2015年10月10日

通过Express、socket.io和AMap构建具有分享地理位置功能的聊天应用

HTML\CSS学习考核

一、引文

在最初与导师沟通的时候，就HTML\CSS方面就给自己定了一个学习目标，和考核标准。当时我就想着做一个具有地理位置分享功能的应用，经过一天的思考后，也就把做这个聊天应用作为了HTML\CSS的考核标的。

HTML\CSS学习的内容：CSS的基本原理和CSS3的一些新的特性，CSS学习重点也主要集中在CSS的animation，transform，Transition属性。关于HTML，主要学习了HTML5中的新特性，包括Canvas，Geolocation，Form API, WebSocket，Worker，LocalStorage等。在WeChat应用中使用到了大部分上面的新技术。

HTML\CSS学习的目标：通过对HTML\CSS的学习，独立完成具有即时聊天、位置分享功能的应用。解决在实现应用中遇到的bug，并对应用以后的改进提出优化方案。

WeChat应用已经部署到了百度开放云：wechat8.duapp.com。

代码地址：https://github.com/Jocs/wechat8

二、应用所使用到的技术和原理

2.1、应用的实现原理

利用Express在NodeJs环境下搭建一个简单的服务器，使用socket.io构建WeChat应用的通讯模块。Express是一个在NodeJs环境中用于快速构建Web应用的框架，该框架封装在NodeJs原生模块Http上的，WeChat应用的后端服务器也就是通过Express来搭建的，因为NodeJs不是本阶段的学习内容，因此这儿也就不再赘述Express的实现。

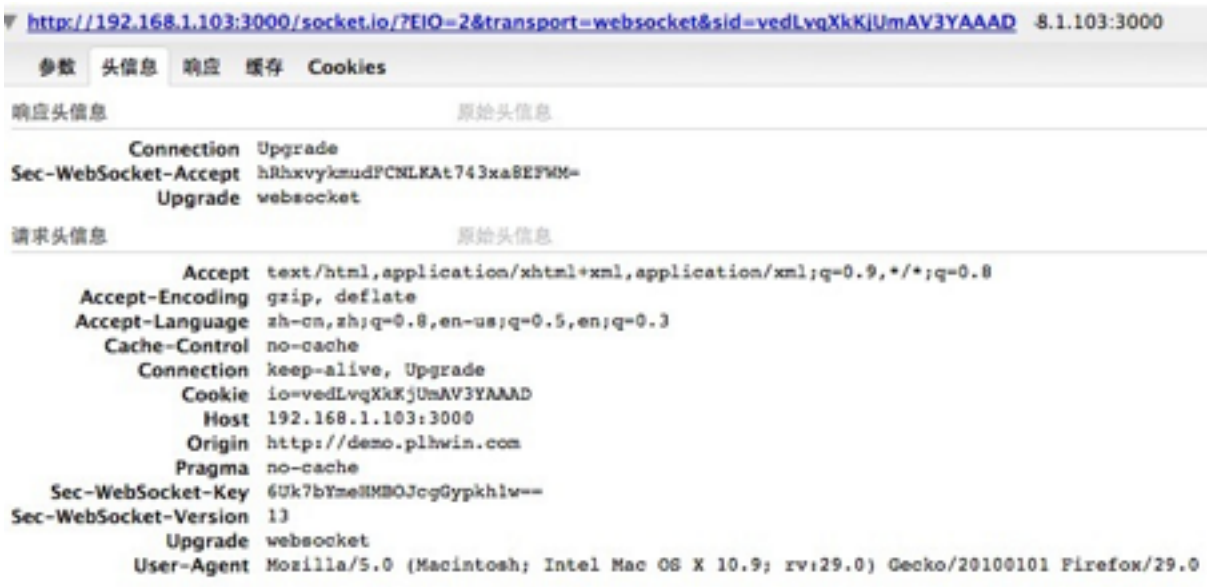
socket.io也是NodeJs的npm包中的一个模块，该框架封装了HTML5的WebSocket之上，通过WebSocket的全双工通讯信道，实现了客户端——服务端——客户端的即时通讯，当浏览器不支持WebSocket的时候，socket.io会优雅降级，使用轮询(Polling)和Comet技术进行替换。

2.2、应用使用到的技术

2.2.1 WebSocket

WebSocket是HTML 5新增的功能，其主要作用就是Web应用的实时通讯，在WebSocket之前，也有一些实时通讯方法，包括轮询、长轮询、流等解决方案，但是这些方案都是在半双工的HTTP协议上模拟全双工通讯通道，这样就不可避免的会发送不必要的请求，或者HTTP请求或响应头部中包含很多不必要的信息。

而WebSocket是构建在TCP协议之上的一种网络传输协议，因此WebSocket更像TCP传输数据的形式，数据按序到达，并且是双向的。为了构建WebSocket连接，首先需要发送一个HTTP请求，在HTTP请求的初次握手时，将HTTP协议升级成WebSocket协议，下图是一个典型的WebSocket请求，我们可以看到请求头信息中有个Upgrade : websocket。表明该请求将升级成WebSocket协议。



