基于微服务架构的统一应用开发平台①

李春阳[1], 刘 迪[2], 崔 蔚[1], 李晓珍[1], 李春岐[3]

[1](国网信息通信产业集团有限公司, 北京 100031)
[2](北京中电普华信息技术有限公司, 北京 100192)
[3](天津市普迅电力信息技术有限公司, 天津 300192)

摘 要: 随着信息化时代的发展, 业务系统越发庞大复杂, 系统开发及维护面临着巨大的挑战. 针对这一问题, 提出基于微服务架构的统一应用开发平台, 重点介绍了平台的微服务架构设计、基于平台的业务系统实现. 通过引入微服务构建和分布式服务注册等相关技术, 平台实现了生成微服务工程的标准开发框架, 解决传统单体架构应用庞大而带来的研发周期长, 难以快速响应用户需求等问题, 为业务系统的开发提供了有效支撑.

关键词: 微服务; 开发平台; 分布式服务注册中心

Unified Application Development Platform Based on Micro-Service Architecture

LI Chun-Yang[1], LIU Di[2], CUI Wei[1], LI Xiao-Zhen[1], LI Chun-Qi[3]

[1](State Grid Information & Telecommunication Industry Co. Ltd., Beijing 100031, China)
[2](Beijing China Power Information Technology Co. Ltd., Beijing 100192, China)
[3](Tianjin Puxun Power Information Technology Co. Ltd., Tianjin 300192, China)

Abstract: Business systems become larger and more complex with the development of information. System development and maintenance are facing enormous challenges. To solve this problem, this paper proposes an application development platform based on micro-services architecture, focusing on the micro-service architecture design and business system practice. This platform implements a standard development framework to generate micro-services engineering, by introducing micro-services building technology and distributed service registry technology. These technologies help to solve problems such as the long development cycle, difficulty to quickly respond to the needs of users and provide a strong support for the system development.

Key words: micro-service; development platform; distributed service registry

传统应用架构的弊端最早在大型企业和互联网行业中呈现, 这些公司都遇到了复杂应用的开发维护成本变高、代码重复率增大、团队协作效率变差、系统可靠性变低、系统水平扩展困难、新功能上线周期变长等问题. 因此众多大型公司经过了反复实践和尝试, 推出了各种轻量级的架构模式, 有效的解决了上述问题.

国家电网公司目前的业务应用系统是按照传统应用架构搭建的, 但是随着企业应用的不断深化和业务数据的几何级增长, 业务用户对应用系统提出了越来越高的要求, 在这样的背景下传统的应用架构已无法满足公司信息化发展的需要, 很多问题已经初露端倪, 其中包括: 1)传统的企业应用代码庞杂并且业务组件之间耦合程度非常高, 造成了业务应用维护难度大、升级成本高; 2)业务模块之间的循环依赖、不合理的调用、冗长复杂的业务流程等问题对新功能的上线造成极大影响; 3)系统功能组件出现无法恢复的故障时, 整个节点处于不可用状态; 4)在扩展性方面, 由于传统应用大多是单一部署, 在这种模式下一个应用某些部分偏 I/O 密集型、某些部分却偏 CPU 密集型, 但应用却只部署在一台机器上, 很难用单一硬件来满足应用各部分对硬件资源的不同要求, 造成无法高效的应对多

① 收稿时间:2016-07-26;收到修改稿时间:2016-10-12 [doi:10.15888/j.cnki.csa.005757]

用户高并发的场景等问题.

为解决现阶段信息系统建设中存在的问题,本文推出了全面支持微服务的开发框架[1], 可将高度耦合的功能分解到各个离散的微服务中以实现对应用系统的解耦. 该架构全面支持了轻量级嵌入式应用容器、轻量级 IOC 组件、去中心化的服务注册中心、高性能的远程过程调用、应用的追踪与监控、分布式会话管理、负载均衡策略及应用可靠性保障等特性.

1　平台简介

平台按照“开发标准化、系统模块化、操作工具化、运行容器化, 应用服务化”的总体目标, 把微服务开发框架、相应的技术和工具整合到平台中, 平台主要包括集成开发工具、基础服务框架、应用开发套件三大功能组件. 总体功能架构如图 1 所示.

