恒大智慧小区项目

**WebSocket集成指南**

**恒大智慧小区项目团队**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **文档信息** | | | | |
| 项目名称 | 恒大智慧小区项目 | | | |
| 文档名称 | WebSocket集成指南 | | | |
| 项目编号 |  | | 文档编号 |  |
| 作者 | 彭志祥 | | 文件类型 | 工作件/交付件 |
| 审批者 |  | | 审批日期 |  |
| 文件名称 | WebSocket集成指南 | | | |
| 内容范围 | WebSocket集成指南 | | | |
| 适用对象 | WebSocket集成的开发人员、测试人员 | | | |
| **版本信息** | | | | |
| 版本编号 | 版本日期 | 作者 | 修改描述 | |
| 0.1 | 2018/01/05 | 彭志祥 | 创建 | |
|  |  |  |  | |
|  |  |  |  | |
|  |  |  |  | |
|  |  |  |  | |
|  |  |  |  | |
|  |  |  |  | |

目 录

1 引言 5

1.1 编写目的 5

1.2 背景 5

1.3 参考资料 5

1.4 术语定义及说明 6

2 概述 7

2.1 WebSocket概述 7

2.2 WebSocket组件介绍 7

2.2.1 WebSocket组件与其他应用集成架构介绍 7

2.2.2 WebSocket组件设计结构介绍 9

2.2.3 WebSocket消息推送组件部署示意图介绍 10

2.1 WebSocket组件推送消息格式规范 10

2.1.1 定义推送消息格式规范 10

2.1.2 定义推送消息MQ对象 11

2.1.3 定义推送消息MQ对象格式规范 12

2.2 WebSocket组件与应用集成步骤 14

2.2.1 步骤一：定义WebSocket注册地址 14

2.2.2 步骤二：定义WebSocket组件与各应用推送的MQ名称 15

2.2.3 步骤三：前端页面中创建WebSocket使用实例 16

2.2.4 步骤四：集成的应用向指定定义的MQ推送消息 17

2.2.5 步骤五：WebSocket组件监听定义的MQ对列 18

# 引言

## 编写目的

为提高团队协作效率,为WebSocket集成相关的开发人员服务.

预期读者：开发人员、测试人员、项目经理。

## 背景

作为恒大集团进军互联网社区服务产业的旗舰公司，恒腾网络受集团指派，开展“智慧小区”的研究和建设工作。恒大希望通过“智慧小区”的建设，一方面通过给客户提供差异化的体验，实现对恒大目前产品的增值，另一方面，实现恒大基于智慧小区的服务能力升级。工程将分阶段进行。未来预期本项目成果将应用到恒大集团800多家小区，及其他房地产合作商。

本项目为“恒大智慧小区“工程的第一阶段，即通过快速而集中的开发建设，预计在2018年3月30日前对整体方案实现现场验证。

## 参考资料

列出详细设计报告引用的文献或资料，资料的作者、标题、出版单位和出版日期等信息，必要时说明如何得到这些资料。

## 术语定义及说明

列出本文档中用到的可能会引起混淆的专门术语、定义和缩写词的原文。

# 概述

## WebSocket概述

WebSocket API是下一代客户端-服务器的异步通信方法。该通信取代了单个的TCP套接字，使用ws或wss协议，可用于任意的客户端和服务器程序。WebSocket目前由W3C进行标准化。WebSocket已经受到IE11、Firefox 4、Chrome 4、Opera 10.70以及Safari 5等浏览器的支持。

WebSocket API最伟大之处在于服务器和客户端可以在给定的时间范围内的任意时刻，相互推送信息。WebSocket并不限于以Ajax(或XHR)方式通信，因为Ajax技术需要客户端发起请求，而WebSocket服务器和客户端可以彼此相互推送信息；XHR受到域的限制，而WebSocket允许跨域通信。

WebSocket技术为指定目标创建，用于双向推送消息，这与Ajax技术没有冲突。

## WebSocket组件介绍

### WebSocket组件与其他应用集成架构介绍



### WebSocket组件设计结构介绍



WebSocket注册地址如：

WebSocket 方式：Ws:url:port/webSocket/readData

例如：

ws:// 127.0.0.1:9018/scp-websocketcomponent/webSocket/readData

### WebSocket消息推送组件部署示意图介绍



## WebSocket组件推送消息格式规范

### 定义推送消息格式规范

|  |
| --- |
| {  "data":{  "extAttributes":{},  "infoDtos":[{  "businessId":"businessId-0",  "sourceSysId":"sourceSysId-0",  "targetSysId":"targetSysId-0",  "extAttributes":{},  "type":"type-0",  "status":"status-0",  "info":"infodesc-0",  "time":"2018-01-03 15:26:00",  "xAxis":"x-0",  "yAxis":"y-0"  },  ...  ]},  "hasError":false  } |

以上JSON消息的内容的传输主要通过Java对象的序列化与反序列化由框架完成。

### 定义推送消息MQ对象

|  |
| --- |
| **public** **class** InfoListDto **extends** BaseBusinessDto {  /\*\*  \* **@Field** long serialVersionUID  \*/  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** List<InfoDto> infoDtos;  /\*\*  \* **@return** the infoDtos  \*/  **public** List<InfoDto> getInfoDtos() {  **return** infoDtos;  }  /\*\*  \* **@param** infoDtos  \* the infoDtos to set  \*/  **public** **void** setInfoDtos(List<InfoDto> infoDtos) {  **this**.infoDtos = infoDtos;  }  } |

