Design and Application of IoT Microservices Based on Seneca

Haidong Lv, Shankun Wang
City Institute of Dalian University of Technology, Liaoning, Dalian, China

ABSTRACT: In order to solve the complexity of traditional Internet of things application architecture and 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 combine the IoT and micro service framework Seneca based on Node.js server platform was developed. It uses unified Node.js programming model and standardized REST API protocol. The result has been a low cost, high performance, easy maintenance and security of IoT enterprise application.

1 GENERAL INSTRUCTIONS

Internet of Things\(^1\) is most growing part of the internet application technologies. It will develop further and get more spreaded under the Chinese government encouraging Internet + policies in the perspective future. As of today, the application development of Internet Things is in its initial stage. There is no matured architecture technologies for application development as the traditional application development tools have. As a result, the application development process is slow, the maintenance after deployment is not easy, the application upgrading and its re-engineering is hard to be handled. Finding a way of fast developing, easy maintaining, and up-scaling friendly, is crucial to Iot applications.

Micro services\(^2\), the new architectural technologies for software development, is better to meet those requirements. It simplifies the development process of Iot applications, enables their deployments faster, the maintenance easier.

Micro services approach is the natural choice for developers in the internet cloud age while the traditional software systems are transiting to the distributed component structures.

To better handle high concurrent requests of applications, overcome the C10K+ problem of traditional server platforms, the totally new schematic server platform technology Node.js\(^3\) is getting to its place. Due to its single thread, its non-blocking, its asynchronous processing and its event-driven features, Node.js is the first choice for today’s Internet of things oriented mobile applications and enterprise applications.

In the same time, various innovation of micro services basing on Node.js are on the horizon, and among them, the open source frame Seneca\(^4\) is recognized by developers for its simple API implementation while supporting HTTP and TCP protocols.

This paper designs a low-cost, high performance, new schematic enterprise –level management system which integrates the internet of things and business management by combining the Internet of Things with micro services, utilizing Seneca -- the Note.js server platform. And the system is implemented for the management of a CNG Gas station. The application includes monitoring gas tank and pumping machines, data transmission, gas card calculation, purchasing, sales, and inventories.

2 DESIGNS OF SYSTEM ARCHITECTURE

As a whole, the system adopts a distributed architecture as shown in Figure 1. It includes a central server, lower computers, PC clients, and mobile clients. It provides real-time monitoring, data transmission and business management.

During the initial construction, the system uses a local server, and in the future, it may use a cloud server. It uses Ubuntu 15.04\(^5\) as its operating system, Node.js as its frame, MariaDB10 as its data storage, Seneca as its micro services.

The monitoring function for gas storage and pipe line is carried on Siemens PLC S3-700\(^6\). Raspberry Pi\(^7\) 2B+ loads the monitoring data from the PLC, and calls the micro servies API on the server end sending the data to the database in the central server periodically. On every Raspberry Pi, a Linux of Debian is installed, so is Note.js. So, both the master computer and sub lower computer have the same environment for developing and routine operations, simplifying the development process, keeping the development cost.
3 DESIGNING AND IMPLANTATION OF THE MICROSERVICES

The system utilizes the Micro services frame Seneca basing on Node.js to provide the micro services required by the system. The following merits simplify the system development and its deployment.

1. It supports a variety of communication schemas, including REST API, TCP, message queue, pushing and subscribing.

2. All data transmission uses JSON format, reducing the network data flow, so the bandwidth is small.

3. The micro services adopts the pattern matching to response, making the searching and calling for micro services simple, overcoming the complexity of registration for traditional micro services, simplifying the deployment of the Micro services by a great deal.

The Seneca micro services development is quite simple. After installing the modules by running npm install Seneca, you can add customer micro service as following Note.js source codes:

```javascript
var seneca = require('seneca')();

seneca.add({role:'customer',cmd:'add'}, function(msg, respond){
    // customer adds processing code
    respond(null, {result: "process result"})
});

seneca.listen(9001);
```

Within the codes, Seneca adds a micro service, the first parameter is the positioning mode for the service, the second parameter provides the handling code for the returning function. Every micro service classifies itself by the role attribute, while the cmd attribute defines the service name.

Seneca micro services can be called through by a quite a few modes. They can be called either within a process or between nodes through Seneca client, the following is major source codes:

```javascript
var seneca = require('seneca')();

client.act({ role: 'customer', cmd: 'add', cid: '101', cname: 'Smith', ... }, function(error, result){
    // the process result code here
});
```

A micro service is called through the method ‘act’ of object ‘client’ of Seneca. By setting the mode of Node.js is set to Asynchronous non-blocking during a service call, high concurrency of customer request can be accomplished.

The default setting of Seneca provides the standard REST full mode, client end raises request to the micro service by HTTP or HTTPS. The request URL is as following:

```
http://host:port/act?role=customer&cmd=add&cid=81011...
```

If Node.js is utilized by all parts of the distributed system, then the object ‘client’ can call micro service directly. In case of other technologies being involved, like mobile clients of android and ios, the service is requested through HTTP REST API. In either case, the returning result is always a JSON data object, easy for clients to analyze and deal with.

All system functionalities are implemented by micro services. The major micro services includes gas storage monitoring, gas pumping machine monitoring, membership management, gas debit card management, CNG purchasing management, CNG inventories.