2.2.2 HTML 的GeoLocation API

Geolocation是HTML5中用来获取地理位置的API,HTML5 中的Geolocation API 主要就包括两个，getCurrentPosition 和 watchPosition。

HTML 5获取位置信息的主要原理是利用WiFi热点、路由器信息、ip地址、GPS定位、基站、网络提供商等来进行定位的，一个需要定位的浏览器应用当发起定位请求时，浏览器首先会对请求进行拦截，询问浏览器的使用者是否允许定位，（当然这个可以在浏览器中设置的，推荐是每次需要定位时都询问），如果允许浏览器进行定位，那么浏览器就会向地理位置提供服务器发送位置请求。最后返回页面，发送请求是个异步过程，因此上面两个方法都需要传递回调函数。发回的位置信息根据地理位置、设备连入的WiFi多少，是否是移动设备，ip地址、使用的浏览器等因素而各异。

1）getCurrentPosition：The Geolocation.getCurrentPosition() method is used to get the current position of the device. 根据文档描述，给方法就是用来获取设备的当前位置。

使用语法：

navigator.geolocation.getCurrentPosition(success, error, options)

该方法传三个参数：

success是请求成功的调用函数。接受一个position对象作为参数，这个参数包含了当前地理位置信息。包括latitude、longtitude、精确度、时间戳等。但是用得最多的还是前面三个，经纬度和精确度。

error是请求失败的回调函数。接受一个error对象，这个对象包含了不同的错误码，根据错误码我们就能够知道是哪儿出错了。可以通过error.code 和error.message来获取具体错误信息。

options对象是个可选参数:主要包括以下三个属性设置。

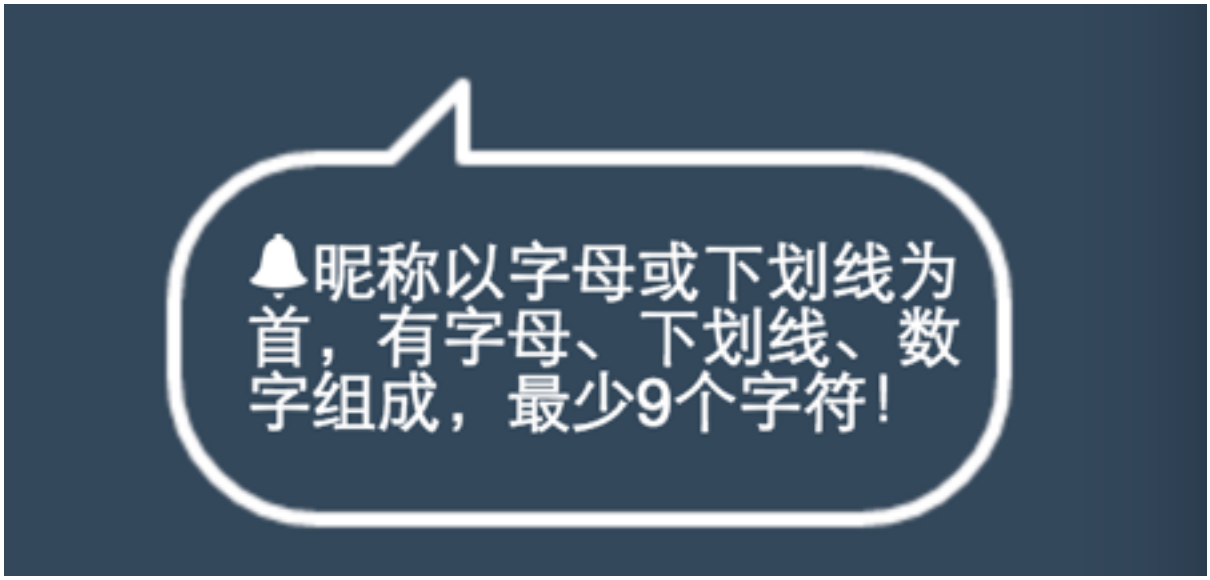
```
{
  // 指示浏览器获取高精度的位置，默认为false
  enableHighAcuracy: true,
  // 指定获取地理位置的超时时间，默认不限时，单位为毫秒
  timeout: 5000,
  // 最长有效期，在重复获取地理位置时，此参数指定多久再次获取位置。
  maximumAge: 3000
}
```

2）watchPosition通过间时调用获取当前位置。传入参数和getCurrentPosition一样。文档如下描述：The Geolocation.watchPosition() method is used to register a handler function that will be called automatically each time the position of the device changes. You can also,optionally, specify an error handling callback function.This method returns a watch ID value then can be used to unregister the handler by passing it to the Geolocation.clearWatch() method.该方法返回一个watch ID， 可以通过Geolocation.clearWatch() 取消事件注册。

在WeChat应用中，我没有直接使用HTML5的GeoLocation API，而是使用了高德地图javascript编写的高德地图API。高德地图浏览器定位是封装了HTML5的GeoLocation API，并对上面的两个方法进行了优化，进行了纠偏，使得返回的位置更加准确，在应用中，主要使用了高德地图的加载地图、浏览器定位和在地图上添加、删除标记的功能。