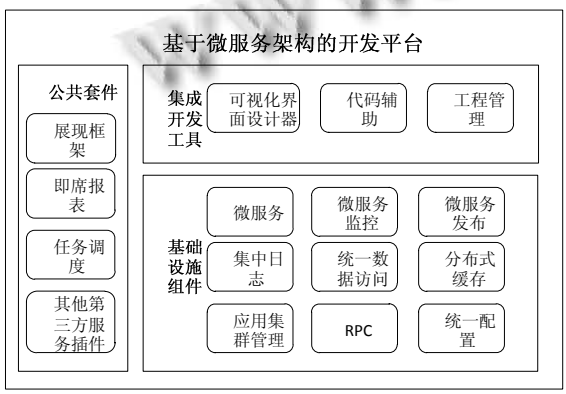

图 1　平台功能架构

集成开发工具主要包括可视化界面设计器、编码与调试辅助工具等与业务系统开发密切关联的功能组件; 基础服务框架用于屏蔽大量的底层技术细节,提供了微服务集群管理、远程过程调用、分布式服务框架、分布式缓存、集中日志等基础性技术组件; 应用开发套件主要包括 MX 展现框架、即席报表组件等高级开发套件, 基于这些套件可以快速开发出业务系统功能逻辑, 保障业务系统稳定、高效运行. 本文主要介绍平台的基础微服务框架.

2　平台微服务架构设计

2.1 总体架构

平台微服务开发框架基于约定优于配置的思想, 封装了支撑微服务构建的组件库, 不需要繁琐的配置即可使用 Java 语言开发微服务. 平台总体技术架构如图 2 所示.

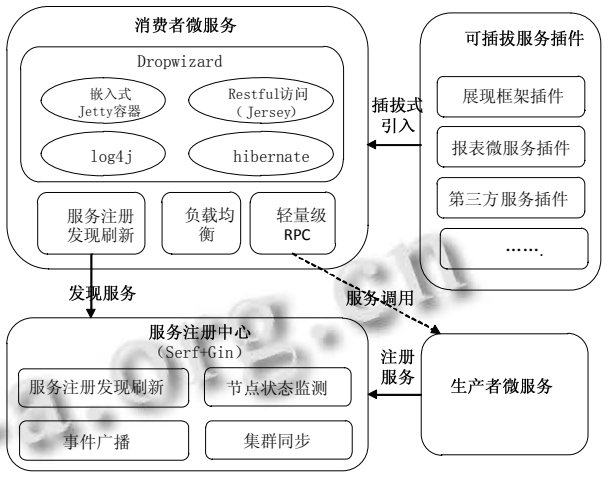

图 2　平台总体技术架构

平台基于 dropwizard 提供微服务开发组件, 以 Jetty 作为 Web 容器, 引入 Jersey 提供标准 Restful 风格的 Web 访问, 集成 Hibernate 等数据持久层访问组件; 提供服务注册发现功能, 自动注册本地服务到注册中心; 通过可选的负载均衡策略, 选择最优的服务地址; 提供轻量级的 RPC 组件调用, 保障高可用的服务调用.

在开发阶段根据微服务框架 Bundle 特性, 能够实现平台各模块的 Bundle 插件集成到微服务中, 如任务调度、大数据组件、报表和展现框架等.

同时, 平台提供了去中心化的分布式服务注册中心[2], 在每个节点上启动一个注册中心, 不同节点的多个注册中心之间没有主次之分, 降低了注册中心的压力, 其中一个注册中心宕机, 不会影响集群中的其它注册中心. 针对注册中心节点的宕机问题, 提供了自动检测失败节点机制及周期性地恢复功能, 为注册中心正常运行提供双重保障.

2.2 微服务开发框架

2.2.1 微服务工程

微服务工程有三种, 服务生产者, 服务消费者和服务定义接口(API), 服务由接口定义, 服务生产者实现接口服务, 服务消费者来调用. 工程结构[3]如图 3 所示.

消费者和生产者微服务通过扩展微服务开发组件实现. 微服务开发组件是对开源微服务组件 dropwizard 的二次封装, 除了具有 dropwizard 提供的

特性外, 还封装了服务注册, 发现, 刷新, 轻量级 RPC, 负载均衡. 由以下核心模块[4]协助完成.

