文章编号:1671-4598(2016)07-0212-03 DOI:10.16526/j.cnki.11—4762/tp.2016.07.057 中图分类号:TP399 文献标识码:B

云平台分布式智能感知物联网应用开发

吕海东,葛日波
(大连理工大学 城市学院,辽宁 大连 116600)

摘要:为克服使用传统嵌入式模式开发物联网应用的复杂性,以实现将千差万别的设备使用标准的方式进行连接和访问,能将各种设备按统一的接口和协议进行访问,并构建分布式物联网架构,最终实现物联网与云平台的融合,通过云平台任何终端均可访问物联网的所有设备;研究使用基于 Node.js[1]的 Zetta[2]框架将所有设备发布为统一的 REST API 接口,通过 Zetta 提供的 Link 机制并使用协议 WebSocket 实现与云平台上 Zetta 服务器的实时数据通讯,保持云平台与物联网设备的同步;实现了全新的物联网应用开发模式和架构,构建了一个高效实时同步、开发快捷、维护简便的分布式物联网应用;使用统一的模型化方法和高效 Node.js 平台实现快速开发基于云平台的物联网应用,克服了传统开发使用 C 语言开发物联网的弊病。

关键词:云计算;物联网;服务接口;分布式;传感器

Development of Cloud Platform Distributed Intelligent IoT Application

Lü Haidong, Ge Ribo
(City Institute of Dalian University of Technology, Dalian 116600, China)

Abstract: To overcome the complexity of traditional pattern of development of embedded networking applications, in order to achieve connectivity and access between vastly different device by a standard way, a variety of equipment is accessed by a unified interface and protocol, and build distributed Things architecture, and ultimately the integration of Things and cloud platforms, all devices can access the Internet of Things through any terminal cloud platform. Study of Zetta framework based Node. js how to publish all devices into a unified REST API interfaces, Link mechanism provided by Zetta and use WebSocket protocol to achieve real-time data communication Zetta server cloud platform, cloud platforms to keep pace with networking equipment. A new application development model and infrastructure has been implemented and an efficient real-time synchronization, develop quick, easy maintenance of distributed networking applications was built. Using the unified modeling methods and efficient Node. js platform cloud-based platform for rapid development of networking applications, and overcame disadvantage of the traditional developed using C language development.

Keywords: cloud computing; IoT; REST API; distribution; sensor

0 引言

在国家提出互联网+政策推动下,物联网(Internet of Things-IoT)[3]成为互联网领域发展最迅猛的产业领域,尤其是在大众创业-万众创新的创新项目中绝大部分都是基于 IoT 的应用。根据 IDC 公司的分析报告,IoT 技术及服务到 2020 年在全球的开支将达到 8.9 万亿。

然而由于各种物联网设备千差万别,使用的通讯协议不尽相同,常用的通讯方式有串口、WIFI、3G、4G、并口等,导致物联网应用开发异常复杂,使用的编程语言和技术纷繁多样,导致 IoT 应用难以实现传统软件所具有的可维护性和可伸缩性。如何能将 IoT 开发变得简单而高效,并能以标准化统一的模型表达各种不同的 IoT 设备,屏蔽其内部的区别,将它们以统一 API 模式与互联网连接,实现分布式 IoT 应用,并使系统能自动感知联网设备的启动和停止,自动传输设备的监测数据以及传递执行动作,以上特点是简化物联网应用开发的核心。

收稿日期:2015-12-29; **修回日期**:2016-01-28。

作者简介:吕海东(1964-),男,内蒙古兴安盟人,副教授,主要从事物联网、云计算、应用架构方向的研究。

葛日波(1968-),男,山东高密人,教授,主要从事物联网、工业控制、云计算方向的研究。

为解决以上问题,在物联网的设备端和云端使用相同的编程技术是最佳的选择,这其中在 IoT 平台使用 Node.js 是最佳的选择。Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型,其轻量和高效,已经成为开发高并发请求处理的互联网应用的最佳平台。

为简化 IoT 应用开发,各大软件公司都开始了基于 Node.js 的 IoT 项目开发,如三星电子正在进行基于 JavaScript 和 Node.js 的 IoT.js 项目的研究,微软宣布将为 Windows10 IoT Core Insider 提供 Node.js 支持。

本文以 Node.js 服务器平台为基础,结合 IoT 开源框架 Zetta 开发了全新模式的云平台分布式物联网应用,通过使用创新的物联网设备抽象模型,将所有的设备包括各种传感器和执行器实现为 Node 智能对象,实现所有 IoT 设备标准的 JavaScript 编程模型,通过实时 WebSocket[4]协议能使云平台智能感知设备的在线或断线,实现设备与云平台之间双向高速的数据通讯,解决了传统物联网使用 HTTP 请求/响应模式实现数据传输时的低速度、大延时的弊病。

