计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(7):121−125 [doi: 10.15888/j.cnki.csa.005829]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于Seneca微服务的过程监测系统①

吕海东, 葛日波, 朱志刚

(大连理工大学 城市学院, 大连 116600)
通讯作者: 吕海东, E-mail: haidonglu@126.com

摘　要: 针对传统监测监控与数据管理应用系统架构复杂, 前后端开发技术不一致, 通讯协议不统一, 难以实现高并发请求的问题, 在高性能服务器平台Node.js下, 使用全新的微服务架构和Seneca框架技术与监测监控技术相结合, 使用统一的Node.js编程模式, 标准化的REST API通讯协议, 实现了低成本、高性能、易维护和安全性的监测与数据管理的企业级应用.

关键词: 微服务; API网关; 工业过程监控系统; Node.js; Seneca框架

引用格式: 吕海东,葛日波,朱志刚.基于Seneca微服务的过程监测系统.计算机系统应用,2017,26(7):121–125. http://www.c-s-a.org.cn/1003-3254/5829.html

SCADA Based on Seneca Micro-Service

LV Hai-Dong, GE Ri-Bo, ZHU Zhi-Gang

(City Institute, Dalian University of Technology, Dalian 116600, China)

Abstract: In view of the complexity of traditional industry process monitoring system(SCADA) architecture, the inconsistency of the development technology and communication protocol between front and backend of application which could not process high concurrent requests, the innovated application which combines the SCADA and microservice framework Seneca based on Node.js server platform is developed which uses unified Node.js programming model and standardized REST API protocol. The application has low cost, high performance, easy maintenance and security of SCADA enterprise.

Key words: micro-services; API gateway; SCADA; Node.js; Seneca framework

当前在开发具有监测监控和数据管理相结合的软硬件应用系统时, 通常都在数据采集层和数据管理层采用不同的技术和编程语言. 在数据采集端一般使用专门的工业控制计算机, 使用C语言甚至是汇编语言完成对现场传感器数据的采集, 而在上位机服务器端普遍使用基于JavaEE的软件架构技术, 如使用Hibernate, Spring和Spring MVC等, 这就导致前端和后端开发技术不一致, 需要熟悉不同技术的开发人员, 导致项目投资大, 编程难度大, 开发周期长, 项目难以维护等诸多缺陷.

如果数据采集前端和数据管理后端采用相同的软件开发技术和编程语言, 将极大的简化此类软件系统的开发, 减低不同软件技术协作的复杂性. 基于JavaScript编程语言的Node.js平台出现, 使该需求的实现成为可能.

为实现能进行超大量并发请求处理的应用开发, 克服传统服务器平台的难以支持超大并发连接的实时应用, 以全新模式工作的服务器平台技术Node.js[1]得以快速发展. Node.js的单线程、非阻塞、异步响应处理、事件驱动的特性, Node.js同时支持在微型设备和

① 收稿时间: 2016-10-18; 收到修改稿时间: 2016-11-21

大型服务器上运行, 微型设备如树莓派, Arduino, Intel Galileo Gen, Intel Edison Arduino, BeagleBone Black等, 使得其成为当今开发面向工业过程监控、物联网、移动应用和企业级应用的首选.

另一方面软件开发最新的架构技术-微服务[2] (Micro services), 由于其能使各类系统应用的开发得以全面的简化, 加快其应用的部署和维护, 克服了传统单一结构模式的软件系统的固有的缺陷, 成为当前软件开发的最流行技术之一.

微服务是互联网云计算时代传统软件系统从单一整体架构向分布式微组件结构转变的必然选择. 通过将传统软件系统的功能拆分为可独立开发和部署的微服务, 提高了软件应用系统的可靠性、可维护性和可伸缩性, 能适应当前以移动客户端为主要访问模式的超大量并发请求处理, 并实现快速响应, 以满足移动客户的对系统性能的极高需求.

基于Node.js的微服务的发展更是以惊人的速度在进行, 众多基于Node.js的微服务框架纷纷出现, 其中佼佼者Seneca[3]以其简单的REST API[4]实现模式, 由于其同时支持HTTP和TCP通讯协议等诸多优点开始得到了认可和应用.

本文结合过程监控和微服务技术, 利用Node.js服务器平台, Seneca微服务架构, 完美实现了一个超低成本、高性能的全新模式的监测监控与业务管理全集成的企业级管理系统, 并在某大型分布式CNG加气站管理系统中应用, 从气罐、加油机监控, 加气数据传输, 加气卡结算, 采购, 销售和库存管理等实现全程信息化处理.