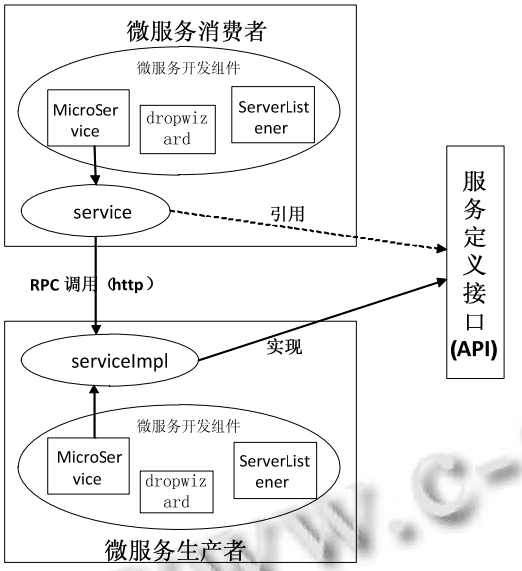

图 3 微服务工程结构

服务实现: 通过自定义注解@MircoService, 将 RPC 调用,负载均衡做了封装, API 接口的实现类和引用类必须注解为@ MircoService 类型.

服务交互监听器: 平台提供 ServerListener 类, 系统启动时加载该监听器, 实现服务注册刷新和发现.

(1) 服务实现

提供注解@Microservice, 将类标记为微服务的业务处理逻辑构件. 标记方式有两种:

① 作为接口引用的标记

```
public class TestFrontController {
private static final Logger logger = LoggerFactory.getLogger(Bug4TestFrontController.class;
    @MicroService //标记为接口引用的微服务构件
        private IBugService service;
    }
```

② 作为接口实现的标记

```
    @MicroService //标记为微服务构件
    public class TestService implements ITestService{
private static final Logger logger = LoggerFactory.getLogger(BugService.class);
        @Inject
        private BackDAO dao;
        @Override
        public String query(String params) {
            return dao.query(params);
        }
    }
```

(2) 监听器 ServerListener

ServerListener 在系统启动时加载, 主要有以下功能.

① 服务注册

发送 http 请求向服务注册中心, 注册服务数据, 注册服务数据为 json 格式:

```
{
addr: http//localhost:8080/ms,
    Provider:
[com.test.api.service, com.test.api.service2],
    Consumer: []
}
```

微服务如果是生产者, 则provider值不为空, 是消费者, 则 Consumer 值不为空.

② 服务刷新和发现

定时向注册中心发送刷新服务请求, 刷新请求返回的是服务的状态和路由列表, 如果服务状态为死亡则重新注册服务, 如果服务正常, 判断本次路由表校验码与本地路由表校验和是否一致, 不一致更新本地路由表.

2.2.2 去中心化的服务注册中心

传统的企业应用架构多采用中心化的分布式服务架构, 一旦服务注册中心出现问题, 整个系统就会瘫痪; 同时所有应用向一个服务注册中心进行远程注册, 会消耗大量的网络资源, 造成服务注册中心压力过大. 因此, 平台采用去中心化的服务注册中心, 如图 4 所示.

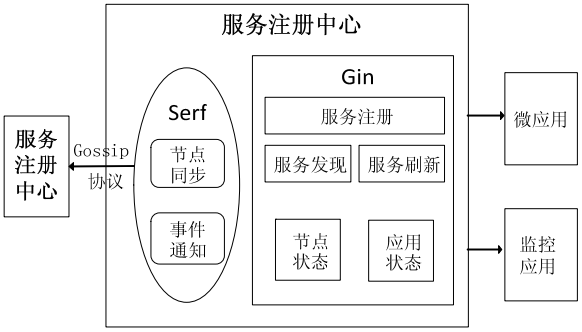

图 4 去中心化的服务注册中心

服务注册中心封装了开源分布式集群框架 Serf, 基于 Gossip 协议实现集群节点之间进行通信, 同步所有的路由信息; 同时封装 go 语言的 Web 服务框架 Gin, 实现服务发现[5], 注册, 刷新的 http 接口服务, 供微服务和监控应用调用. 节点之间, 以事件的方式提供了信息同步功能, 实现节点信息、路由信息的同步.

(1) 服务注册

采用 Gin Web Framework, 对外提供服务注册的 Restful 服务. 接收微服务传递来的注册信息, 包含: 微服务访问地址、提供哪些服务的列表、消费哪些服务的列表, 然后保存到本地的哈稀表中.

接口地址: POST/GET http://注册中心地址:端口/msd/register

参数:

{

addr:http://192.168.0.245:8082/ms,

providers:[com.sgcc.uap.hello.api.IHelloService],

consumers:[]

}

返回值: http Code

(2) 服务刷新

服务注册中心接收到服务发现的 Rest 请求后, 从本地的哈希表中获取所有微服务的注册信息, 拼装成数组, 然后再对所有注册信息计算 MD5 校验码, 将数组和校验码一并返回. 服务发现器接收返回数据后, 将注册信息数组保存到微服务容器的哈希表中, 作为服务提供者的列表.