1 系统总体架构设计

Zetta 是基于 Node.js 的物联网开发开源框架,其通过 Node.js 的模块和中间件机制,在物联网平台如树莓派(RaspberryPi)[5], BeagleBone Black[6], Intel Edison[7], Spark

Core 上运行 Node.js 和 Zetta 框架，并创建发布 Zetta 服务器，将所有连接到物联网平台的各种设备抽象为统一的 Node.js Zetta 状态机模型对象。

Zetta 服务器将连接的每个设备通过 Zetta 对象发布为一个使用 JavaScript 编写的基于微服务的 REST API[8]，此 API 发布了每个设备的数据读取方法和动作执行方法。通过这种统一的 API 模型，极大简化了物联网应用的编程。

同时在云平台主机上部署 Zetta 服务，与分布 IoT 平台上的 Zetta 服务器进行互联，将每个本地的 IoT 平台上的设备 API 发布为云端的 REST API。如此其他客户端如 PC、手机、平板或其他 IoT 设备可以通过云上发布的 API 实现对设备的操作和监测，可构建如图 1 所示的大规模的分布式物联网应用，而且本地的设备的在线和离线状态都可以通过 WebSocket 实时协议在云平台的 Zetta 服务器上实现智能感知，实现对各处 IoT 设备的控制和监测。

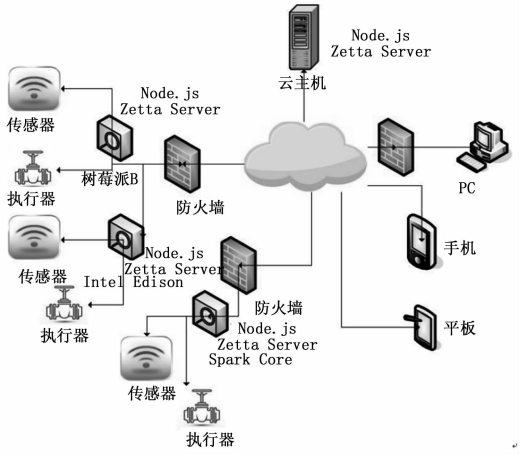

图 1　云分布式物联网总体架构示意图

2　物联网设备端设计与实现

设备端负责与各种 IoT 传感器和执行器相连，实现对监控对象的监测和控制。此层采用常用的物联网主机如树莓派、BeagleBone Black、Intel Edison、Arduino 等。这些主机可以采用任何支持 Node.js 的操作系统，如 Linux、Window10 IoT 版等，它们使用各种接口与传感器和执行器连接。在这些微主机上安装 Node.js 和 Zetta 模块，执行 Zetta Server 实例，将连接设备使用 Zetta 机制抽象为设备对象，并发布为 REST API 接口，通过与云主机 Zetta Server 连接后，发布为面向 Internet 的 API，供外部应用调用。

2.1　设备的抽象编程

在 Zetta 中使用 Zetta Driver 将 IoT 设备进行抽象模型化，在 Driver 内部封装其不同的数据访问、通讯协议和操作方法。每个设备再通过 Node.js 发布为 REST API 接口。

Zetta 已经为近 70 多种设备编写了 Driver，可在 Node.js 中直接引用，如直接访问连接到 Intel 麦克风的 zetta－microphone－edison－driver，其引用的语句如下。

var mic = require（" zetta－microphone－edison－driver");

如果没有现成的驱动，可以使用 Zetta 提供的设备 Device 类继承机制实现定制的设备驱动。在 Zetta 中推荐的设备驱动的命名规范为 zetta－｛device｝－｛platform｝－driver，其中 device 为设备名，platform 为平台名，如 edison 表示 Inter Edison，bonescrip 表示 BeagleBone Black，raspberrypi 表示树莓派驱动。如下为 LED 信号灯驱动的示意实现代码。

```
/* LED驱动 文件名:led.js 目录:divers */
var Device = require('zetta－device');
var util = require('util');
var LED = module.exports = function() {
Device.call(this);
};
util.inherits(LED, Device);
LED.prototype.init = function(config) {
config
.state('关').type('led').name('信号灯')
.when('开', { allow:['关闭', '切换'] })
.when('关', { allow:['打开', '切换'] })
.map('打开', this.turnOn).map('关闭', this.turnOff)
.map('切换', this.toggle);
};
LED.prototype.turnOn = function(cb) {
var self = this; self.state = '开';
cb(); };
LED.prototype.turnOff = function(cb) {
var self = this; self.state = '关';
cb(); };
LED.prototype.toggle = function(cb) {
if (this.state === '开') {
this.call('关闭'); cb();
} else {
this.call('打开'); cb();
} };
```

