**小程序之websocket的使用**

[闲余幽梦](https://www.jianshu.com/u/18cace04fb2d)关注

0.1032018.05.28 23:28:15字数 1,162阅读 14,475

最近工作中需要实现在小程序中的玩家对战功能，因此需要使用websocket来实现用户之间的数据传递和实时对战功能。随笔主要记录自己工作中使用websocket遇到的问题和解决方法。

**onSocketMessage使用**

在wepy.page创建的当前页面中通过使用onSocketMessage来对服务端socket的返回进行监听，但当我在另一个页面中通过sendSocketMessage发送数据时发现，原先运行后的onSocketMessage监听一直存在，并不会随着页面的跳转消失。如：

// 页面一

wepy.onSocketMessage(function(res) {

console.log('组件收到内容：' + res.data)

})

// 页面二

wepy.sendSocketMessage(function(res) {

data: JSON.stringify(data) // data为传送的数据

})

测试发现当页面跳转到页面二进行触发的时候，页面一的监听结果打印出来了。即在开发不同的页面时，需要在当前页面写onSocketMessage来进行监听，覆盖之前其他页面中的onSocketMessage（一般性当前页面用到sendSocketMessage自然也会用到onSocketMessage，所以也不会存在什么问题）。但当onSocketMessage和SocketTask进行混合使用的时候就需要小心了。

**SocketTask**

SocketTask即WebSocket 任务，可通过wepy.connectSocket()接口创建返回，如：

// 创建SocketTask

let SocketTask= wepy.connectSocket({

url: 'ws://localhost:8181'',

header: {

'origin': '',

'content-type': 'application/json'

},

method: 'GET',

success () {

console.log('连接成功')

}

})

这样创建的SocketTask基本可以替换小程序中的wepy.connectSocket这一类的使用。但SocketTask的使用存在一个问题，即它是动态绑定的，有点像jQuery中的on绑定，会出现累计绑定。当我在页面的onLoad中直接书写时，如:

socketTask.onMessage((res) => {

console.log(111)

})

测试打印结果发现，当页面多次运行的时候(不是重新编译)，打印次数累加了，第一次打印一次，第二次打印两次......没有出现每次打印一次的效果。最后我就动态的在全局下扩展了一个变量来判断，只要小程序不重启，变量就不会清除，修改代码如下：

let friendGameMath = this.$parent.globalData.friendGameMath

if (!friendGameMath) {

this.$parent.globalData.friendGameMath = 1

socketTask.onMessage((res) => {

console.log(111)

})

}

测试发现终于多次运行也只打印一次了。

在使用sendSocketMessage发送指令的时候，一定要将发送指令发在onSocketOpen的回调函数中，不然运行会报错，提示websocket未连接。wepy.onMessage最好写在onShow中，onLoad这个生命周期在页面返回的时候不会触发，可能导致写在当前页面的websocket不会开启监听，一直沿用之前的。socketTask在socket断开后会重新创建，所以在断开以后需要对之前扩展的变量初始化。不然重连后之前的绑定会失效。

**onSocketMessage和socketTask的混合使用**

在我们通过wepy.page创建页面时，考虑页面的复杂性，有时会将页面进行组件化。在不同的组件中使用websocket时，由于一个页面只能有一个onSocketMessage，当在不同的组件中使用onSocketMessage时，只会有一个生效，所以一般的方法就是在页面主体部分进行数据监听再通过父子组件通信或wepy-redux来实现这一类的功能。但如果组件中的内容都是通过websocket来传递数据的话，我觉得可以考虑使用onSocketMessage和socketTask的混合使用来实现。如下

// 主体

export default class PlayGame extends wepy.page{

....

onLoad () {

wepy.onSocketMessage((res) => {

console.log(1111)

}

}

....

}

// 组件

export default class PlayGame extends wepy.component{

....

onLoad () {

let friendGameMath = this.$parent.globalData.friendGameMath

if (!friendGameMath) {

this.$parent.globalData.friendGameMath = 1

socketTask.onMessage((res) => {

console.log(2222)

})

}

}

....

}

这样就可以在主体和组件中都启动一个socket监听，来负责各自的业务逻辑和页面效果，不需要通过父子通信或wepy-redux来进行组件之间的传值了。

**websocket重连**

如果希望websocket连接一直保持，我们会在close或者error上绑定重新连接方法。

// 重连函数reconnect

ws.onclose = function () {

reconnect();

};

ws.onerror = function () {

reconnect();

};

我们可以在每一个组件中使用上述方式进行监听连接是否断开，在重新建立连接。但每个页面组件中都设置相同的代码会导致代码的臃肿且后期不好更改。所以考虑只需要实现监听连接是否断开，决定将这部分相同部分写在wepy的Mixin混合中，在app.wepy中导入一次即可。  
在mixin中的代码如下：

wepy.onSocketClose((res) => {

console.log('websocket关闭')

this.globalData.socketOpen = false

})

wepy.onSocketOpen((res) => {

console.log('WebSocket连接已打开！')

this.globalData.socketOpen = true

})

心跳包：

pdSocketOpen (data) {

if (this.$parent.globalData.socketOpen) {

wepy.sendSocketMessage({

data: JSON.stringify(data),

success: () => {

console.log('getGameView:success')

}

})

return

}

setTimeout(() => {

this.pdSocketOpen(data)

}, 633)

}

通过在监听页面全局变量中boolean值，实现页面socket的重新连接和发送消息。

**小程序服务端模拟**

服务端采用的express框架和ws模块，在服务端启动websocket

var express = require('express');

var app = express();

var ws = require('ws').Server;

var wss = new ws({port: 8181}); // 小程序 connectSocket的url地址

// 模拟数据

var subjects = [{

cmd: 'reqq',

title: 'aaa',

matter: 'asdasfasdasd',

result: ['新西兰', '加拿大', '巴西', '澳大利亚'],

trueRes: '澳大利亚'

}]

wss.on('connection', function (ws) {

console.log('client connected')

ws.on('message', function (message) {

let obj = JSON.parse(message)

if (obj.cmd == 'excans') {

wss.clients.forEach(function (client) { // wss.clients表示连接的所有客户端socket

let data = JSON.stringify(obj)

client.send(data)

})

} else if (obj.cmd == 'reqq') {

let index = obj.index

wss.clients.forEach(function (client) {

let data = JSON.stringify(subjects[index])

client.send(data)

})

}

})

})

app.listen(3000, function () {

console.log('listen 3000')

})

以上皆为个人观点，如有错误，请大家指正。  
微信小程序中的websocket参考文档： <https://developers.weixin.qq.com/miniprogram/dev/api/network-socket.html>