2.2.3 Canvas

Canvas是我最喜欢的一个HTML5 元素，其实它也就是一个DOM元素。但是他又不简简单单是个DOM元素。在WeChat应用中使用到的Canvas比较少，就在登陆页面，当用户输入错误的时候，会显示一个输入错误提示框。如下图：



上图这个输入框就是通过canvas的lineTo和arcTo等方法画出来的。当然其中的文字不是通过fillText（）和strokeText（）画的了。代码也很简单，这儿就不贴代码了。

2.2.4 CSS Transform

其实整个应用中对CSS Transform的应用不是用来做动画，而仅仅用Transform对未知高宽的元素进行页面居中显示。css代码如下：

```
选择器{
  position: absolute;
  left:50%;
  top: 50%;
  -webkit-transform:translate(-50%,-50%);
  -moz-transform:translate(-50%, -50%);
}
```

transforms属性的使用还需要加浏览器前缀。

2.2.5 CSS 的 table布局（display：table）

在登陆页面中，有一个选择头像的界面，如下：

该头像的布局使用了table布局：

html结构如下：

```
<div id="portrailTable">
```

```
        <!-- <div class='portrailRow'>
                <div class='portrailCell'>
                        <img src="images/portrail/1.jpg" alt="">
                </div>
                <div class='portrailCell'>
                        <img src="images/portrail/2.jpg" alt="">
                </div>
                <div class='portrailCell'>
                        <img src="images/portrail/3.jpg" alt="">
                </div>
                <div class='portrailCell'>
                        <img src="images/portrail/4.jpg" alt="">
                </div>
        </div>
        ……
  </div>
```

就是把整个4*5头像矩阵看成一个table，每四个头像就是一行（portrailRow），总共有5行，每一行中的每一个头像就是一个cell（class='portrailCell'）；

css代码如下：

```
 #portrailTable{
        display: none;
}
 .portrailRow{
        display: table-row;
}
 .portrailCell{
        display: table-cell;
        width: 25%;
        cursor:pointer;
        border-radius: 50%;
        overflow: hidden;
        box-sizing:border-box;
        border:4px solid lightgray;
}
```

我觉得table布局很适合这种图片页面的布局。

三、在构建WeChat的过程中遇到的问题及解决方案

在写代码的过程中遇到很多问题，有些是简单的typing error，有些是对知识的了解不够，以至于出现bug很难被排除，但无论是遇到什么问题，都是一个既痛苦又难忘的过程。这部分将分三块来写，首先会对我做这个应用的需求做一个分析，其次通过代码来描述整个的写作过程，最后叙述我在做应用中遇到的问题。

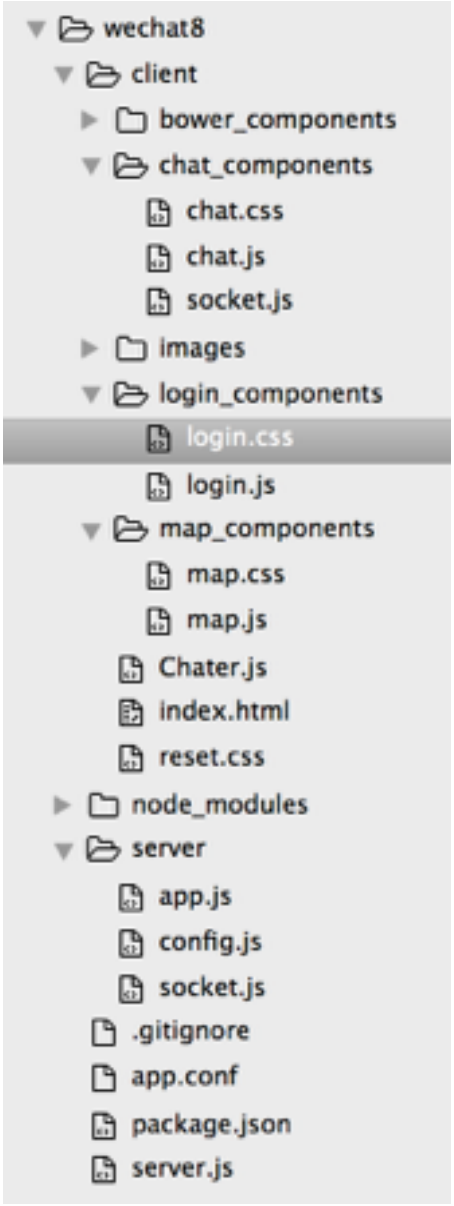
3.1 应用的需求

当然根据本部分的考核，就应该以HTML和CSS为重点，完全使用原生Javascript来写，因此前端代码没有使用Javascript框架，虽然使用了BootStrap，也仅仅是使用了BootStrap的图标，因此布局也完全是使用CSS来写的，这样可以加强对CSS和Javascript的理解，当然后端的话，运用了Express和Socket.io框架。

用户需求呢？做一个聊天工具也是我一直以来的想法，因为WeChat应用主要是满足用户户外及时聊天和分享位置的需求，这也是最常用的户外需求了，因此我也就选择了这两个需求来实现。这也正好利用所学的WebSocket和Geolocation API知识。