通过继承 Device 类，并实现 init 方法，定义设备的 state，并定义 JavaScript 的原型方法 prototype 实现对设备的操作，可实现任何设备的驱动编程。在 Zetta 的 modules 网址新的驱动在不断地更新和增加。

2.2　本地端 Zetta Server 的编程

在本地 IoT 主机连接各种设备，并准备设备驱动模块后，需要编写本地 Zetta Server。在其中使用设备驱动，将每个设备发布为 API。本地 Server 再与云端的 Zetta Server 互联，通过云的 Server 将设备发布到互联网上实现全局访问和调用。下面代码简要示意了一个本地 Server 的实现。

```
* 本地 Zetta Server 实现 */
var zetta = require('zetta');
var LED = require('./devices/led.js');
var ledapp=require("./apps/led.js");
var car=require("zetta－car－mock－driver");
var mic=require("zetta－microphone－edison－driver");
zetta()
.name('CityIOT')
.use(LED)
.use(car)
.use(mic)
.load(ledapp)
.link("http://city－iot.herokuapp.com/")
.listen(9001);
```

代码中先引入各种设备驱动，再调用 zetta 实例对象的 use 方法，将设备作为中间件嵌入到 Zetta Server，通过 link 方法与云端的 Zetta 服务器相连，将本地 API 发布为互联网 API，内部使用 WebSocket 协议实现本地设备与云端 API 数据的实时传输，这样在云中 API 中可实时监测各种设备的状态和数据，也可以使用 API 调用对在线的设备发送控制指令和参数。

2.3 本地端 IoT 应用编程

物联网本地端除了可以引用设备驱动外，还可以根据应用功能需求，使用 Node.js 的事件驱动、异步、非阻塞的响应式编程模式编写设备间相互协作完成的功能模块，根据监测设备的状态和数据以执行相关的动作，如发送报警信号，切断电路开关和控制阀门流量大小等。如下简要示意代码展示了 IoT 应用的编程实现。

```
module.exports=function(server){
var ledquery=server.where({type:"led",name:"LED灯"});
var micquery = server.where(type:"mic",name:"内置麦克");
server.observe([ledquery,micquery],function(led,mic){
led.state.on("开",function(){
   mic.speek(100); });
}); };
```

IoT 应用编程使用 Node. js 的模块机制，通过 Zetta Server 的 where 方法定位 IoT 设备和 observe 方法智能感知设备是否启动、停止或在线，在 JavaScript 回调方法中取得在线设备，最后监控设备的状态变化事件，以异步响应式编程模式实现需要的执行的动作和功能。

3 云端服务器实现

通过 Zetta 实例对象的 link 方法可以连接多个 Zetta 服务器，由 Zetta 自身完成连接时使用的协议、请求握手、防火墙穿越等复杂的工作细节，极大减轻了开发者的编程负担，使开发者集中精力专注于业务功能实现。

将本地 Zetta 服务器与云中 Zetta 的服务器互联，实现本地服务器连接设备的 API 公布在互联网上，任何连接互联网的应用客户端都可以通过此 API 访问所有分别各地的 IoT 设备，实现分布式物联网。

云端 Zetta 服务器与本地服务器实现基本相同，为连接众多本地的本地 Zetta 服务器，并集中发布每个联网 Device 的 API，Zetta 提供了最简单的方法 expose（*），可以发布每个通过 link 方法连接的本地 Device 的 API。云端服务器的简化的示意代码如下所示。

```
var zetta = require('zetta');
var PORT = process.env.PORT || 3000;
zetta()
.name('cloud')
.expose('*')
.listen(PORT);
```

创建 Node.js 项目，并将此服务器启动代码保存为 server.js，在项目的配置文件 package.json 中指定依赖的 Node.js 模块，包括 zetta 和其他模块。在百度云、阿里云或开源云平台 Heroku 上申请云主机，推荐使用 Linux 操作系统，并安装 Node.js 的最新版。

本文使用 Heroku 免费云平台作为测试目的，生产环境要使用商业化平台以保证系统的可靠运行和售后服务。使用平台提供的 Git 工具将 Zetta 服务器项目部署到 Heroku，选择应用类型为 node.js 即可。

假如部署的应用名称为 city－iot，则 Zetta 服务的请求地址为 https：//city－iot.heroku.com.

