定。

1.3　服务接口描述语言

服务接口由 GENIVI 的 CommonAPI C++所提供的接口描述语言 Franca IDL 来定义。Franca IDL 分为 2 种类型：fidl 和 fdepl。

fidl 是对服务接口本身进行逻辑编排，如接口能提供方法、广播、属性等模块。以天气预报服务为例，分别对提供者和消费者的行为进行分析，抽象后的描述文件 weather_serivce.fidl 如图 2a 所示。天气预报服务对应接口有提供者和消费者：

（1）天气预报服务的提供者，提供查询功能以及定时发送天气状态信息，方法名是检查（Check），广播名是通知（Notice）。

（2）天气预报服务消费者能够主动调用该接口提供的 Check 方法，在输入日期后，该天气预报服务返回指定日期的天气情况。在订阅 Notice 广播后，在每晚 8 点被动接收天气预报服务发送的明日天气情况。

根据系统所使用的通信中间件或平台不同，服务接口能够灵活解耦地与不同的通信中间件绑定，以及进行相关部署工作，这一部分工作依赖于 fdepl 描述文件。比如使用 SOME/IP 通信中间件后，必须确定

SOME/IP协议中所包含的方法ID、事件组ID、服务ID以及实例ID等。还是以天气预报服务为例，给出与SOME/IP通信中间件绑定的描述文件weather service_SOME/IP.fdepl，如图2b所示。

```
package demo.weatherService
interface weatherIface {
    version {major 1 minor 0}
    method check {
        in {
            Int32 date
        }
        out {
            String dateweather
        }
    }
    broadcast notice{
        out {
            String tomorrorweather
        }
    }
}
```

(a)天气预报服务接口配置文件1

```
define org.genivi.commonapi.someip.deployment for interface
    demo.weatherService.weatherIface {
    SomeIpServiceID = 1000
    method check {
        SomeIpMethodID = 30000
        SomeIpReliable = false
        out {
            dateweather {
                SomeIpstringEncoding = utf16le
            }
        }
    }
    broadcast notice {
        SomeIpEventID = 33020
        SomeIpEventGroups = {33020}
        out {
            tomorrorweather {
                SomeIpstringEncoding =utf16le
            }
        }
    }
define org.genivi.comonapi.someip.deployment for provider as MyService {
instance demo.weatherservice.weatherIface {
        InstanceId = "demo .weatherservice.node1"
        SomeIpInstanceID = 22000
        SomeIpUnicastAddress  "192.168.199.128"
        SomeIpReliableUnicastport = 30500
        SomeIpUnreliableUnicastPort = 30501
    }
}
```

(b)天气预报服务接口配置文件2

图2　天气预报服务接口配置文件

根据SOME/IP协议要求，本研究定义了一系列服务、接口及事件，包括服务ID、Check方法的方法ID、传输协议、Notice广播事件的事件ID和事件组ID，以及字符串类型的小端序UTF-16编码方式。同时定义了服务提供者的服务实例ID、通信端口IP地址和端口号。根据这2个描述文件，通过CommonAPI C++所提供的代码生成器生成C++语言的Proxy类和Stub类，以及与SOME/IP通信中间件部署相关的类，如图3所示。开发人员利用Stub类实现服务提供的接口方法和事件相关的代码，向外提供服务，然后再通过Proxy类提供的程序接口订阅服务后调用远程方法。开发过程中，开发人员不必将时间精力耗费在底层通信的实现细节上，只需调用生成类对应的接口，着重完成业务逻辑方面的实现，以此提升开发效率。

```
core
└── v1
    └── demo
        └── weatherService
            ├── weatherIface.hpp
            ├── weatherIfaceProxyBase.hpp
            ├── weatherIfaceProxy.hpp
            ├── weatherIfaceStubDefault.hpp
            └── weatherIfaceStub.hpp
```

(a)天气预报服务代码结构1

```
someip
└── v1
    └── demo
        └── weatherService
            ├── weatherIfaceSomeIPDeployment.cpp
            ├── weatherIfaceSomeIPDeployment .hpp
            ├── weatherIfaceSomeIPProxy.cpp
            ├── weatherIfaceSomeIPProxy.hpp
            ├── weatherIfaceSomeIPStubAdapter.cpp
            └── weatherIfaceSomeIPStubAdapter.hpp
```