3.2 WeChat应用的实现

整个应用的代码结构如下：



主要有两个文件夹，一个是client用来放前端（客户端）文件，一个是server文件夹，用来放服务端代码，服务端代码很少，就两个文件，一个server.js一个server/socket.js。

//server.js

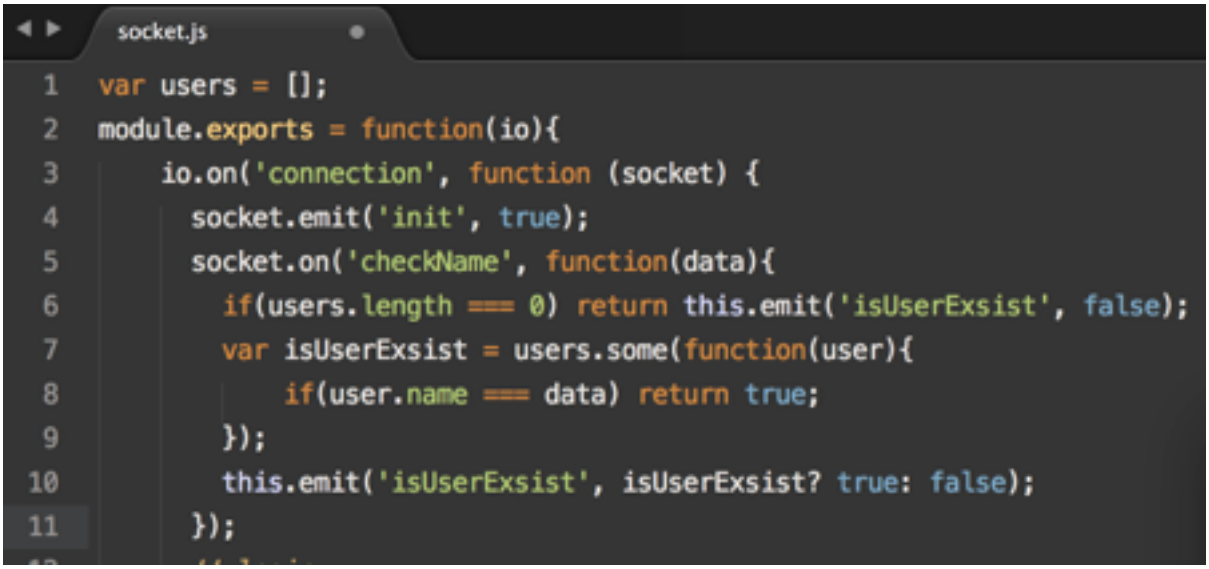
server.js

```
var express = require('express');
var app = express();
var server = require('http').Server(app);
var io = require('socket.io')(server);
var sockets = require('./server/socket.js')(io);

app.use(express.static(__dirname + '/client'));
server.listen(18080, function(){
    console.log('app listen at port: 18080');
});

```

socket.js文件也很简单，主要作用就是消息的中转站，消息从客户端发送过来，根据客户端需求，（是广播还是一对一发送）然后对客户端发来的消息转发给目标客户

```
    // login ------------------------
    socket.on('login', function(data){
      users.push(data);
      socket.emit('users', users);
      socket.broadcast.emit('login', data);
    });
    // handle message event ----------------
    socket.on('message', function(data, fn){
      fn();
      if(data.receiver.name !== '聊天室'){
          socket.broadcast.to(data.receiver.socketId).emit('message', data);
      } else {
          socket.broadcast.emit('message', data);
      }
    });
    // handle disconnect ------------------
    socket.on('disconnect', function(){
      for(var i = 0; i < users.length; i ++){
          if(users[i].socketId === this.id) {
              io.sockets.emit('logout', users[i]);
              users.splice(i, 1);
          }
      }
    });
  });
}
```

端。同时socket.js还监听了用户的链接和断开，当用户断开连接时，服务器广播消息，让其他在线用户知道，这样也便于其他用户对在线用户进行更新（这也是实现在线，离线功能的基础）。



在socket.js文件中，声明了一个users数组，用来存在线用户数据，当然如果用户过多，且数据量过大，最好用户数据保存在数据库中，通过users数组存在，当服务器重启时，也就不存在了，当然这而只是事例。监听用户链接，如果用户连接上就会发送一个init事件，告诉用户已经连接上，同时监听checkName事件，来判断是否有重名。

socket.js剩下的代码如上，当有用户登录，把用户push进users数组中，然后监听和转发消息，监听disconnect事件，在users中删除离线用户。

客户端代码比服务端代码多一些，结构也复杂一些了，通过不同的功能我把前端代码划分了不同功能模块，有login_componengts\chat_components\map_components\images。和三个主文件：index.html，reset.css, Chater.js。index.html主要是用来放应用的所有html代码和代码模板，reset.css对浏览器默认的css样式重置，Chater.js文件中有是一个Chater构造函数，我们可用通过这个构造函数来new一个chater对象，这个对象包含了和聊天有关的方法和属性，包括sendMessage、receiveMessage，displayMessage等。前端代码结构如上，首先我们来说说Chater构造函数：

function Chater(){

```
  this.name = '匿名用户';
  this.portrail = '';
  this.loginAt = '';
  this.lastMessageAt = '';
  this.socket = null;
  this.socketId = '';
  this.activeChater = null;
  this.users = [];
  this.profile = null;
  this.map = null;
  this.marker = null;
}
```

