链滴

【小程序 IM】使用 Node-WebSocket 实现微信小程序 IM 即时通信服务

作者：yf98

原文链接：https://ld246.com/article/1596761591071

来源网站：链滴

给小程序接入IM即时通信的功能，本来打算接入易信或者腾讯云的sdk，但是太贵了，负担不起，最使用的是node后端，ws用的 node-websocket。

主要分以下几个部分

1.小程序端聊天记录获取，小程序收发消息

2.node端收发消息给指定用户，添加聊天记录到数据库

3.部署服务

后端搭建websocket服务

首先安装两个node包

首先是WS包，搭建服务。

npm install nodejs-websocket

其次是request包，用来请求接口的。

npm install request

启动服务

这里要注意的是

conn

服务一旦启动后，每次新用户加入都会会拥有一个唯一的连接：conn，为了防止有的用户离线了，进来无法继续使用，将user和conn匹配，每个新用户都存储到users里，每次用户离线后，再次进入就更新连接：conn。

关于聊天记录存储

用户从客户端打开后，拥有conn后，发送新消息，首先存储到本地的state里，ws接收到后，发送指的用户conn，如果该conn无法连接（用户不在线），则将消息存储到数据库，标记未读，如果用户线则，存储到数据库后标记已读后，通过ws推送到指定的conn用户，用户接收到消息，存储到本地的tate里。下次进入，先拉数据库的历史记录就好了。在其他页面，如果想提醒用户，则查询标记未读消息即可。

心跳包

由于ws连接有时候不稳定，我们需要定期发送一个没用的包去激活连接服务，防止断开，也可以修改ginx的超时。我使用的是发送心跳包。后面会写

用户来源检查

ws参数，我使用的是url的path来确定用户id，此外通过客户端的login获取的code来确定用户是否是

过小程序过来的，这里用到的request包就是去请求wx的API了。

内容安全检查

我用的是珊瑚API。

看一下大概的代码，细节已抹去，MiniApi是我写的一个微信接口工具，获取微信相关接口数据。

```
var ws = require("nodejs-websocket")
var MiniApi = require('./miniAppApi');
//获取token
(async () => { MiniApi.access_tokenObj = await MiniApi.getAccessToken(); console.log(MiniAp
) })()
var port =9999;//本地端口，自选
let users = [];//存储当前在线用户
const start_time = new Date();//打个时间
console.log('容器服务初始化:'+ `${start_time.getFullYear()}-${start_time.getMonth() + 1}-${start
time.getDate()}${start_time.getHours()}:${start_time.getMinutes()}:${start_time.getSeconds()}`);
/我是用的docker部署的，后面会说。

//开始写ws服务
var server = ws.createServer(async function (conn) {
    console.log("ws启动...");
    //入参
    let param = conn.path;
    let from_user_openid = 来信人;
    let to_user_openid = 收信人;
    let code = 客户端发来的code;
    console.log('发送人：' + from_user_openid);
    console.log('收信人：' + to_user_openid);
    console.log('code:'+ code);
    if(!from_user_openid || !to_user_openid){
        conn.close();
        console.log('参数缺失')
        return;
    }
    //检查用户合法
    if(! await MiniApi.checkCode(code)){
          conn.close(1000,'无权限');
          console.log('用户不合法，拒绝连接')
          return
    }
    let mes = {};
    let isExit = false;
    for (let v in users) {
        if (users[v].from_user_openid === from_user_openid) {
            isExit = true;
            //已存在该用户,更新连接对象
            users[v].conn = conn;
            break;
        }
    }
    //新加入用户
    !isExit && users.push({ from_user_openid, conn });
```

```
console.log('当前整个WS承载人数：' + users.length + '人')
conn.sendText(JSON.stringify({
    data: 'link ok',
    status: 200,
    type: 'init'
}))



//向客户端推送消息 每当有用户发送消息 该回调执行
conn.on("text", async function (str) {
    str = typeof str === 'object' ? str : JSON.parse(str);
    if(str.type === 'heart'){
        console.log('心跳检测')
    //忽略心跳包
        return;
    }
    mes = str;
    mes.status = 200;
    //发送给目标用户
    console.log('给目标用户发送消息');
    console.log(str.to_user_openid);
    console.log('当前整个WS承载人数：' + users.length + '人')
    for (let v in users) {
       查询是否在users里
        if (users[v].from_user_openid === str.to_user_openid) {
            //存储到云端数据库
            try {
        users[v].conn.sendText(JSON.stringify(mes))
                await MiniApi.addMessage(true);
            } catch (e) {
                //用户可能断开连接了，发送离线消息
        await MiniApi.addMessage(false);
            }

            return ;
        }
    }
    //用没有打开过连接 直接存储到云端数据库
    try {
        await MiniApi.addMessage(false);


    } catch (e) {
        console.log(e);
    }
});

//监听关闭连接操作
conn.on("close", function (code, reason) {
    console.log("关闭连接");

    console.log(code, reason);
```

```
    });

    //错误处理
    conn.on("error", function (err) {
        console.log("监听到错误，异常断开");
    });
}).listen(port);
```

客户端WS

打开连接

```
wx.connectSocket({
        url: 'wss://xxx:9999' + '/' + this.data.openid + '?to_user_openid=' + this.data.to_user_o
enid + '&code=' + code,
        header: {
            'content-type': 'application/json'
        }
    });
```