(b)天气预报服务代码结构2

图3　天气预报服务代码结构

2　服务注册中心的设计与实现

服务注册中心负责管理服务注册与订阅2个主要功能，其保存并持续更新服务相关信息，包括服务接口域名称、服务提供者地址的映射关系、服务调用者订阅服务的相关信息，实现低耦合的通信方式。

2.1　注册中心面临的问题

若分布式系统未配备注册中心，可能出现以下问题：

(1)难迁移扩展：服务之间的交互，无论是通过套接字直接通信还是依托于通信框架，交互双方均需明确对方的IP地址和端口号。若无注册中心，服务需在本地保存对方的地址信息，这属于硬编码方式，即服务预先在本地完成其所依赖服务地址的配置。如图4所示，当服务A的服务实例1由于环境变动而迁移，其地址由地址1变成地址2，依赖于服务A的服务B需手动更新服务实例1的地址信息并重启，期间可能会出现服务停用等问题。为提升服务可用性，同一服务一般存在多个服务实例，这些服务实例被部署到不同的机器上，组成服务集群。服务B作为消费者拥有服务A的所有实例地址信息，服务B能够从中选择合适的服务实例进行通信。然而，当业务增长需要对服务A进行横向扩展(增加服务A的服务实例)时，服务B需手动增加服务A新增实例的地址信息。实际情况中，服务A不只依赖单个服务，如果不采用有效的解决方案，会造成额外的维护成本。

(2)难感知隔离：在分布式系统中，服务实例会不可避免地出现故障，导致该服务实例不可用。此外，在服务升级过程中，服务节点需下线进行升级再重新上线。当这些情况出现时，需要对受影响的服务节点进行隔离。

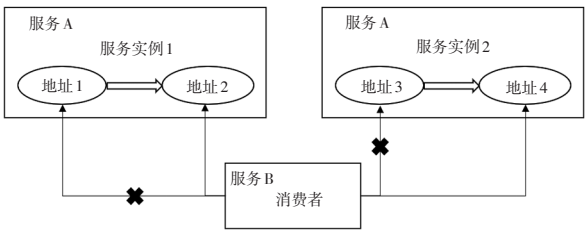

图4　服务实例迁移

如图5所示，服务A依赖于服务B并通过节点2进行请求访问，若节点2出现故障被迫下线隔离，服务A则无法及时感知到节点2不可用状态，造成请求失败，并对服务A的运行状态产生负面影响。即使节点2在下线隔离前可以通知服务A其不可用状态，其他服务仍试图与节点2建立连接，造成资源和时间浪费。因此，在执行服务节点隔离措施时，需保证现有连接中不再接受新的请求访问，并禁止新的连接建立，直至服务B的配置信息被手动更新后重启服务。

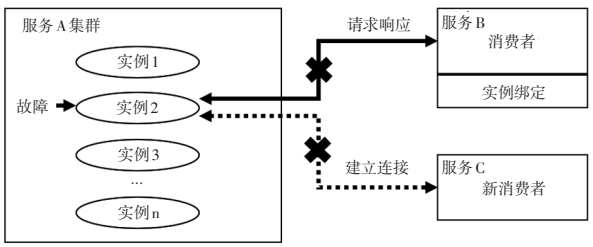

图5　服务实例故障

2.2　服务信息关系

服务注册中心管理的信息对应关系如图6所示。

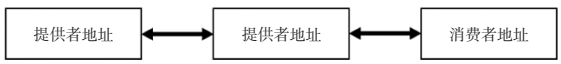

图6　服务信息关系示意

（1）一个服务可以有多个服务提供者，即多个服务实例组成集群，保证该服务的可用性与系统的稳定性。而每个提供者可能具备多个服务能力从而提供多个服务接口。

（2）一个服务能够被多个服务消费者所使用，实现软件程序的高复用性，减少开发成本。而一个服务消费者能够订阅多个服务。

（3）在分析服务提供者与服务消费者之间的地址信息关系时，必须将服务本身作为核心考量，若忽略服务本身，则毫无意义。在双方共享同一服务接口域名称关联的情况下，表现出多对多的关系，即：一个服务提供者的地址信息能够被多个服务消费者所保存。而针对同一服务接口，一个服务消费者能够识别并连接同一个服务的多个服务提供者。