在这个构造函数中定义了，用户姓名，用户头像，登陆事件等信息，然后我们在Chater实例对象的原型上定义了一些方法：

```
Chater.prototype = {
  //程序初始化，this.socketId初始化---
  init: function(url){
        var socket = io.connect(url);
        this.socket = socket;
        this.socket.on('init', function(data){
              if(data === true) this.socketId = this.socket.id;
        }.bind(this));
  },
```

通过init方法我们建立一个socket连接，在实例对象上，我们调用init方法传入url就可以建立一个socket连接了。

当然不可能把Chater上面的所有方法写出来，重点说说sendMessage方法吧；

```
sendMessage: function(value, fileType){
        fileType = fileType || 'text';
        var isShowTime = !!(new Date() - this.lastMessageAt > 1000 * 60);
        var currentChater = document.querySelector('#currentChater');
        var activeRoomUl = document.querySelector('#rooms .active');
        if(fileType == 'text' || fileType == 'image') this.lastMessageAt = new Date();
        var data = {
              showTime: isShowTime,
              fileType: fileType,
              message: value, // 消息内容
```

```
                receiver: this.activeChater,
                sender: this.profile,
                date: this.lastMessageAt.toTimeString().slice(0,8)
        };

        if(fileType == 'text' || fileType == 'image'){
                // 显示自己发送的消息
                this.displayMessage('me', activeRoomUl, data, fileType);
                // 给服务器发送message事件。
                this.socket.emit('message', data, function(){
                        currentChater.textContent = data.receiver.name;
                });
        } else {
                this.socket.emit('message', data, function(){
                        if(data.receiver) currentChater.textContent =
                        data.receiver.name;
                });
        }

  },
```

在sendMessage方法中，我们发送的数据包括如下：

```
var data = {
                showTime: isShowTime,
                fileType: fileType,
                message: value, // 消息内容
                receiver: this.activeChater,
                sender: this.profile,
                date: this.lastMessageAt.toTimeString().slice(0,8)
        };
```

showTime是否显示发送事件，fileType是发送消息的类型，（因此更加恰当的命名应该是messageType）消息类型包括了「text」「image」「position」「typing」「askPosition」「answerAskPosition」等，根据不同的消息类型，在接收到消息的时候也就选在了不同的消息处理方式，具体做法可以参见receiveMessage和displayMessage方法。

login_components\chat_components\map_components\三个组件，每个组件都包含和根据功能划分的css文件和javascript文件。为了避免变量污染，在全局作用域中new了一个Chater对象的实例chater，因此全局变量仅包括chater，Chater，window.onload的事件处理函数(loadLogin, chat, socketRelate, loadMap)。这样就有效地避免了变量污染。

login_components/login.js文件主要用来处理和登陆有关的业务逻辑，包括初始化chater对象，初始化chater.socket对象，和服务器通讯判断添加的用户名是否有重名，保存用户头像的路径地址等，当所有的验证都通过，就展现聊天页面给用户。

chat_components/chat.js主要用来处理聊天页面的一些动画和效果，也就是点击好友就展现好友列表、点击广播，就展现广播消息的聊天界面，点击地图就展现完整且定位的地图界面，也就相当于一个路由文件，只是整个页面都是一个单页面，单URL。因此也没用路由这个词了。

chat_components/socket.js这个文件主要用来处理和聊天有关的业务逻辑，包括初始化好友列表，当有用户登录时更新好友列表，初始化聊天界面，处理发送消息、接收消息、展现消息，监听键盘和鼠标事件。主要是keypress和click事件。而很多方法都写在了chater的原型对象中，因此socket.js看起来很简洁，把所有的复杂逻辑都交给了chater原型中的方法来处理。socket.js主要代码如下：

```
chater.initUserListAndRooms();
  chater.updateLoginAndLogout();
  chater.receiveMessage();
  // 发送消息
  send.addEventListener('click', function(e){
        var value = inputTextInput.value;
        if(value.length !== 0){
              inputTextInput.value = '';
              inputTextInput.focus();
              chater.sendMessage(value);
        } else inputTextInput.focus();
  }, false);
```

map_components/map.js主要处理和分享地理位置、地图有关的业务逻辑，包括初始化地图和geolocation对象。并把初始化的地图对象赋值给chater.map。还处理了发送分享位置请求，接收分享位置请求，当获取到当前位置后发送位置信息给对方的业务。比如A向B发送位置共享，A发送一个askPosition事件，同时把自己的Position通过position事件也发送过去，无论B接收请求或者拒绝请求，B都会发送一条answerAskPosition事件的消息，告诉A位置共享请求的结果，当然如果B接收位置分享，那么B也会同时发送位置监控信息给A，当然拒绝就不会发送位置信息给A了，拒绝也不会展现A发送过来的位置信息。整个分享位置信息的逻辑就是上面这样的，说

