**微信小程序WebSocket实践**

2019年4月4日

| [front-end](http://yrq110.me/categories/front-end)

微信基础库1.7.0之后的版本提供了新版的WebSocket API，考虑到兼容性问题，尝试整合新旧两种版本的API，进行简单的封装。

从以下几个角度对微信小程序中所提供的WebSocket API封装

* API兼容性
* 重连机制

**小程序WebSocket API**

**旧版WebSocket API的使用**

* 创建 - wx.connectSocket
* 发送消息 - sendSocketMessage
* 监听事件 - onSocketOpen

let socketOpen = false

const socketMsgQueue = []

// 连接socket

wx.connectSocket({

url: 'test.php'

})

// 监听打开事件

wx.onSocketOpen(function(res) {

socketOpen = true

for (let i = 0; i < socketMsgQueue.length; i++){

sendSocketMessage(socketMsgQueue[i])

}

socketMsgQueue = []

})

// 发送消息

function sendSocketMessage(msg) {

if (socketOpen) {

wx.sendSocketMessage({

data:msg

})

} else {

socketMsgQueue.push(msg)

}

}

**新版WebSocket API的使用**

支持1.7.0+，连接后会返回一个SocketTask对象，在该对象上监听该连接的各种事件与执行发送消息等操作。

let socketTask = wx.connectSocket({

url: 'test.php'

})

// 监听打开事件

socketTask.onOpen(function(res) {

socketOpen = true

// 发送信息

socketTask.send({msg: "hello world"})

})

**并发数**

* 1.7.0 及以上版本，最多可以同时存在 5 个 WebSocket 连接。
* 1.7.0 以下版本，一个小程序同时只能有一个 WebSocket 连接，如果当前已存在一个 WebSocket 连接，会自动关闭该连接，并重新创建一个 WebSocket 连接。

**模块封装**

简单封装封装一个兼容新旧socketAPI的模块，仅考虑存在单个socket连接的情况

**创建与事件监听**

* 新版: 在socket连接时会返回一个socketTask对象，监听事件是在该对象的基础上进行
* 旧版: 直接使用wx放进行创建与监听方法

统一创建与添加监听函数

init() {

let st = this.connect()

this.listen(st)

this.manualClose = false

}

创建连接

connect() {

let st = wx.connectSocket(this.config.server)

return st

}

添加事件监听函数

listen(st) {

if (st !== undefined) {

this.ws = st

this.ws.onOpen(() => this.openHandle())

this.ws.onError(() => this.errorHandle())

this.ws.onClose(() => this.closeHandle())

this.ws.onMessage(res => this.messageHandle(res))

} else {

wx.onSocketOpen(() => this.openHandle())

wx.onSocketError(() => this.errorHandle())

wx.onSocketClose(() => this.closeHandle())

wx.onSocketMessage(res => this.messageHandle(res))

}

}

**重连机制**

预设标记位

retryLock = false; // 避免多次同时重连

socketOpen = false; // 连接状态

manualClose = false; // 主动断开标记

在连接关闭监听函数中执行重连

closeHandle() {

console.info('WebSocket closed')

this.socketOpen = false

this.retryLock = false

// 不论是error还是close都会触发close事件，统一在这里进行重连

// 初次连接失败不进行重连(失败不会进入到onOpen的监听事件中，那时未声明retryTimes变量)

this.retryTimes !== undefined && this.reconnect()

}

判断重连锁与是否主动断开进行重连

reconnect() {

if (this.retryLock) return

this.retryLock = true

// 若manualClose为true，表明不是主动断开

if (!this.manualClose) {

// 开始重连

setTimeout(() => {

this.retry()

}, this.retryInterval)

}

}

重连函数，包含重连次数的限制

retry() {

if (

this.socketOpen ||

(this.retryTimes > 0 && this.retryCount <= this.retryTimes)

) {

console.warn(`reconnect ending. reconnect ${this.retryTimes} times`)

if (!this.socketOpen) {

this.config.closeCallback()

}

return

}

this.retryTimes += 1

console.warn(`[ ${this.retryTimes} ]th reconnect WebSocket...`)

this.init()

}

**消息队列**

添加消息队列，当重连后自动发送缓存消息

openHandle() {

this.retryTimes = 0

this.socketOpen = true

this.retryLock = false

this.messageQueue.map(e => this.send(e))

this.messageQueue = []

}

若发送时断开则先将消息缓存到消息队列中

send(value) {

let data = this.msgWrapper(value)

data = JSON.stringify(data)

if (!this.socketOpen) {

this.messageQueue.push(data)

} else {

if (this.ws) {

this.ws.send({ data })

} else {

wx.sendSocketMessage({ data })

}

}

}

**辅助函数**

添加一些包裹消息格式的工具函数

messageIndex = 0;

helper = {

isPlainObject: val =>

Object.prototype.toString.call(val) === '[object Object]',

nextId: () => {

this.messageIndex += 1

return this.messageIndex

},

id: () => Date.now() + '.' + this.helper.nextId()