2.3　注册中心功能

为解决上述问题，分布式系统中需要添加服务注册中心，如图7所示，其可提供注册（Register）、订阅（Subscribe）、通知（Notify）、检查（Check）4个核心功能。

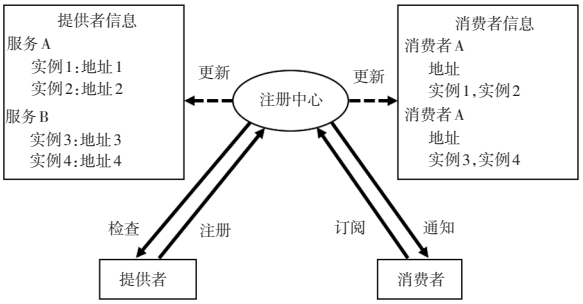

图7　服务注册中心核心功能

（1）注册：接收服务提供者的服务注册或注销请求，保存服务提供者的地址信息以及维护服务接口域名称和服务提供者地址信息的映射关系。

（2）订阅：接收服务消费者的服务发现或停止发现请求，将与服务接口相关的服务节点全部发送给服务消费者。消费者在选择某个节点进行消费后，注册中心存储该消费记录，维护服务接口域名称和消费者地址信息的映射关系，便于遇到突发情况后快速通知。

（3）通知：当服务节点发生变动时，如某节点出现故障或进行升级而隔离或横向扩展时增加某节点，服务注册中心能及时通知关联消费者，使其实时感知服务节点最新状态，避免服务请求失败。

（4）检查：注册中心具备状态查询功能，能及时同步服务节点状态，通常通过心跳检测或者长连接的方式来实现。

2.4　注册中心设计

服务注册中心是基于VSOME/IP中的核心组件——路由管理器来实现。

每个服务应用均配备路由管理器，负责处理通信以及提供服务注册中心的基本功能。如图8所示，VSOME/IP框架中的路由管理器主要分为2类：主路由管理器和代理路由管理器。主路由管理器负责管理同一机器上或虚拟环境中的服务应用通信，且在任何给定时间点，仅允许单个主路由管理器存在，具备主路由管理器的服务应用被称做主服务应用。而同一机器可以存在多个代理路由管理器，除了主服务应用外的其他应用均配备代理路由管理器。主服务应用可以通过配置文件显示指定，若未在配置文

件中指定，则系统默认首次启动的服务应用作为主服务应用。

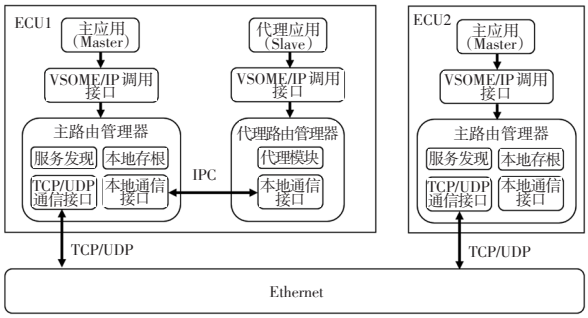

图8　VSOME/IP架构

主路由管理器中包含本地存根和服务发现模块2个模块。本地存根用于本地通信，当服务提供者所提供服务的使用范围仅在本地ECU或服务消费者能够在本地上找到对应的服务实例，本地存根利用本地套接字建立通信端口，负责与本地其他节点进行通信，实现本地的服务注册与发现功能。而服务发现模块用于远程通信，当本地服务提供者所提供的服务需被其他ECU服务消费者所订阅，或者本地服务消费者无法找到满足需求的服务实例时，服务发现模块通过网络套接字建立通信端口，负责与远程其他节点进行通信，实现ECU之间的服务注册与发现功能。

代理路由管理器与主路由管理器中的本地存根类似，只用于本地通信，若其需发布服务或者调用服务，可通过本地套接字将请求发到主路由管理器，主路由管理器中的服务发现模块通过网络将请求发往其他ECU的服务注册中心。同样，网络中的访问请求也只能通过本地主路由管理器转发到代理路由管理器上。这样的优势是其既适用于主服务应用也适用于辅助服务应用，通过使用相同通信接口，有效屏蔽底层通信差异。而主服务应用中的主路由管理器是唯一负责与外部通信的组件，从而简化了网络连接管理并提高其效率。本地通信的实现除了使用本地套接字，也可以使用共享内存等技术。