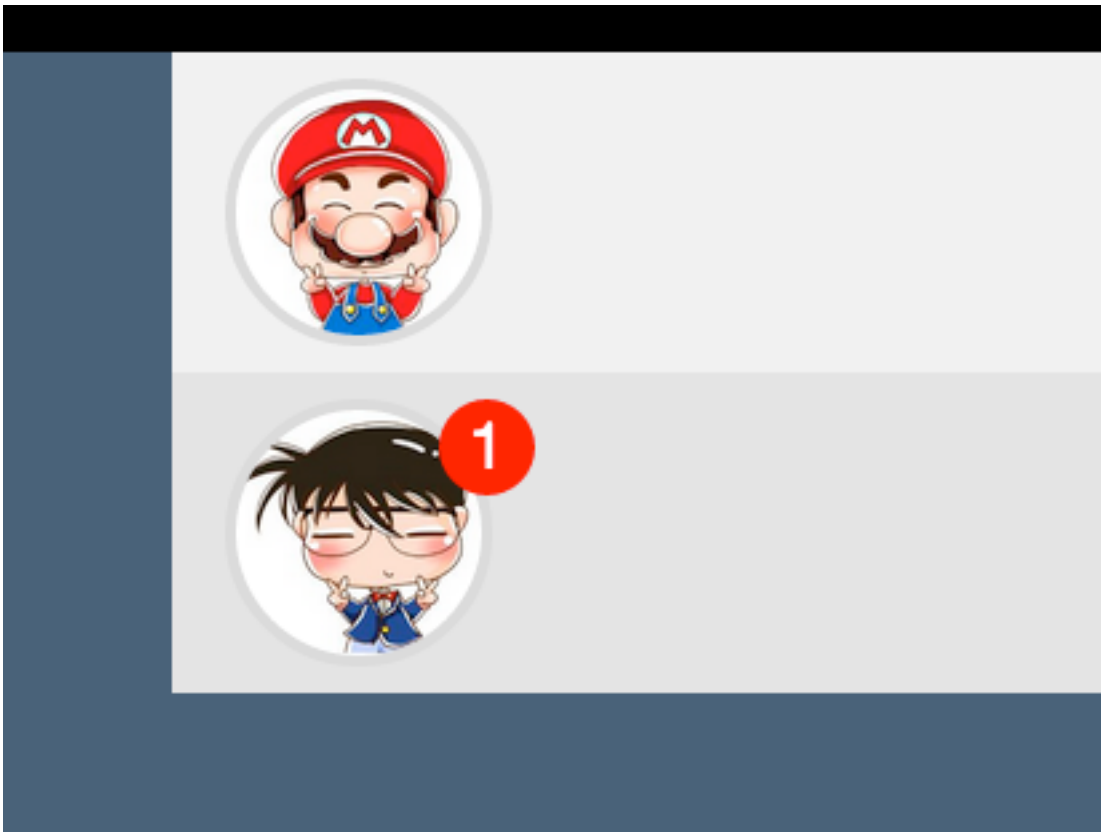
起来挺简单的，但是实际做起来，也花了一整天的时间，遇到很多问题，遇到的问题将会在本章第三部分详述。

整个应用的代码结构，及实现过程也就是上面这样的。下面将说说我在写代码的过程中遇到的问题。

3.3 WeChat is easy but not easy

通过websocket构建聊天应用，基本原理很简单，但是实际做起来并不easy，我将按实现的功能来写在每个功能中遇到的一些问题。

3.3.1 消息提示功能



消息提示功能困扰了我好久，我一直在思考，提示位置应该展现在哪？怎么来计数未读消息？什么时候增加未读消息？什么时候更新未读消息？什么时候重置未读消息？

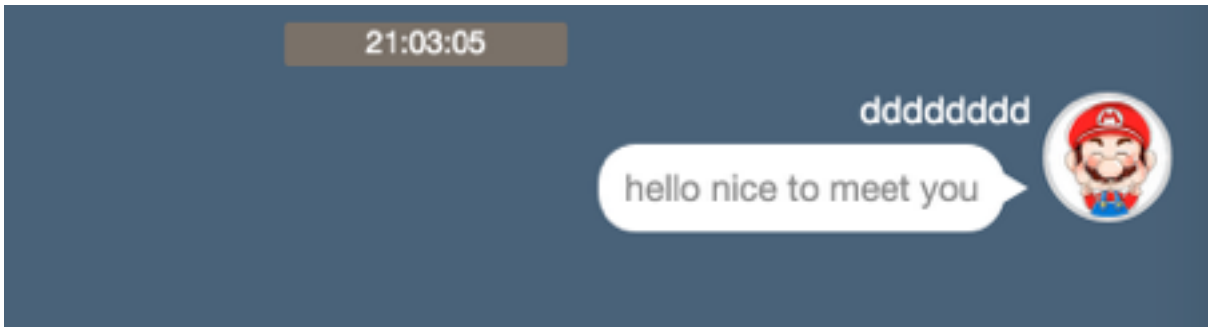
消息提醒放在哪呢？我想过放在title标签中，这样就可以在切换浏览器页面的时候也能够看到提醒了，在title标签中放入新消息提示加入blink效果，应该是个很不错的体验，但是由于时间原因，我还没实现这个效果，我参考了微信的消息提醒，在好友列表页头像的右上角展示未读消息。如下图：