};

msgWrapper(data) {

let msg = data

if (this.helper.isPlainObject(msg)) {

if (msg.type) {

return msg

} else {

return this.msgWrapper({ type: 'message', msg, id: this.helper.id() })

}

} else {

return this.msgWrapper({ type: 'message', msg, id: this.helper.id() })

}

}

**完整代码**

export default class WXWebSocket {

messageQueue = []; // 消息队列

retryLock = false; // 避免多次同时重连

socketOpen = false;

manualClose = false; // 主动断开标记

constructor(config) {

this.config = config || {}

// 重连间隔

this.retryInterval =

this.config.retryInterval && this.config.retryInterval > 100

? this.config.retryInterval

: 3000

// 重连次数

this.retryCount = this.config.retryCount || 5

this.init()

}

init() {

let st = this.connect()

this.listen(st)

this.manualClose = false

}

connect() {

let st = wx.connectSocket(this.config.server)

console.log('current socket: ', st)

return st

}

listen(st) {

// 添加监听事件

if (st !== undefined) {

// 若存在SocketTask，则要通过readyState判断状态

// CONNECTING: 0

// OPEN: 1

// CLOSING: 2

// CLOSE: 3

this.ws = st

this.ws.onOpen(() => this.openHandle())

this.ws.onError(() => this.errorHandle())

this.ws.onClose(() => this.closeHandle())

this.ws.onMessage(res => this.messageHandle(res))

} else {

wx.onSocketOpen(() => this.openHandle())

wx.onSocketError(() => this.errorHandle())

wx.onSocketClose(() => this.closeHandle())

wx.onSocketMessage(res => this.messageHandle(res))

}

}

close() {

this.manualClose = true

if (this.ws) {

this.ws.close()

} else {

wx.closeSocket()

}

}

send(value) {

console.log('send value: ', value)

let data = this.msgWrapper(value)

data = JSON.stringify(data)

if (!this.socketOpen) {

// add new message to queue

this.messageQueue.push(data)

} else {

if (this.ws) {

this.ws.send({ data })

} else {

wx.sendSocketMessage({ data })

}

}

}

openHandle() {

console.info('WebSocket connected')

this.retryTimes = 0

this.socketOpen = true

this.retryLock = false

this.messageQueue.map(e => this.send(e))

this.messageQueue = []

}

errorHandle() {

console.error('WebSocket error')

this.socketOpen = false

}

closeHandle() {

console.info('WebSocket closed')

this.socketOpen = false

this.retryLock = false

// 不论是error还是close都会触发close事件，统一在这里进行重连

// 初次连接失败不进行重连(失败不会进入到onOpen的监听事件中，那时未声明retryTimes变量)

this.retryTimes !== undefined && this.reconnect()

}

reconnect() {

if (this.retryLock) return

this.retryLock = true

// 若manualClose为true，表明不是主动断开

if (!this.manualClose) {

// 开始重连

setTimeout(() => {

this.retry()

}, this.retryInterval)

}

}

retry() {

if (

this.socketOpen ||

(this.retryTimes > 0 && this.retryCount <= this.retryTimes)

) {

console.warn(`end reconnect. reconnect ${this.retryTimes} times`)

if (!this.socketOpen) {

this.config.closeCallback()

}

return

}

this.retryTimes += 1

console.warn(`[ ${this.retryTimes} ]th reconnect WebSocket...`)

this.init()

}

messageHandle(res) {

this.config.responseCallback(res)

}

msgWrapper(data) {

let msg = data

if (this.helper.isPlainObject(msg)) {

if (msg.type) {

return msg

} else {

return this.msgWrapper({ type: 'message', msg, id: this.helper.id() })

}

} else {

return this.msgWrapper({ type: 'message', msg, id: this.helper.id() })

}

}

messageIndex = 0;

helper = {

isPlainObject: val =>

Object.prototype.toString.call(val) === '[object Object]',

nextId: () => {

this.messageIndex += 1

return this.messageIndex

},

id: () => Date.now() + '.' + this.helper.nextId()

};

}

**使用**

**创建连接**

let socketTask = new WXWebSocket({

server: this.wsServerOption,

responseCallback: e => {

let { data } = e

let { msg } = JSON.parse(data)

this.msgStack.push(msg)

},

closeCallback: () => {

this.socketTask = null

}

})

this.socketTask = socketTask

**发送消息**

sendWSMessage(msg) {

this.msgStack.push(msg)

this.socketTask && this.socketTask.send(msg)

},

**关闭连接**

closeWS() {

if (!this.socketTask) return

if (this.socketTask.socketOpen) {

this.socketTask.close()

this.socketTask = null

}

}

* **原文作者：**[yrq110](http://yrq110.me/)
* **原文链接：**<http://yrq110.me/post/front-end/wxapp-ws-practice/>
* **版权声明：**本作品采用[知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议](https://creativecommons.org/licenses/by-nc-nd/4.0/)进行许可，非商业转载请注明出处（作者，原文链接），商业转载请联系作者获得授权。

**See Also**

* [微信小程序开发中的代码片段总结](https://yrq110.me/post/front-end/code-snippets-of-wxapp/)