**WebSocket Server详细设计书**

# 一、服务描述

Websocket Server作为中间层，提供了实时数据和HTML5页面进行双向交互的服务。在下层，适配器、报警管理服务以及其它服务，向Websocket Server提供实时双向交互数据。在上层，Websocket Server向登录用户的HTML5页面提供实时双向交互数据。

Websocket Server还需开发指令服务功能，通过编排微服务来提供云服务总线的数据交互功能。

Websocket Server也可作为IM的双向信息服务器。

## 1、开发语言

Node.js/java语言

## 日志文件

日志时间、日志类别、日志内容等，日志文件存在本地

**二、流程模块**

## 1、适配器配置流程



## 客户端联接流程(分适配器和其它客户端)



## 客户端关闭流程



## 数据订阅流程



## 数据发布流程



## 适配器通信流程



## 应用服务端通信流程



## 8、用户端通信流程



# 三、功能模块

|  |  |  |
| --- | --- | --- |
| 序号 | 功能模块 | 功能描述 |
| 1 | Restful接口 | Restful api调用，列出所有功能 |
| 2 | 用户权限认证 | 本地数据库或认证服务器进行用户认证 |
| 3 | 环境变量配置 | 配置环境变量，如名称，ID，分布数据库地址、IP等等 |
| 4 | 管理配置web页面 | 1. 适配器接口管理 2. 用户端管理 3. 报警端管理 4. 性能分析 5. 指令增删改管理 |
| 5 | 适配器接口 | 待定 |
| 6 | 管理服务中心接口 | Restful，向管理服务中心进行对象注册和服务注册 |
| 7 | 用户端接口 | 通过Websocket client接入，具有用户端管理和数据监控功能、日志管理功能。 |
| 8 | 指令（带参数）管理 | 云服务总线的指令置中心，定义不同的指令函数，可以通过自定义格式编程，调用其它服务的RestAPI，通过服务编排，最终形成结果数据集（自定义json格式）。用户端通过发送指令服务要求，返回结果集。  指令参数包括：实时数据的订阅，非实时数据的请求等等。 |
| 9 | 在线检测 | 向管理服务中心发送心跳检测（时间间隔可调）。  通过RestAPI请求，提交性能分析参数，设置性能瓶颈报警。 |

# 总线通信协议模块

总线接口定义有两种，一实一虚，实的规划为实际通信的内容，虚的规划为对实的理化解析，像虚拟机一样，支撑对实际程序的运行。

虚的规划，目的是为了实现通信双方的自动应答。

先实后虚，本文先定义实的总线接口，从实际需求出发，逐步增加功能定义。

一、客户端和websocket server的通信协议

\* JSON格式

\* C:客户端 S:服务端

\* 当C向S发消息时，callback通常是一个随机的数字，S回应C时只需原样返回callback就可以了。

只有当S主动向C发消息时使用特定的callback值，且S不需要等待C的回应。

1. 客户端发起的心跳响应

C -> S:

'heartbeat'

S -> C:

'heartbeat'

\*说明：客户端每隔30秒像服务端发送’heartbeat’字符串，服务端收到后原样返回’heartbeat’字符串即可

2. 适配器连接注册

C -> S:

'{

"uri": "adapter.register",

"data": ["适配器ID", "适配器名称"，"适配器URL"],

"callback": 42

}'

S -> C:

'{

"callback": 42,

"data": true,

}'

\*说明：适配器连接服务器成功后，向服务器注册适配器信息

\*本例：向服务器注册适配器的ID、名称、URL，加上服务器自动识别的适配器IP和端口号，形成关联

3. 客户端向服务器发送数据

C -> S:

'{

"uri": "channel.send",

"data": ["适配器ID"，"点名1:值"，"点名2:值"，...],

"callback": 21

}'

S -> C:

'{

"callback": 21,

"data": true,

}'

\*说明：本接口为适配器向服务器发送变化了的数据，"适配器ID"本身也设置为管道名，向订阅该适配器ID的客户端发送变化了的点值。

\*本例：点名1、点名2变化了的值发送到服务器，服务器对该适配器ID的相关点值进行更新

4. 服务器向客户端发送数据

S -> C:

'{

"uri": "channel.send",

"data": ["点名1:值"，"点名2:值"，...],

"callback": 21

}'

C -> S:

'{

"callback": 21,

"data": true,

}'

\*说明：本接口为服务器向客户端发送变化了的数据。

\*本例：如前端把输出点名1、点名2变化了的值发送到服务器，服务器作为中转，向适配器ID输出控制值。

5. 客户端开启管道监听（订阅请求）

C -> S:

'{

"uri": "channel.addListener",

"data": ["主题名"，"文本类型"，"文本串"],

"callback": 42

}'

S -> C:

'{

"callback": 42,

"data": true,

}'

\*说明：本接口为批量处理接口

\*举例：h5前端的编号是H0BA0001，有2个空调的数据要展示，它订阅的主题是：网络控制器，文件类型是：元素文本，文本串是：<网络控制器1><网络控制器2>。元素文本指，后面都是一个个元素，每个元素用<>包起来。

6. 服务器向订阅客户端管道发送数据

S -> C:

'{

"callback": “channel.receive",

"data": ["主题名"，"文件类型"，"字符串"],

}'

\*说明：本接口为服务器主动向客户端推送信息,只推送给监听了管道[适配器ID]的客户端。当收到客户端（适配器）写入管道[通过channel.send]的数据后触发。

本例：收到来自管道[适配器ID]的数据，点名1和点名2的变化值。

7. 管道监听异常反馈

S -> C:

'{

"callback": "channel.error",

"data": ["me", "监听中断"],

}'

\*说明：本接口为服务器主动向客户端推送信息,只推送给监听了管道[me]的客户端

\*本例：管道[me]监听异常，原因是[监听中断]

8. 未知异常反馈

S -> C:

'{

"callback": "unknown.error",

"data": "适配器ID发生未知异常，请尝试重启服务",

}'

\*说明：本接口为服务器主动向客户端推送信息，推送给所有连接的客户端，也就是广播。当不明原因的异常出现时，通知用户

\*本例：[适配器ID发生未知异常，请尝试重启服务]

# 五、数据库表格设计模块

1、适配器属性表（适配器要各种添加：网络控制器（包含楼宇控制器、门禁控制器、灯光控制器等））

字段：注册时间、适配器ID、端口号、适配器名称、适配器类别、描述、版本号等

|  |  |  |
| --- | --- | --- |
| Name | Code | Type |
| 适配器ID | adapterId | varchar(20) |
| 适配器端口号 | adapterPort | int |
| 适配器名称 | adapterName | varchar(20) |
| 适配器类别 | adapterCate | varchar(20) |
| 注册时间 | registerTime | date |
| 适配器描述 | adapterDesc | varchar(100) |
| 适配器版本号 | adapterVersion | varchar(10) |

2、网络控制器属性表（适配器里面包含多个网络控制器，网络控制器里面包含多个点）

字段：网络控制器ID，适配器ID，网络控制器名称，IP地址、端口号、通信协议、描述、安装地点、安装时间等

Network controller = NC 网络控制器

|  |  |  |
| --- | --- | --- |
| Name | Code | Type |
| 网络控制器ID | ncId | varchar(20) |
| 适配器ID | adapterId | varchar(20) |
| 网络控制器名称 | ncName | varchar(20) |
| 网络控制器IP地址 | ncIpAddress | varchar(10) |
| 网络控制器端口号 | ncPort | int |
| 网络控制器通信协议 | ncComProtocol | varchar(10) |
| 网络控制器描述 | ncDesc | varchar(100) |
| 网络控制器安装地点 | ncInstallAddress | varchar(50) |
| 网络控制器安装时间 | ncInstallTime | date |