小红圆圈是个span元素，通过border-radius CSS3属性，实现圆形效果。未读详细的计数我是通过把所有在线和离线用户放在了chater.users数组中，然后对每个user添加了一个unreadCount属性，并初始化为0。这样，当我接受到新消息时，判断消息的类型是text，imange、askPosition，我就把该发送用户的未读消息加1。然后调用updateUnreadCount函数，该函数会判断users中是否有未读消息，如果user的未读消息不为0，就显示出来。因此在每次接受到消息，我都会更新未读消息。

当每次点击用户列表项进入聊天界面的时候，就应该重置该用户的未读消息属性为0，并更新未读消息展现。同时在有新用户登录时，也应该对他的unreadCount属性进行重置。这样就完成了整个的未读消息提示功能，不过应用现在关于未读消息依然还存在设计问题，就是聊天室发送的消息依然还会被计数，这应该再次判断消息的接受者是否为『聊天室』，如果是的话，就不要再加1了。

在修改bug的过程中，还遇到了一个现实未读消息的数目比实际情况多很多的问题，找了好久最后才发现，我并没有把「typing」消息排除在外，也就是说，对方只要在键盘上输入，按下一次键盘按键，我这边就会显示一天未读消息，这显然不是我们所希望的，因此在对未读消息计数的时候，对消息类型做了严格的限制。

同时，正在聊天的对象，其发送过来的消息也是不计数的，这也是需要考虑的因



素，不然最后计数结果也会比实际多。

3.3.2 时间、姓名显示遇到的问题

如上图，在头像一侧显示了用户姓名，因为用户都是卡通图像，每次登陆都可以选择不一样的头像，因此谁也不知道谁是谁，还是把姓名显示出来好。在显示时间也遇到了问题，我是判断两条消息的时间间隔，如果时间间隔少于1分钟，我就不显示时

间了，谁也不想聊天屏幕中全是显示的时间吧，但是做出来的效果是都没有显示时间，这又是为什么呢？我是把是否显示时间涉及到了sendMessage函数中，通过现在的时间减去lastMessageAt事件，如果少于一分钟就不显示，结果如果一直在键盘上敲击发送typing事件，或者一直在分享位置，发送position事件，这样事件间隔就很难大于一分钟了，因此，在记录lastMessageAt的时候，应该只记录text，image类型的消息。这样才能够保证消息时间的正常显示。

3.3.3 正在输入状态消息中遇到的问题

正在输入状态，不好截图，因为我一截图，输入状态就消失，可以登录Demo网站试验下。正在输入状态就需要检测输入框的变化，当我使用：

input.addEventListener('change', eventHandler, false)

检测input输入框的变化时候，发现并不能够如我所愿，当输入框一有变化就触发change事件，而往往输入框输入，然后输入框失去焦点的时候才能够检测到change事件，后来我在document上检测keypress事件，判断事件的target是当前输入框，我就认为现在输入框正在输入。代码如下：

```
document.addEventListener('keypress', function(e){
  if(e.target.id === xxx ){…}
}, false)
```

这样的效果比直接检测input上的change事件要好很多，那么什么时候认为停止输入或者没有输入了，通过以下代码监听：

input.addEventListener('blur' , eventHandler, false)

也就是当input失去焦点的时候，认为输入框没有在输入了。然后，如果正在输入的话，我们就发送「typing」类型的消息，如果停止输入就发送「disTyping」类型的消息，接受「typing」和「disTyping」类型消息，当接收到typing消息，显示一个正在输入的状态，当接收到disTyping消息，把刚才显示正在输入的状态remove掉。这样就实现了正在输入状态显示功能。

3.3.4 位置分享功能实现中遇到的问题

位置共享功能类似于微信的共享位置，在聊天对话框中，向对方发送一个共享位置请求，如果对方接受，双方共享位置，如果对方拒绝，对方的位置信息无法获取。代码实现就不是上面两句话就能搞定的，首先需要初始化一张地图，这儿我选择的高德地图，因为高德地图叫做AMap，而百度地图叫做BMap，A比B大，所以我选择了A，等等，上面只是一个玩笑，选择高德地图的真正原因还是因为AMap比BMap功能更强大，开放的API更多，而且高德地图的浏览器获取当前位置有两个方法，是getCurrentPosition和watchPosition。而BMap只有一个方法，就是getCurrentPosition。而WeChat应用需要实时监控当前位置，因此watchPostion更加适合。因为watchPostion会根据你的设备位置是否变化，而决定是否向位置服务器发送获取位置请求，因此很难通过getCurrentPosition方法对watchPosition进行模拟。我曾经也做过这样的事，通过promise对象，向位置服务器发送一个位置请求，当位置服务器返回位置信息的时候，resolve promise对象，然后间隔3s再向位置服务器发送位置请求，这样通过getCurrentPosition来模拟watchPosition，但是这样也有一个弊端，就是造成了很多不必要的请求，就是当设备位置没有发生变化的时候，依然向位置服务器发送了位置获取请求，当然这不是我想要的，所以选择了AMap。