1 系统总体架构设计

整个系统采用分布式架构, 包括中央服务器、监控下位机、管理PC客户端、移动客户端, 实现从监控、数据传输、业务管理的全覆盖, 其系统总体架构参见图1所示.

系统建设初期采用本地的服务器, 未来可升级为云主机. 服务器采用Ubuntu Server15.04[5]操作系统, 安装Node.js的最新版V5.1, 数据的存储采用MariaDB10[5], 微服务实现使用Seneca.

储气罐和管道的监控使用西门子PLC S3-700, 通过树莓派2B+[6]读取PLC的监测数据[7], 定时调用服务器端的微服务API传输到中央服务器数据库中. 树莓派运行Linux操作系统Debian, 同样安装Node.js, 由于上位服务器和监测下位机相同的开发和运行环境, 简化系统的开发成本, 加快开发进度.

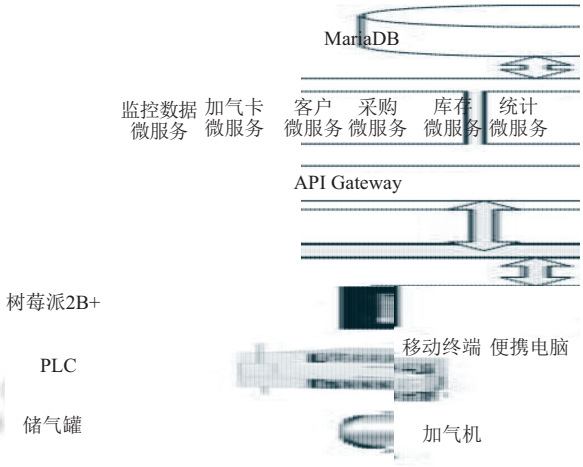

图1 系统总体架构图

2 系统微服务设计与实现

系统采用基于Node.js的微服务框架Seneca实现所业务所需的微服务. Seneca的如下优点极大简化了微服务的开发的部署.

(1) 支持多种通讯模式, 包括REST API, TCP, 消息队列, 发布/订阅模式等.

(2) 数据的传输全部采用JSON格式, 减少了网络传输数据量, 节省了网络带宽.

(3) 微服务采用模式匹配加响应工作, 简化了微服务的查找和调用, 克服了传统的微服务的复杂的注册和服务机制, 极大简化了微服务的部署.

整个系统无论是数据采集客户端还是数据管理服务器端, 都采用Seneca框架将各种功能发布为基于REST API的微服务, 通过这些API采集端和服务器端都可以进行双向的数据通讯, 并使用标准的JSON数据格式进行数据的传输, 极大简化了系统的开发难度.

为简化系统的投资, 本系统没有采用昂贵的工业控制PC机作为下位监测端, 而是使用廉价的用于物联网系统的树莓派卡片式计算机, 在其内置的Linux平台支持下, 安装了Node.js和Seneca框架, 使用专门用于Node.js平台的连接各种传感器的数据采集框架Johnny-Five. 通过Johnny-Five内置的各种数据采集模块, 将采集的数据转换为JSON格式, 使用Seneca的微服务发布为REST API, 任何终端包括服务器管理层, 手机或平板, 甚至其他软件系统都可以通过标准的REST API接

口读取监测的数据, 或发送执行动作的指令给连接的各种执行器, 如阀门, 步进电机等. 数据采集和控制端系统的微服务结构参见图2所示.

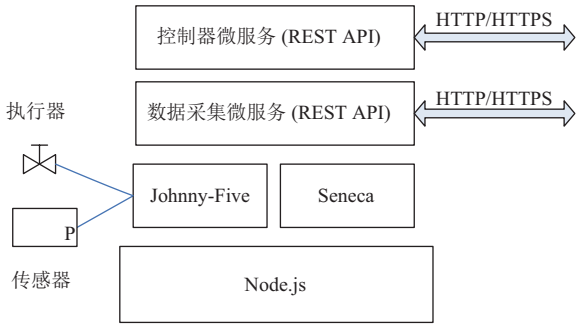

图2 下位机采集端微服务结构

服务器处理端同样基于Node.js平台, 与数据库的连接服务采用了支持ORM功能的Sails.js框架, 该框架实现了MVC模式, 极大简化了数据库编程. 微服务框架Seneca调用Sail.js框架, 对外发布为支持REST API的微服务, 可供各种客户端访问, 如Web, 手机, 其他物联网设备等, 服务端的微服务设计架构参见图3所示.