## 3、监控点表（实时数据，写入redis中）

字段：点ID、网络控制器ID，适配器ID、点类型、点名称、值、单位，更新时间等

|  |  |  |
| --- | --- | --- |
| Name | Code | Type |
| 网络控制器ID | ncId | varchar(20) |
| 适配器ID | adapterId | varchar(20) |
| 点ID | dotId | varchar(20) |
| 点类型 | dotCategory | varchar(10) |
| 点名称 | dotName | varchar(10) |
| 点的值 | dotValue | varchar(10) |
| 值的单位 | valueUnit | varchar(10) |
| 值的更新时间 | valueUpdateTime | date |

## 4、树形结构表

字段：云服务总线ID、一级对象类别、一级对象ID、二级对象类别、二级对象ID

|  |  |  |
| --- | --- | --- |
| Name | Code | Type |
| 网络控制器ID | ncId | varchar(20) |
| 适配器ID | adapterId | varchar(20) |
| 点ID | dotId | varchar(20) |
| 点类型 | dotCategory | varchar(10) |
| 点名称 | dotName | varchar(10) |
| 点的值 | dotValue | varchar(10) |
| 点的单位 | doc | varchar(100) |
| 点的更新时间 | dotUpdateTime | date |

5、客户端登记表（需要将操作的用户一起登记）（实时数据，写入redis中）

字段：登录用户id,用户名，客户端id，客户端名称，客户端类型，ip地址，端口号，登录时间

|  |  |  |
| --- | --- | --- |
| Name | Code | Type |
| 用户ID | userId | varchar(20) |
| 用户名 | userName | varchar(10) |
| 客户端ID | clientId | varchar(20) |
| 客户端名称 | clientName | varchar(10) |
| 客户端类型 | clientCategory | varchar(10) |
| 客户端ip地址 | clientIpAddress | varchar(10) |
| 客户端端口号 | clientPort | int |
| 客户端登录时间 | clientLoginTime | date |

## 6、订阅表（实时数据，写入redis中）

字段：客户端ID、订阅主题、文件类型、订阅子项

|  |  |  |
| --- | --- | --- |
| Name | Code | Type |
| 客户端ID | clientId | varchar(20) |
| 订阅主题 | SubscribeTopic | varchar(20) |
| 文件类型 | fileCategory | text |
| 订阅子项 | subscribeSubitem | varchar(20) |

## 7、云服务总线信息表

字段：云服务总线ID，云服务总线名称，IP地址，端口号，描述

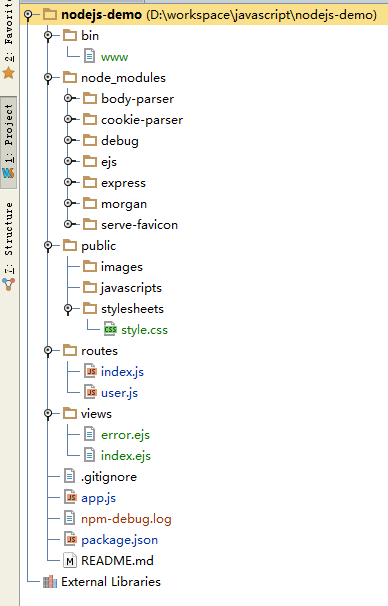
Cloud Server = cs 云服务

|  |  |  |
| --- | --- | --- |
| Name | Code | Type |
| 云服务总线ID | csId | varchar(20) |
| 云服务总线名称 | csName | varchar(20) |
| 云服务IP地址 | csIpAddress | varchar(20) |
| 云服务端口号 | csPort | int |
| 云服务描述 | csDesc | varchar(40) |

# 六、node.js编程文件层次结构规范

该WebSocket Server使用的是node.js语言编写的，该项目使用了node.js中的express框架创建的，该框架是一个web框架，它是目前最稳定、使用最广泛，而且Node.js官方推荐的唯一一个 Web 开发框架。

下面介绍一下express项目的文件结构:



-bin,存放启动项目的脚本文件

-node\_modules,存放所有的项目依赖库

-public,静态文件的存放处，例如css、js、img等

-routes,路由文件,相当于是MVC中的Controller

-views,页面文件,本项目中使用的是ejs模板

-package.json,项目依赖配置及开发者信息

-app.js,应用核心配置文件

## 七、java编程规范