每个连接的设备都有一个唯一的请求地址，其格式为：

https：//city － iot.heroku.com：3000/servers/CityIOT/devices/e6f5b480－e96e－4fdc－8718－91aeb0234c99

其中 CityIOT 是本地的 Server 名称，最后的数字编码是 Zetta 自动生成的设备 ID，使用如下的 JSON 格式请求数据，实现对设备的控制。

```
{
"method":"POST",
"href":"https://city－iot.heroku.com:3000/servers/CityIOT/devices/e6f5b480－e96e－4fdc－8718－91aeb0234c99",
"fields":[{
"name": "action",
"value": "开"
}]}
```

此控制数据可以使用任何客户端发送 HTTP POST 请求给联网的 IoT 设备。

4 物联网客户端设计编程

分布式物联网实现云服务与各个端点的 IoT 设备连接后，就可以通过互联网，在任何 PC、手机或平板上使用 Web 方式控制所有的连接设备，包括实时接收监测的数据，以及发送对 IoT 设备的控制指令。

Web 客户端可以使用任何 AJAX 框架如 jQuery，AngularJS，Dojo 等请求云端设备 API 实现对 IoT 设备的访问和控制。如下是演示了使用 jQuery 的异步 AJAX 请求方法. post 控制 LED 灯点亮的实现代码。

```
.post("https://city－iot.heroku.com:3000/servers/CityIOT/devices/e6f5b480－e96e－4fdc－8718－91aeb0234c99",{action:"开"},function(){
//回调处理代码
})
```

Zetta 设备最引人注目的特点是支持实时 WebSocket 协议，支持设备数据的高速、实时传输，客户端使用此协议可以实时接收设备的监控数据，不再需要定时的 HTTP 请求，由 IoT 设备采用推方式自动发送给客户端。

5 系统的实际应用

使用此云平台分布式 IoT 应用模式对传统的供热监控系统进行改造，在系统的投资上大大节省。原来使用昂贵的工业控制计算机，现在下位机全部采用微型树莓派。在每个供热锅炉房设置一个 Zetta Server 端点，将多个供热站的 Server 与云 Zetta 进行互联，加大简化了系统的应用开发，使得项目的开发进度成倍提高，系统的可维护性和可伸缩性得到巨大改善，新增换热站时，通过部署一个 Zetta Server 可立即进行入网监测。

6 结论

将新的服务器平台 Node.js 和 Zetta 开源框架应用于基于云的 IoT 应用开发，简化了 IoT 应用开发的复杂性，加快了物联网应用的开发效率，提高了应用的可维护性、可升级性和可

（下转第 218 页）

在同一 λ 值下，不同的 Pico 基站数对应不同的 β 值，12 个 Pico 基站下需要的 β 值小于 3 个 Pico 基站下的 β 值，这是因为 Pico 基站越多，但是高干扰用户数反而更少，所以需要分配的 ABS 会少。

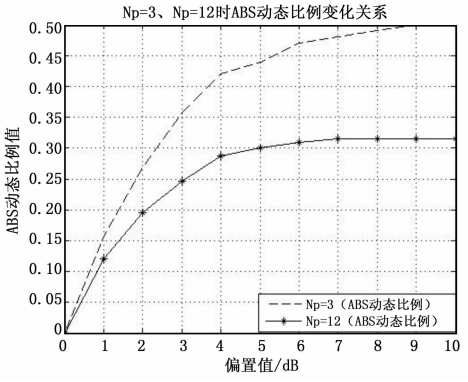

图 4 不同 λ 时的 ABS 动态比例

图 5 为固定 ABS 比例和动态分配 ABS 比例情况下的系统容量对比。表明在 3 个微基站下，固定的几乎空白子帧比例值下的系统容量基本没有明显变化，动态的几乎空白子帧比例值下系统容量在一定范围内有所增加，且动态的几乎空白子帧比例值下系统容量高于固定的几乎空白子帧比例值下的系统容量。