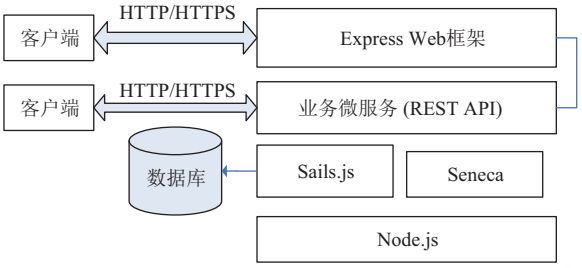

图3 服务器端微服务架构设计

每个微服务通过Seneca的role属性实现服务的分类, 属性cmd则定义服务的名称. 当微服务的客户通过seneca实例传入指定的模式信息后, Seneca根据模式匹配定位到指定的微服务, 执行回调函数, 通过msg取得传入的数据参数, 使用respond对象返回微服务操作结果.

Seneca微服务可以通过多种方式调用. 在Node.js进程内部以及不同Node进程之间, 都可以通过Seneca的客户端实现.

使用Seneca的client对象的方法act对指定的微服务进行调用, 并且调用时依旧使用Node.js的异步非阻塞模式, 得以实现超大量客户并发请求的快速处理.

Seneca的微服务默认提供标准的RESTful API模式调用, 可以通过HTTP或HTTPS对微服务进行请求.

如果分布式系统都采用Node.js进行开发, 则直接使用Seneca的client对象对微服务进行请求调用, 而采用其他技术时, 可以使用HTTP REST API进行请求, 如移动客户端的Android和iOS手机. 无论哪种请求方式微服务的返回结果都是JSON[8]数据对象, 方便客户端进行数据的解析和处理.

系统中功能都采用微服务实现, 核心微服务包括气罐监测数据、加气机数据监测、会员管理、加气卡管理、CNG采购管理、CNG库存管理等.

系统中Web客户端使用jQuery通过REST API HTTP请求微服务完成对客户的管理业务.

由于定义的微服务较多, 系统采用Node.js的模块机制将每个微服务定义在单独的JavaScript文件内, 再使用Node.js和Seneca的模块载入机制实现微服务的部署, 其示意加载和启动实现代码如下:

```
var seneca = require('seneca')();
seneca.use(require('./CustomerService.js'))
        .use( require('./CardService.js'))
.use( require('./GasDataService.js'))
        ..../载入其他微服务
        .listen();
```

3 微服务API Gateway[9]设计与实现

在基于微服务架构的企业级应用开发中, 为提高系统的安全性、可维护性和可伸缩性, 基本上都采用微服务API网关来接收客户的请求, 将微服务与外部客户端进行隔离, 避免客户端与微服务进行直接的相互通讯, 保护微服务避免受到各种攻击. 另外当微服务发生改变时, 只修改API Gateway, 不需要客户端进行任何修改. 所有客户端都与API网关进行通讯, 由网关实现动态均衡负载, 再定位到系统中指定的微服务.

本系统采用Senaca和Express[10]整合, 实现微服务API网关的Web接口, 以便微服务调用能通过企业内部的防火墙, 提供微服务调用的对外端口. API网关封装所有的微服务调用, 如数据采集、加油卡, 业务处理等, 并将多个微服务的响应结果进行整合以单一结果发送回客户端.

为演示一个微服务网关API的实现过程, 下面以加气站客户管理业务微服务API网关的实现加以说明, 该

API网关将代理实际微服务, 供外部客户端访问, 并可以进行安全性验证.

首先使用Node的模块机制, 定义API网关的功能函数, 并在函数内部调用微服务公布的功能, 其简要实现代码如下所示:

```
module.exports = function api( options ){
this.add({role:“api”, path:“customer”}, function (msg, respond){
client.act({role:“customer”, cmd:msg.action, data:msgdata}, function(error, result){ respond(null, result); });
});
```

实现API网关函数后, 将其/发布为Web REST API调用, 实现与Web集成, 这样任何客户端都可以使用Web访问协议HTTP或HTTPS通过API网关对指定的微服务进行访问.

```
this.add(‘init:api’, function( msg, respond ) {
    this.act(‘role:web’, {use:{
        prefix: ‘/api’, pin:‘role:api, path:*’,
        map: {customer: { GET:true, suffix:‘/:action’ }
        } }}, respond )
    });
};
```

将微服务的API网关的实现代码, 通过Seneca和Express的整合, 即可实现REST API模式的微服务. 最后使用Express支持的中间件功能, 将Seneca的微服务和Express Web服务进行整合, 实现微服务的REST API访问协议.