Since the outlines of micro service programming for business process are genetically the same, the following codes provide the general-purpose implementation of client end micro services.

```javascript
var mysql = require("mysql");

var pool = mysql.createPool({ //Connection pool setting
    "host": "192.168.1.101",
    "user": "root",
    "password": "878757879",
    "database": "CNG"
});

//customer microservice method
module.exports = function customer( options ) {
    this.add( 'init:user', function(msg, respond){
        //the code for processing result
        respond(null,{result:"ok"});
    });

    this.add({role:'customer',cmd:'add'}, function(msg, respond){
        //define customer add business method
        respond(null, {result:"ok"});
    });

    pool.getConnection(function(error, connection){
        if(error!=null){
            respond (error, { result:null} );
        }
    });

    var insertsql="insert into customer (CID,CName,Tel,Mobile,mail) values (?, ?, ?, ?, ?)";

    connection.query(insertsql,[msg.id,msg.name,msg.tel,...]);

    var result;
    if(error){
        respond (error, { result:null} );
    }
}
```

Figure 1. System architecture figure.
Here, only the micro service of adding customer is shown. Other customer maintenance like customer deletion should be defined in the service too.

Because the number of micro services is large, the system adapts the modulation mechanism to define one micro service to an individual JavaScript file. Then the module loading mechanisms of Node.js and Seneca are used for the deployment of the micro service, as shown in below.

```javascript
var seneca = require('seneca')();

seneca.use(require('./CustomerService.js'))
    .use(require('./CardService.js'))
    .use(require('./GasDataService.js'))
    .... // Loading other micro services
    .listen();
```

4 THE DESIGN AND IMPLEMENTATION OF MICROSERVICE API GATEWAY

In the development of enterprise level applications on micro services frame, generally the micro services API gateway\[9\] is used to improve the security, maintenance and scalability. The gateway accepts requests from clients, separates the micro services from outside clients, avoiding the contacts between outside client and the micro services, further protecting the services from attacks. In addition, when micro services changes, only the gateway API gateway needs to be modified, not the clients. All clients communicate with the API gateway. The gateway carries the load balance dynamically and redirect to the designated micro service.

The system integrates Seneca with Express, implements a web interface for the micro services API gateway. In turn, it provides information to outside though its own firewall. The API gateway encapsulates all micro service call interfaces, for instances, the data collection, gas debit card, business processes, returning the data back to outside requests.

The following codes briefly demonstrate the implementation of the micro service API gateway only. Actually, a lot more services are encapsulated within the API gateway.

```javascript
this.add({role:"api",path:"customer"},function(msg,result)
    {client.act({role:"customer",cmd:msg.action,data:msg.data},function(error,result)
        {respond(null,result); }});
```

The REST API gateway is accomplished by integrating Seneca with Express\[10\], and the below code demonstrate the implementation for integration.

```javascript
var express =require("express");

var seneca = require('seneca')();

//load all micro service require("./micro service/Main.js");

//load all server API gateway seneca.use(require('./api/apigateway.js')).client({
    //start Express Web service type:'tcp', pin:'role:*'});

//publish API Gateway as REST WEB API app.use(seneca.export("web"));

app.use(express.static(__dirname+'/web'));

app.listen(80);
```

5 CLIENT END DESIGN AND IMPLEMENTATION OF DATA COLLECTION

To reduce the system investment cost, the client end for data collection adopts Raspberry Pi, the mainstream IoT card computer which through TCP/IP connects PLC that in turn links to the censors within the gas storage and on the pumping machines. After Node.js is installed onto Raspberry Pi which has its built-in Linux operating system, both the central sever and the clients have identical programming settings, benefiting the system maintenance and upgrading later.

Raspberry Pi interconnects Siemens PLC S7-300 through Ethernet port, loads monitoring data such as storage pressure, through socket programming of TCP/IP.

Raspberry Pi clients use the net module of Node.js to implement the TCP connection with PLC, loading monitored data periodically. With the data loaded, the object ‘client’ of Seneca client end sends the data to micro services API Gateway, in turn, the micro service stores the data and processes the data.

The simplified source codes are as follow:
var net = require("net");
var seneca = require("seneca")(;
var msclient = seneca.client("9001");
var client = net.connect(port, deviceip);
    client.on('data', function(data) {
      var gasdata = convertGas(data);
      msclient.act({role: 'gasdata', cmd: 'put', gasdata: gasdata}, function(error, result) {
        console.log(result);
      });
    });
});

6 DESIGN AND IMPLEMENTATION OF BUSINESS MANAGEMENT CLIENT

The system uses web browser as the business client, users access the web server by PC, mobile phone, tablet to get their monitoring and managerial web pages. The client end uses HTML, JavaScript [11], jQuery to implement the two-way data communication with the micro service API gateway.

Between the client and the micro service API gateway, the communication is set up by call to the getJSON function of jQuery. The simplified source codes are as follow:

```javascript
$.getJSON
("http://192.168.1.101/api/customer/add", {data: custo
erData}, function(result){
  // Add codes for dealing with the returned result from API call
});
```

The IP address in the above request is the TEST address of the micro service API gateway, while the customer Data the customer gathered from the web page forms. Shown by the above programming, calling micro service from client end are very simple and efficient. So it speeds up the development processes of enterprise level applications.

7 CONCLUSION

The design and implementation of the system will speed up the innovation practices of integrations of Internet of Things with micro services for the totally new low-cost enterprise-level network applications. It will reduce the economic burdens for businesses, improve the motives of re-engineering for the enterprises. The adoption of new asynchronous responding programming methodology and the high performance of Node.js, simplifies the programming and maintenance for monitor and control systems. It speeds up the development and improves the deployment efficiency. The new micro service technology provides reliability and scalability. Furthermore, micro services clusters could be used to better serve the performance requirement such as real-time responses.

REFERENCES