与固定的 ABS 子帧比例值为 $\beta=0.5$ 情形相比，新算法增加了系统容量，不仅提高了信道资源利用率，同时提高了增强小区用户的数据传输率。

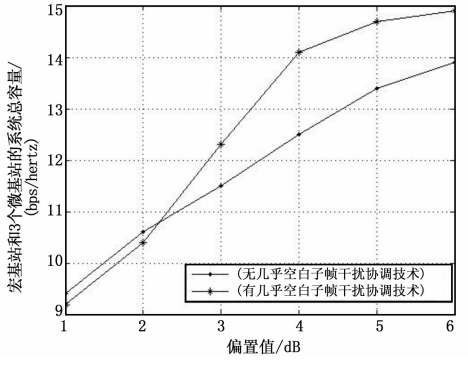

图 5 宏基站和 3 个微基站下的系统总容量

4 结论

为了提高两层异构网络中下行信道资源的利用率，在增强小区中调度几乎空白子帧的干扰协调方案的基础上提出了一种动态确定几乎空白子帧比例的算法。新算法通过增强小区用户数与总用户数的比例来确定的几乎空白子帧比率，改善了信道资源的利用。仿真结果表明，新算法不仅提高了系统总容量，同时也实现了下行信道资源的合理利用。

参考文献：

[1] Myunggoon H，Seungyoung P. Recursive load balance scheme for two－tier cellular networks using enhanced inter－cell interference coordination [J]. EURASIP Journal on Wireless Communications and Networking，2015，2015 (1)：1－12.

[2] 廖 原. LTE－Advanced 异构网络小区干扰协调技术的研究 [D]. 兰州：兰州交通大学，2014.

[3] 张 琛，粟 欣，王文清等. 异构网络跨层协作传输技术研究 [J]. 通信学报，2014 (8)：198－205.

[4] Fei Z S，Ding H C，Xing C W，et al. Performance analysis for range expansion in heterogeneous networks [J]. Science China Information Sciences，2014，57 (8)：1－10.

[5] 孙长印，姜 静，卢光跃. 异构网中载波聚合系统的成员载波选择和干扰协同 [J]. 电讯技术，2012，12：1887－1892.

[6] 周 昊. LTE－A 异构网络中自适应 ABS 配置与功率控制算法的研究 [D]. 北京：北京邮电大学，2015.

[7] Damnjanovic A，Montojo J，Wei Y B，et al. A survey on 3GPP heterogeneous networks [J]. Wireless Communications，IEEE，2011，18 (3)：10－21.

[8] ISMAIL G I. Capacity and Fairness Analysis of Heterogeneous Networks with Range Expansion and Interference Coordination. [J]. IEEE Communications Letters，2011，15：1084－1087.

[9] Guvenc I，Moo－Ryong J，Demirdoeng I，et al. Range expansion and inter－cell interference coordination (ICIC) for picocell networks [A]. IEEE Vehicular Technology Conference (VTC) [C]. San Francisco，2011：1－6.

[10] Interference management in UMTS femtocells [EB/OL]. http：//small－cellforum. org/smallcellforum/resources－white－papers Index Num－ber：003.

[11] Seungseob L，Sukyong L，Kyungsoo K，et al. Optimal deployment of pico base stations in LTE－Advanced heterogeneous networks [J]. Computer Networks，2014，72：127－139.

[12] Gao L Q，Tian H，Wang M，et al. Dynamic bias setting for range extension in LTE－advanced macro－pico heterogeneous networks [J]. The Journal of China Universities of Posts and Telecommunications，2013，20 (4)：28－33.

[13] Roy C P，Hoill J，Sun－Moon J. ABS Scheduling Technique for Interference Mitigation of M2M Based Medical WBAN Service [J]. Wireless Personal Communications，2014，79 (4)：2685－2700.

(上接第 214 页)

扩展性。通过 Zetta 服务器互联可以实现超大规模的分布式 IoT 应用，未来可以在智慧城市，智能社区，智能工厂等各行各业中广泛应用，开创互联网＋普及时代。

参考文献：

[1] 陆凌生，Node.js 权威指南 [M]. 北京：机械工业出版社，2014.

[2] Zetta online [Z/OL]. http：//www. zettajs. org.

[3] 薛燕红，物联网技术与应用 [M]. 北京：清华大学出版社，2012.

[4] 薛珑斌，刘钊远. 基于 Websocket 的远程实时通讯 [J]. 计算机和数字工程，2014 (3).

[5] 汪 鑫，彭雨薇. 基于树莓派的网络监控系统的研究与实现 [J]. 硅谷，2014 (14)：25－26.

[6] 孙 溪. 800 Mhz TETRA 无线政务专用网的物联网数据传输模块的开发 [J]. 中国仪器仪表，2014 (7).

[7] 苗 雪. 基于 Intel/NXP/TI 智能家居安防系统网关解决方案 [J]. 智能建筑和城市信息，2015 (11)：36－39.

[8] 高嘉译，高强等. 面向移动应用的后端服务平台 [J]. 计算机系统应用，2014 (2)：22－27.