4 数据采集客户端设计与实现

为减少系统的投资, 数据采集客户端采用廉价的物联网核心设备树莓派卡片式计算机通过TCP/IP与加气站的气罐、加气枪的传感器和PLC相连. 系统利用树莓派内置的Linux操作系统, 并安装Node.js, 实现与服务器端微服务相同的编程模式, 有利于系统的维护和升级.

树莓派客户端利用Node.js的johnny-five框架[11]实现与PLC连接, 并定时读取监控数据, 包括气罐的压力, 容量等参数. 在取得监控数据后, 使用Seneca的客户端对象client向服务器端监控数据微服务API网关发送监控数据, 由微服务对监控数据进行存储和处理. 传感器数据读取示意实现代码如下:

```
var five = require(“johnny-five”);
var board = new five.Board();
board.on(“ready”, function() {
    var sensor = new five.Sensor(“A0”);
    // 当检测传感器数据改变
    sensor.on(“change”, function(value) {
    //调用数据发送微服务
    });
});
```

johnny-five框架封装了大量的传感器和执行器的访问代码, 通过使用johnny-five可以以统一方式实现监测监控的编程, 简化了监测系统的开发.

5 业务管理客户端设计与实现

系统业务处理客户端采用Web方式, 可以使用PC、手机、平板等访问由Express实现的Web服务器, 请求系统的监控和管理页面. 客户端使用HTML、JavaScript、jQuery实现与服务器端的微服务API网关进行双向的数据通讯.

客户端与微服务API Gateway的通讯, 通过jQuery的AJAX JSON调用函数getJSON实现, 其实现的简要示意代码如下:

```
$.getJSON(“http://192.168.1.101/api/customer/add”, {data:customerData}, function(result){
//增加客户API调用返回的结果处理
});
```

其中请求的地址是微服务API网关的REST地址, 参数customerData是封装了增加客户表单提交的客户数据, 再使用异步响应模式取得微服务API返回的处理结果, 使用这种异步工作模式, 将极大改善Web客户的响应处理速度. Web客户端调用微服务API编程极其简单且高效的, 由此加快了企业级应用的开发进度.

6 结语

此系统的设计与实施, 开创了工业过程监测监控系统和微服务整合的低成本全新企业级互联网系统的创新实践, 通过使用廉价的物联网板式计算机如树莓派等, 极大减轻了企业的经济负担, 提高了企业投入技术改造的积极性. 使用全新的异步响应式编程模式和

高性能的Node.js结合, 极大简化了监控系统的编程和维护, 加快了系统的开发和部署效率, 通过微服务技术, 提高了系统的可靠性、可维护性和可伸缩性, 未来拟采用微服务集群技术, 能更好满足了此类监控和管理集成系统对性能和实时性需求.

参考文献

1 陆凌牛. Node. js权威指南. 北京: 机械工业出版社, 2014.

2 唐文宇. 面向SOA架构微服务的安全系统的设计与实现[硕士学位论文]. 南京: 南京大学, 2016.

3 Rodger R. Seneca Web. http://senecajs.org/2010-2015.

4 BojinovV.RESTfulWebAPIdesignwithNode.js.Birmingham-Mumbai: Packt Publishing, 2015.

5 van Vugt S. Pro Ubuntu server administration. Berkeley, USA: Apress Publishing, 2009.

6 汪鑫, 彭雨薇. 基于树莓派的网络监控系统的研究与实现. 硅谷, 2014, 7(14): 25–26. [doi: 10.3969/j.issn.1671-7597.2014.14.015]

7 席英杰, 刘文丽. 简述西门子S7-300/400的通讯功能及工业应用. 自动化与仪表, 2007, 22(1): 37–40.

8 于京, 詹晓东. 一种基于JSON格式的生产线数据采集系统模型. 制造业自动化, 2007, 34(3): 154–156.

9 唐鼎, 秦小伟. 物联网应用化智能网关技术. 信息通信技术, 2013, (6): 78–82.

10 Mardan A. Pro Express. JS. Berkeley: Apress Publishing, 2014.

11 Tilkov S, Vinoski S. Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, 2010, 14(6): 80–83. [doi: 10.1109/MIC.2010.145]