### 定义推送消息MQ对象格式规范

|  |
| --- |
| **public** **class** InfoDto **extends** BaseBusinessDto {  /\*\*  \* **@Field** long serialVersionUID  \*/  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** String type;  **private** String status;  **private** String info;  **private** String time;  **private** String xAxis;  **private** String yAxis;  **public** String getType() {  **return** type;  }  **public** **void** setType(String type) {  **this**.type = type;  }  **public** String getStatus() {  **return** status;  }  **public** **void** setStatus(String status) {  **this**.status = status;  }  **public** String getInfo() {  **return** info;  }  **public** **void** setInfo(String info) {  **this**.info = info;  }  **public** String getTime() {  **return** time;  }  **public** **void** setTime(String time) {  **this**.time = time;  }  **public** String getxAxis() {  **return** xAxis;  }  **public** **void** setxAxis(String xAxis) {  **this**.xAxis = xAxis;  }  **public** String getyAxis() {  **return** yAxis;  }  **public** **void** setyAxis(String yAxis) {  **this**.yAxis = yAxis;  }  } |

## WebSocket组件与应用集成步骤

### 步骤一：定义WebSocket注册地址

一．WebSocket全局使用的注册地址定义规范需协商定义，如目前egsc-UI全局的初步定义为：ws://xxx:port/webSocket/readData，对应的MQ名称：EGSC\_SCP\_EVENTCOMPONENT\_DATA\_QUEUE。

二．WebSocket应用局部使用的注册地址定义规范需协商定义，应根据应用的场景来定义。

WebSocket注册地址格式：/egsc\_平台名称\_应用与组件名称/WebSocket地址标识，WebSocket注册地址名称全部为小写英文字母。

egsc：固定格式

平台名称：scp（小区平台）

应用与组件名称：参考《EGSC\_公共平台\_应用与组件命名清单\_v1.0\_20171208.xlsx》，使用“英文名称”列并改为全部字母小写。

前端通过webSocket监听消息：ws://xxx:port/webSocket/XXX

* **示例**

ws://xxx:port/webSocket/egsc\_scp\_cassso/readData

* **使用规范**

所有WebSocket注册地址格式定义由WebSocket组件项目组统一维护，各应用定义WebSocket注册地址上报WebSocket组件项目组，WebSocket组件项目组负责审核，审核通过后维护在WebSocket组件中，并下发至各应用。各应用使用WebSocket注册地址时，通过前端WebSocket注册调用及可。

### 步骤二：定义WebSocket组件与各应用推送的MQ名称

各应用推送的MQ名称规则参考《应用开发 – MQ使用规范》

WebSocket组件与各应用推送的MQ名称需要对应上**，**目前WebSocket组件监听的MQ名称：EGSC\_SCP\_EVENTCOMPONENT\_DATA\_QUEUE，WebSocket组件默认的 rabbitmq配置信息如下，对应了ws://xxx:port/webSocket/readData地址：

|  |
| --- |
| #default mq 使用时放开  spring.rabbitmq.host=192.168.0.208  spring.rabbitmq.port=5672  spring.rabbitmq.username=rabbitmq  spring.rabbitmq.password=rabbitmq  #spring.rabbitmq.virtualHost=test/test  ##需要定义的queue名称，如果不需要定义，可以注释此行  spring.rabbitmq.queues=EGSC\_SCP\_EVENTCOMPONENT\_DATA\_QUEUE |

### 步骤三：前端页面中创建WebSocket使用实例

|  |
| --- |
| export default {  data () {  return {  websocket: null,  wspath: '127.0.0.1:9018/scp-websocketcomponent'  }  },  mounted () {  this.loadWebsocket()  },  methods: {  loadWebsocket () {  this.websocket = new WebSocket('ws://' + this.wspath + '/webSocket/readData')  this.websocket.onopen = () => {  // Web Socket 已连接上，使用 send() 方法发送数据  this.msg += 'WebSocket链接开始！\n'  console.log('数据发送中...')  }  this.websocket.onmessage = evnt => {  console.log('数据已接收...')  this.msg += evnt.data + ' \n'  }  this.websocket.onclose = evnt => {  // 关闭 websocket  console.log('连接已关闭...')  this.msg += 'WebSocket链接关闭！\n'  }  this.websocket.onerror = evnt => {  // 链接出错  console.log('链接出错...')  this.msg += 'WebSocket链接出错！\n'  }  // 组件销毁时调用，中断websocket链接  this.over = () => {  this.websocket.close()  }  console.log(' this.msg: ' + this.msg)  },  sendMessage () {  this.websocket.send(this.sendmsg)  }  }  } |

### 步骤四：集成的应用向指定定义的MQ推送消息

参考Demo中MQ推送方式，按前期定义的WebSocket组件与应用推送的MQ名称推送上MQ对列。

### 步骤五：WebSocket组件监听定义的MQ对列

WebSocket组件监听定义的MQ对列，获取消息后自动推送给注册上的Web Socket Client浏览器列表，Web Socket Client中onmessage中能获取到对应的消息数据，即可实现相关的业务。