作为网络通信桥梁，主路由管理器实现的关键要素是构建高效的网络输入输出（Input Output，IO）处理引擎。VSOME/IP的主路由管理器通过引入Boost::Asio异步网络IO库和Boost::Thread线程库，搭建高性能且安全可靠的通信框架，其高效性主要基于2个因素：首先，其结合库中多种IO对象类（如套接字、定时、域名解析器）、IO服务类和线程管理类，实现了基于事件驱动的Reactor或Proactor网络模型以及具有任务调度机制的线程池，为主路由管理器提供强大的网络驱动能力；其次，Boost库中bind和function组件具有灵活性，主路由管理器能够自动感知并处理网络IO上发生的事件（如读写事件或连接事件），实现消息转发功能，并为上层应用提供异步回调机制。作为中间层，网络IO处理引擎实现了网络层与应用层相互隔离，使开发人员更专注于业务逻辑的实现。路由管理器为上层应用提供的核心接口可以提供以下4项功能：

（1）提供服务：路由管理器将服务注册表中服务接口与服务实例地址的映射关系通过C++ STL库中的map容器进行保存。路由管理器先将服务信息保存到本地服务注册表，方便本地其它服务应用通过主路由管理器感知到本地的服务实例，并通过服务发现模块构建SOME/IP-SD报文，将服务信息以广播方式发布给远程机器上的路由管理器。远程机器上的路由管理器接收到该SOME/IP-SD报文后将服务信息保存到远程服务注册表，方便服务消费者能够寻找到该服务实例。

（2）请求服务：路由管理器利用map容器来保存服务实例的订阅信息，即服务订阅表。路由管理器先查看本地服务注册表，若存在本地服务实例，订阅该本地服务实例并将订阅信息保存到本地服务订阅表。若本地服务注册表不存在服务实例，路由管理器查看远程服务注册表，将服务实例信息告知上层应用，上层应用根据服务路由策略选择服务实例并消费，然后路由管理器将该订阅信息保存到服务消费表中并通过服务发现模块构建SOME/IP-SD报文，将订阅信息以广播方式发送给远程服务实例所在的路由管理器。远程的路由管理器接收到该SOME/IP-SD报文后同步远程服务订阅表中该服务实例的订阅信息。

（3）报告服务状态：服务提供者通过该接口，能够主动通知本地路由管理器自身服务状态。当服务状态为不可用时，主路由管理器中的服务发现模块发送停止提供服务类型的SOME/IP-SD报文。订阅该服务实例的服务消费者们通过路由管理器接收该SOME/IP-SD报文并同步该服务实例不可用的状态。

（4）注册可用服务：服务消费者通过该接口，将关注的服务或服务实例结合回调函数注册到路由管理器上，若服务实例的状态发生改变，执行回调函数。当服务实例从不可用状态到可用状态时，服务消费者能够调用该服务实例提供的功能；当服务实例从可用状态到不可用状态时，服务消费者重新寻找其他可用服务实例。该接口还实现了服务注册中心的通知功能，通过将关注的服务实例设置成ANY，当某个服务

集群中节点增加或减少，服务消费者都能立即感知。

路由管理器不仅为上层应用提供交互接口，还具备定期检测已注册服务提供者和消费者状态的功能。当检测到某个服务提供者或消费者上、下线时，路由管理器能够及时向与之关联的对象发出通知。

3 服务调用的设计与实现

服务调用是分布式服务框架中的重要环节。如图9所示，服务消费者借助服务注册中心发现可用的服务实例，然后借助服务路由获得服务实例相关信息并将信息实例化成本地代理类对象，通过代理类对象进行服务调用。

图9 服务调用示意

根据章节1.1所述，服务模型构建中定义了2种通信方式：请求/响应式和发布/订阅式。服务调用也主要分为2类：方法调用和事件通知。根据有无应答能够将方法调用划分为Oneway模式（无应答）和请求/响应模式（有应答），其区别在于服务提供者是否将执行结果返回给服务消费者。还能根据服务消费者在方法调用后是否进入阻塞状态，将方法调用分为同步或异步。