接口地址: POST/GET http://注册中心地址:端口/msd/refresh

参数: 微服务 http 访问地址的 base64 编码值

返回值:

{

AppStatus:true,

chckNum:h#$!@#@#hweflkj 21dja!$jild;

}

(3) 服务发现

服务注册中心接收到该 Rest 请求[6]后, 解码 URL 中的地址参数, 去本地的哈希表中查询该地址是否存在, 如果存在, 服务注册中心认为该微服务存活, 否则认为微服务已经掉线. 当微服务处于存活状态时, 还要从本地的哈希表中获取所有微服务的注册信息后计算 MD5 校验码, 最终将存活状态信息和校验码一并返回. 服务刷新器接收返回数据后, 判断存活状态为掉线时, 去调用服务注册器重新注册, 然后拿本地保存的校验码与返回的校验码比对, 不一致时, 表明服务注册中心的注册信息发生变化, 调用服务发现器去注册中心重新获取, 保证微服务容器的服务提供者列表是最新的数据.

接口地址: POST/GET http://注册中心地址:端口/msd / fetch

参数: 微服务 http 访问地址的 base64 编码值

返回值:

{

"service":"com.sgcc.bug.IBugService",

"addrs": [

{"name":"dgo-dev","addr":"http://192.168.20.3:8084/ms"},

{"name":"dgo-dev2","addr":"http://192.168.20.4:8083/ms"}

],

"checksum":"yposdfpapsdfpwerdaf=yxl$3"

}

(4) 节点同步

服务注册中心节点之间通过 Gossip 协议进行周期性的消息通信[7], 传递节点信息, 保证最终所有节点的路由信息一致. 全局节点状态信息同步发生在新添加节点、全局同步周期. 报文格式如下:

Version (byte) | Nonce (12 bytes) | CipherText | Tag (16 bytes) | Message Type (4 bytes)

Version: 目前总是设置为 0, 允许未来改变报文使用的算法后改变版本值.

Nonce: 随机数, 保证消息的完整性

CiperText: 密文, 消息主体

Tag: 标记, 校验消息的完整性

Message Type: 信息类型, 使用大端格式(Big Endian format)编码.

3 平台实现与应用

3.1 启动注册中心

开发微服务之前, 首先要启动本地注册中心程序, 以支持开发环境中不同微服务之间的通信. 在 Windows 中执行 cmd 命令进入 Windows 控制台环境,

找到平台提供的 blued.exe 组件所在的目录，输入命令 blued agent 启动微服务环境，如图 5 所示.

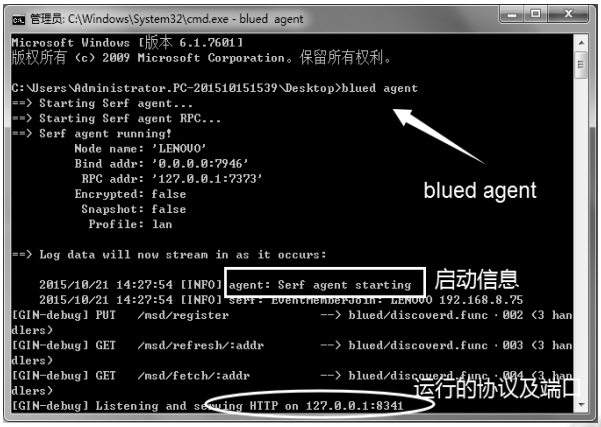

图 5 启动注册中心

3.2 微服务项目创建及运行

(1) 创建微服务示例项目，在向导中输入项目名称，并配置数据库信息，如图 6 所示.

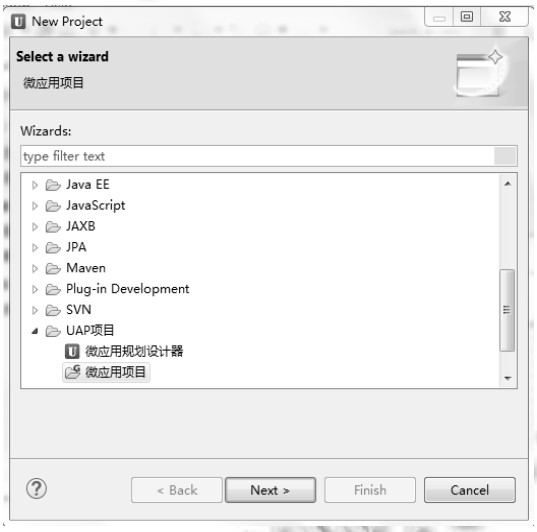

图 6 创建微服务示例项目

(2) 完成项目创建后，在当前工作空间中创建三个代码工程组件: API 项目，定义了微服务的项目 API 接口; Client 项目，微服务客户端项目; Server 项目，微服务服务端项目. 工程结构如图 7 所示.