高德地图所有的属性、方法都在AMap这个命名空间里，通过new AMap.Map（）来初始化一张地图，代码如下：

```
chater.map = new AMap.Map('map', {
        resizeEnable: true,
        level: 16
    });
```

高德地图还有个优点就是，为了精简代码量，我们需要什么模块加载什么模块，比如WeChat应用中我们需要位置定位的模块，然后我们就需要map原型上的plugin方法，加载AMap.Geolocation.代码如下：

```
chater.map.plugin('AMap.Geolocation', function() {
        geolocation = new AMap.Geolocation({
            enableHighAccuracy: true,//是否使用高精度定位，默认:true
            timeout: 10000,          //超过10秒后停止定位，默认：无穷大
            maximumAge: 0,           //定位结果缓存0毫秒，默认：0
```

```
            convert: true,          //自动偏移坐标，偏移后的坐标为高德坐标，
            showButton: true,       //显示定位按钮，默认：true
            buttonPosition: 'LB',   //定位按钮停靠位置，默认：'LB'，左下角
            buttonOffset: new AMap.Pixel(10, 20),//定位按钮与设置的停靠位置的偏移
            showMarker: true,       //定位成功后在定位到的位置显示点标记
            showCircle: false,      //定位成功后用圆圈表示定位精度范围，默认：true
            panToLocation: true,    //定位成功后将定位到的位置作为地图中心点
            zoomToAccuracy: true    //定位成功后调整地图视野范围
        });
    });
```

这样我们就可以使用geolocation对象上面的getCurrentPosition和watchPosition方法了。

位置分享的难点是在消息类型的设计，刚开始我以为一个「position」类型的消息就可以了，后来发现不行，我们还得有一个「askPosition」类型消息来请求共享位置，然后还得有一个「anwserAskPosition」类型消息来回复接受或拒绝位置共享。首先发送一个askPosition类型的消息来向对方请求位置，对方通过answerAskPostion来反馈位置共享消息请求。

现在位置分享功能依然有一个小问题，就是在接受位置共享的一方如果应用界面是好友列表界面，当前接受了位置共享请求，也进入了位置共享的聊天界面，但是接受位置的用户就会半离线状态，只能接受消息，不能够发送消息，重新登录就又好了，如果是在聊天对话界面接受位置共享就不会出现这个问题，到现在我还不知道这个是服务器的原因，还是socket.io框架的原因，还是代码的原因，因为这个问题也不总是出现。在以后优化的过程中还需进一步改进。

3.3.5 为未来添加的元素添加事件

在聊天应用中，所有的聊天消息都是通过Javascript动态生成的，或者使用index.html文件中html片段作为模板。因此很多元素都是动态添加，那么怎么向未来元素添加时间呢？当然直接添加的事件是不行的，因为添加事件的未来元素还不存在呢。

可以通过事件冒泡来解决这个问题，比如在未来元素的父节点添加事件监听，最后判断e.target是否是新添加的元素。

四、应用可待优化和改进的地方

5.1 消息提醒的优化

当然应用还有很多需要改进的地方，首先就是消息提醒只要有新的消息，就可以在title标签中显示「新消息（2）」，（括号中是消息的条数），然后通过setInterval方法来交替显示新消息和原来的titile，做出blink效果。当我们点击未读消息时，title恢复成为原来的title。

5.2 语音和视频的支持

以后有时间的话，学习下WebRTC技术，可以添加到WeChat应用中，做出一个可以视频聊天的应用。

5.3 地图应用现在只是共享位置和获取当前位置的功能

在以后可以添加位置查询、路线规划功能，这个API 在AMap中都有提供。

5.4 发送离线消息功能

发送离线消息功能我的初步设想是，首先我把要发送的「text」类型消息，推到一个消息数组中，比如user.preToSendMessage = [];，每次发送消息是，都把消息作为该数组的一个元素，存入其中，然后判断用户是否在线，如果在线，就遍历该数组，发送所有消息，如果对方不在线，就储存消息，同时监听login事件，如果有人登陆，就遍历users对象中找到新登录的用户，遍历该用户的preToSendMessage数组，把所有的离线消息发送出去。当然如果有服务器端数据库就不用这样了，存服务器上的数据库更好了，因为存在应用数组中，当关闭应用或浏览器，也就没有了，当再登陆也不会发送离线消息了。所以这个功能还是等以后加入数据库后在添加了。

5.5 聊天记录的储存

聊天记录的储存也有两种选择，一种储存在本地，一种储存在服务器数据库中，储存在本地可以选择HTML5的localStoreage API，通过登陆用户名作为键，登陆用户的聊天记录作为值，储存在localStorage中。

5.6 加入服务器数据库

增加注册功能，这样就可以为每一个用户分发唯一的id值，用户登陆token验证等啊，数据库也可以用来储存用户信息，用户聊天记录，离线消息发送等。