本文服务框架中的服务调用机制基于CommonAPI C++实现。章节1.3中提到，根据服务接口描述文件CommonAPI C++的代码生成器会生成相关的C++类，其中包含应用层面的Proxy类和Stub类，还包含通信中间件层面的SOME/IP部署类。以天气预报服务为例，自动生成的C++代码如图3所示，后文根据该天气预报服务描述SDSF提供的服务调用方式以及实现原理。

3.1 同步方法调用

同步方法调用工作原理如图10所示，调用步骤如下：

（1）服务消费者通过代理对象提供的接口，发起远程方法调用，然后进入阻塞状态。

（2）代理对象通过运行时环境将调用请求序列化成Request类型的SOME/IP报文，发送到远程机器上。

（3）服务提供者收到请求通知后调用存根对象实现的本地方法，并将执行结果序列化成Response类型的SOME/IP报文发给服务消费者。

（4）服务消费者收到返回结果后结束阻塞状态，继续完成后续工作。

假设服务消费者先于服务提供者启动，天气预报服务check方法同步调用时序如图11所示。

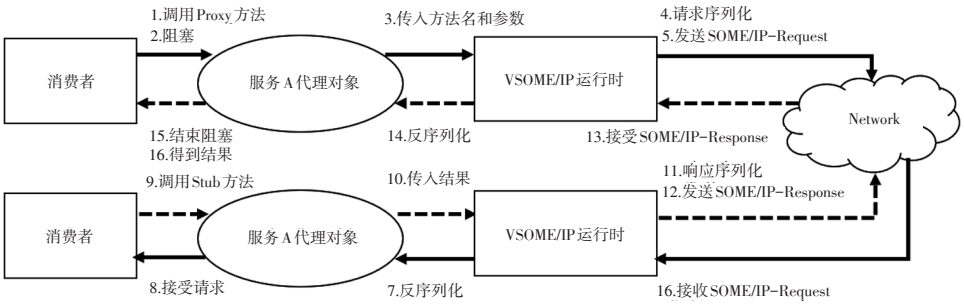

图10 同步方法调用工作原理

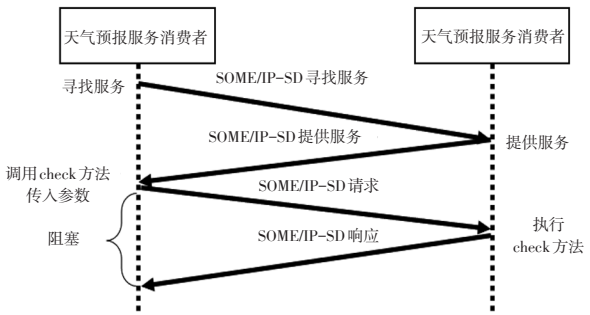

图11 同步方法调用时序

3.2 异步方法调用

异步方法调用相比于同步方式，工作原理和使用较为复杂。异步方法调用工作原理如图12所示，调用步骤如下：

（1）服务消费者通过代理对象调用异步接口并传入回调函数作为参数。

（2）代理对象通过运行时环境将调用请求序列化成Request类型的SOME/IP报文，发送到远程机器上，并返回Future对象给服务消费者，提供给服务消费者能够通过get方法同步阻塞等待调用结果返回的可能性，而在不调用get方法之前不进入阻塞状态，继续执行后续任务。

（3）服务提供者收到请求通知后调用存根对象实现的本地方法，并将执行结果序列化成Response类型的SOME/IP报文发给服务消费者。

（4）请求访问在网络传输过程中，服务消费者端负责管理IO的线程会循环监听通信端口的状态。若收到返回结果，将结果保存到Future对象中，并且监听模块执行对应的回调函数，用来处理返回结果。