各种回调

监听打开

```
wx.onSocketOpen((result) => {
        console.log(result)
        this.socketOpen = true;
        heartHandler = setInterval(() => {
            this.sendHeart();
        }, 30000)
    });
```

监听消息

```
wx.onSocketMessage((result) => {});
```

监听异常

```
 wx.onSocketError((result) => {
        console.log(result)
    });
    wx.onSocketClose((result) => {
        console.log(result)

    });
```

发送消息

```
 wx.sendSocketMessage({
        data: JSON.stringify({
            message,
```

```
            //发送人
            from_user_openid: this.data.openid,
            type: 'text',
            //日期
            date,
            //排序用时间
            created_at,
            //目标用户
            to_user_openid: this.data.to_user_openid

        })
    });
```

每次发消息，滚动到底部

```
scrollBottom() {
    wx.createSelectorQuery().select('#talk__body').boundingClientRect(function (rect) {
        console.log(rect)
        wx.pageScrollTo({
            duration: 500,
            scrollTop: rect.height
        })
    }).exec()

  },
```

客户端UI

两侧布局，其实很简单，每个消息都有收信人和发信人，根据这个向右布局： margin-left: auto;

```
<view  scroll-x="{{true}}" class="talk__body" id="talk__body">
  <block wx:for="{{talks}}" wx:key="index">
    <view class="talk__item {{item.from_user_openid !== openid ? 'talk__left':'talk__right'}}">
      <view class="talk__content">
        <block wx:if="{{item.from_user_openid !== openid }}">
         <image bindtap="happy" class="talk__avatar"   src="{{otherInfo.userInfo.avatarUrl}}
></image>
         <view class="talk__text">{{item.message}} </view>
        </block>
        <block wx:else>
         <view class="talk__text">{{item.message}}</view>
         <image class="talk__avatar"  src="{{userData.userInfo.avatarUrl}}"></image>
        </block>
      </view>
      <view class="talk__time">{{item.date}}</view>
    </view>
  </block>
</view>
```

wxss:我加入了夜间模式的适配

```
/* miniprogram/pages/chat/room/room.wxss */
page{
  background-color: #f2f2f2;
```

```
}
.footer{
    position: fixed;
    bottom: 0;


    width: 100%;
    left: 0;
    background-color: #fff;
    box-shadow: 0 2px 2px 2px rgba(0,0,0,0.1);
}

.talk__right .talk__content{
    display: flex;
  margin-left: auto;
  align-items: center;
    align-content: center;
}
.talk__left .talk__content{
    display: flex;
    margin-right: auto;
    align-items: center;
    align-content: center;
}
.talk__time{
    opacity: .4;
    text-align: center;
    font-size: 20rpx;
    margin-top: 10px;

}
.footer__send{
    display: flex;
    margin: 6px 0;
    align-items: center;
}
.footer__send__input{
    width: 80%;
    margin-left: 2.5%;
    height: 40px;
    padding: 5rpx;
    box-sizing: border-box;
    padding-left: 15rpx;
    border-radius: 5px;
    background-color: rgb(235, 232, 232);

}
.footer__send__plus:hover{
    background-color: rgb(1, 68, 1);
}
.footer__send__plus{
    width:15%;

    opacity: .7;
```

```
    height: 40px;
    background-color: green;
    border-radius: 5px;
    padding: 5rpx;
    display: grid;
    place-content: center;
    box-sizing: border-box;
    text-align: center;
    color: #fff;
    margin-left: 1%;
    margin-right: 1%;
}
.talk__avatar{
    width: 100rpx;
    border-radius: 200rpx;
    height: 100rpx!important;
    background-color: #fff;
}

.talk__text{
    border-radius: 15px;
    padding: 20rpx;
    opacity: .9;
}
.talk__right .talk__text{
    background-color: green;
    color: #fff;
    opacity: .7;
    margin-right: 10px;

}
.talk__left .talk__text{
    background-color: #fff;
    margin-left: 10px;

}

.talk__item {
    width: 98%;
    display: flex;
    margin-left: 1%;
    flex-direction: column;
    margin-top: 15px;
}
.talk__body{
   overflow: scroll;
   height: auto;
    padding-bottom: 60px;

}

@media (prefers-color-scheme: dark) {
.footer{
    background: #2e2d2d;
```

```
}
.footer__send__input{
    background:  #474747;
}
.talk__body{
    background: #2e2d2d
}
.talk__left .talk__text{
    background-color:#474747;
    margin-left: 10px;

}


}
```

容器部署

把项目打包成镜像下面是Dockerfile

```
FROM node:10.15
RUN echo '正在复制项目文件到镜像/app目录'
COPY . /app/
RUN echo '设置工作目录'
WORKDIR /app
RUN echo '开始安装NPM包'
RUN  npm install
RUN echo '安装PM2环境'
RUN npm install pm2 -g
RUN echo 'expose设置9999端口'
EXPOSE 9999
RUN echo '运行服务'
ENTRYPOINT [ "pm2-runtime","init.js" ]
CMD ["bash"]
```

使用docker build -t xxx 创建镜像

使用docker run --name -itd -p 端口:端口 -v 主机目录:容器目录映射 镜像名字

然后每次更新，只需要调整主机文件就好了。

最后就是好好优化调整一些逻辑与安全性的地方，大致的结构就是这样。

可以扫码文章下面的签名二维码体验这个IM。