(3) 运行微服务项目. 选中项目，单击右键，选择 Gradle build...选项. 此时内置的 Gradle 插件会对项目做解析，如图所示，右下角有解析进度. 随后输入 run 即可启动项目.

(4) 打开浏览器，验证服务是否正常运行.

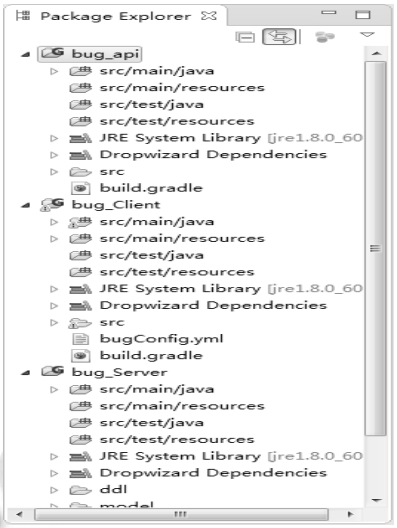

图 7 微服务工程结构

图 8 运行微服务项目

图 9 服务运行效果

4 平台优势及应用情况

平台提供了具有容错性和高可用性的去中心化的分布式服务注册机制，不需维护一个高可用的服务注册中心，而是将其分散到集群中的每个节点. 在每个节点上启动一个注册模块，不同节点的多个注册模块之间没有主次之分，降低了注册模块的压力，而且其中一个注册模块宕机，不会影响集群中的其它注册模块，有效避免了只有一个注册模块的宕机而导致系统瘫痪的风险. 针对注册模块节点的宕机问题，提供了自动检测失败节点机制及周期性地恢复功能，为注册模块正常运行提供双重保障.

同时平台微服务开发框架提供了本地化的服务注册和发现功能, 服务只需要向本地的注册模块进行注册. 集群中不同注册模块之间同步注册信息, 同步过程无需应答, 有效减少了网络资源的消耗. 从本地注册模块取到的注册信息即为集群中注册的所有服务信息, 供微服务之间调用.

本平台自发布以来, 已在国家电网公司包含基建管控、协同研发工具在内的多个重点项目中进行了应用实践. 基于微服务架构的统一应用开发平台具备去中心化的服务注册、高性能的远程过程调用框架、稳健的负载均衡策略, 大幅提升了业务系统开发和维护的效率, 有力地提高了系统的可扩展性和可靠性.

5 结语

本文研究并实现了基于微服务架构的统一应用开发平台. 平台提供生成微服务的标准开发框架, 屏蔽了复杂的技术细节, 研发人员只需关注业务代码的编写和微服务的配置; 平台的微服务框架强化了系统的模块化结构, 在该架构下每一个业务模块都是一个可以独立部署和运行的单元, 模块间以消息驱动 API 的形式定义了清晰的界限; 同时, 平台为微服务封装了服务注册器、服务发现器、服务提供者列表和服务注册中心. 通过平台提供的微服务开发框架, 可以快速建立起一个高内聚、低耦合的微服务应用, 达到“开发标准化、系统模块化、应用服务化”的目标.

参考文献

1 王磊.微服务架构与实践.北京:电子工业出版社,2015.
2 李林锋.分布式服务框架原理与实践.北京:电子工业出版社,2016.
3 温昱.软件架构设计.北京:电子工业出版社,2015.
4 纽曼(Sam Newman)微服务设计.崔力强,张骏,译.北京:人民邮电出版社,2016.
5 李勇.分布式 Web 服务发现机制研究[博士学位论文].北京:北京邮电大学,2007.
6 顾志峰,李涓子,胡建强,许斌,王克宏.Web 服务之间数据关联的建模与应用.计算机学报,2008,31(8):3–21.
7 Videla A, Williams JJW,汪佳南.高效部署分布式消息队列.北京:电子工业出版社,2015.