假设服务消费者先于服务提供者启动，天气预报服务check方法异步调用时序如图13所示。

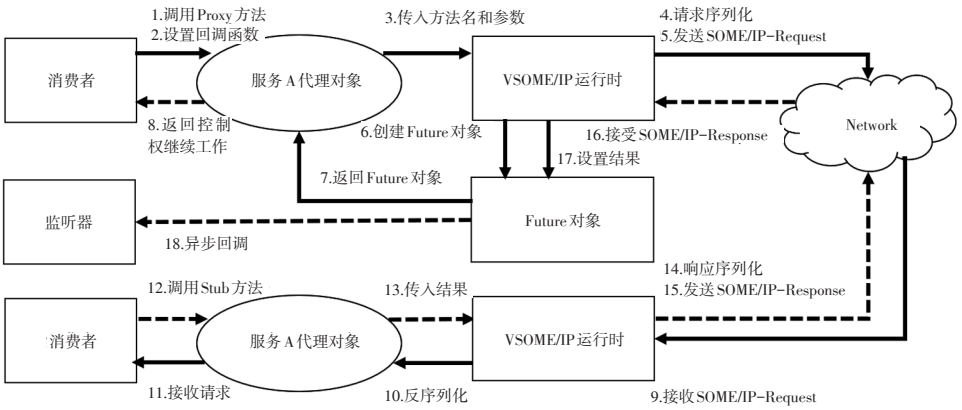

图12　异步方法调用工作原理

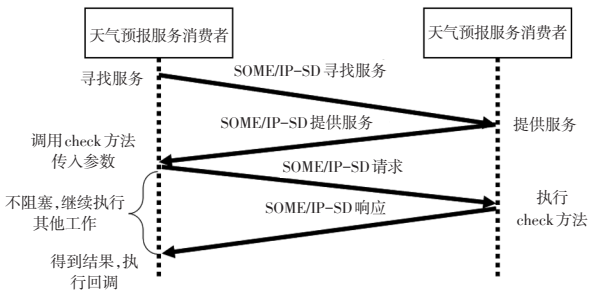

图13　异步方法调用时序

3.3　事件通知

事件通知是属于事件驱动的一种异步服务调用方式。服务消费者订阅服务提供者的话题或者事件，将继续向后执行任务，而服务提供者在适当时间或者当事件发生时向服务消费者发送消息通知。

（1）服务消费者通过代理对象调用事件订阅接口并传入回调函数作为参数。

（2）代理对象通过运行时环境将订阅请求序列化成Subscribe Eventgroup类型的SOME/IP-SD报文，发送到远程机器上。

（3）服务提供者收到订阅通知后记录该服务消费者信息，在事件发生时将通知内容序列化成Notification类型的SOME/IP报文发给所有订阅该事件的服务消费者。

（4）在订阅请求在网络传输过程中，服务消费者端负责管理IO的线程会循环。

（5）监听通信端口的状态，若收到通知消息，立刻执行对应的回调函数，用于处理消息内容。

天气预报服务Notice事件通知时序如图14所示。

3.4　特性分析

针对以上3种类型的服务调用过程和实现原理进行特性分析，如表1所示。

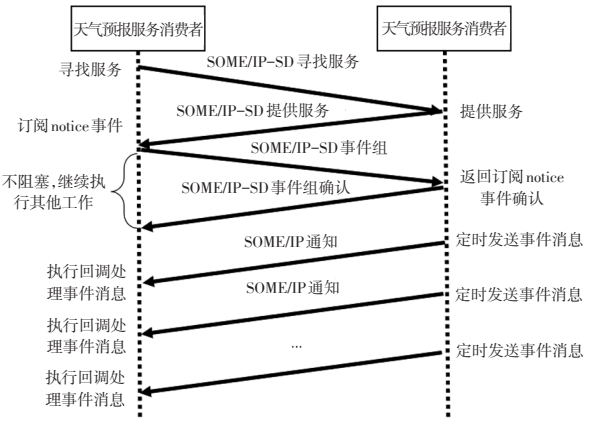

图14　事件通知时序

表1　服务调用特性分析

特性	同步方法调用	异步方法调用	事件通知
发送方式	一对一	一对一	一对多
时间耦合性	紧耦合	松耦合	松耦合
接口耦合性	紧耦合	紧耦合	松耦合
服务独立性	低	低	高
调用复杂度	简单	复杂	简单
响应实时性	高	低	低
调用性能	低	高	高
逻辑处理	简单	复杂	复杂

同步方法调用和异步方法调用均类似于函数调用，需要通过函数标签以及参数列表定义其接口。相比于事件通知，采用同步方法调用通常会使应用程序和服务接口耦合程度更高，若服务接口发生改变，应用程序也必须相应地进行修改。而异步方法调用和事件通知过程是异步的，允许在等待调用结果或事件响应的同时执行其他任务，提升调用效率，且有助于充分发

挥硬件平台性能。然而，由于返回调用结果或事件发生的时间不确定，不利于需要严格实时响应的任务。因此，异步方法调用不适合用于对实时性要求较高的场景。

可根据不同业务场景需求选择合适的服务调用：

（1）若业务场景不关注调用结果，可以选择单向调用或者事件通知。

（2）若业务场景对逻辑处理和响应实时性要求较高，则优先选择同步方法调用。

（3）若业务场景中逻辑块可独立并行处理，不存在依赖关系，则优先考虑异步方法调用或者事件通知。

（4）若需实现一对多的通信方式，或对特定事件和状态进行监控，则优先选择事件通知。

4 结束语

基于VSOME/IP的分布式服务框架设计方案，有效降低了汽车软硬件之间的耦合性，增加车载软件的复用性，提高开发效率。该框架采用SOME/IP作为通信协议，并依托开源的分布式通信中间件VSOME/IP实现基于服务的通信，改善了传统汽车基于信号的通信方式在当下基于域控制器的新型汽车电子电气架构中存在的不足，为车载ECU提供灵活动态的服务通信能力，以适应目前或未来复杂的通信需求。

参 考 文 献

[1] NAN J, LI H, CAO W, et al. Research on Improvement and Experiment for Cyber Security of Automotive Electronic and Electrical Architecture[C]//2022 IEEE 7th International Conference on Intelligent Transportation Engineering (ICITE). IEEE, 2022: 400-405.

[2] FÜRST S, SPOKESPERSON A. Autosar the Next Generation- the Adaptive Platform[J]. CARS@ EDCC2015, 2015: 215-217.

[3] Oertel M, Zimmer B. More Performance with Autosar Adaptive[J]. ATZelectronics worldwide, 2019, 14(5): 36-39.

[4] HOFFMEISTER K. Automated Driving Necessary Infrastructure Shift[J]. ATZelektronik worldwide, 2016, 11(1): 42-47.

[5] OERTEL M, ZIMMER B. More Performance with Autosar Adaptive[J]. ATZelectronics worldwide, 2019, 14(5): 36-39.

[6] ZERFOWSKI D, BUTTLE D. Paradigm Shift in the Market for Automotive Software[J]. ATZelektronik worldwide, 2019, 121(9): 28-33.

[7] GUISSOUMA H, HOHL C P, LESNIAK F, et al. Lifecycle Management of Automotive Safety-critical over the Air Updates: A Systems Approach[J]. IEEE Access, 2022, 10: 57696-57717.

[8] KENJIĆ D, ŽIVKOV D, ANTIĆ M. Automated Data Transfer from ADAS to Android-based IVI Domain over SOME/IP [J]. IEEE Transactions on Intelligent Vehicles, 2023, 8(4): 3166-3177.

[9] IOANA A, KORODI A. VSOMEIP-OPC UA Gateway Solution for the Automotive Industry[C]. 2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), 2019: 1-6.

[10] KATO S, TOKUNAGA S, MARUYAMA Y, et al. Autoware on board: Enabling Autonomous Vehicles with Embedded Systems[C]. 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), 2018: 287-296.

[11] STARON M, DURISIC D. Autosar Standard, Automotive Software Architectures: Springer, 2017: 81-116.

[12] ISO. Road Vehicles- Functional Safety: ISO 26262: 2018.[S]. Switzerland: ISO Copyright Office, 2018.

[13] BELLANGER M, MARMOUNIER E. Service Oriented Architecture: Impacts and Challenges of An Architecture Paradigm Change[C]. 10th European Congress on Embedded Real Time Software and Systems (ERTS 2020), 2020.

[14] ANGGORO W, TORJO J. BOOST. Asio C++ Network Programming[M]. Packt Publishing Ltd, 2015.

（责任编辑 **梵铃**）

【作者简介】

周辉煌（1999—），男，同济大学，硕士研究生，研究方向为汽车电子嵌入式软件。

E-mail：zhouhuihuang78@gmail.com

朱元（1976—），男，同济大学，副教授，研究方向为新能源汽车电机控制技术、汽车电子嵌入式软件。

E-mail：yuan.zhu@tongji.edu.cn