# Mint UI - Unit01

# 1.组件库基础

## 1.1 什么组件?

组件(Component)是对复用的结构<template>、表现(<style>)及行为<script>的封装，其优势在于一次定义，多次使用。

组件库是由多个组件形成的完整的架构体系。

## 1.2 组件库的分类

根据组件库应用平台的不同，可以分为：

* 移动端组件库
  + Mint UI(饿了吗) -- http://mint-ui.github.io/#!/zh-cn
  + Vant UI(有赞) -- https://vant-contrib.gitee.io/vant/#/zh-CN/
  + Cube UI(滴滴) -- https://didi.github.io/cube-ui/#/zh-CN
* 桌面端组件库
  + Element UI(饿了吗) -- https://element.eleme.cn/#/zh-CN
  + AT-UI(凹凸实验室) -- https://at-ui.github.io/at-ui/
  + View UI(视图更新) -- https://iviewui.com/

UI(User Interface)，用户接口

# 2. Mint UI基础

Mint UI是基于Vue.js 的移动端组件库。

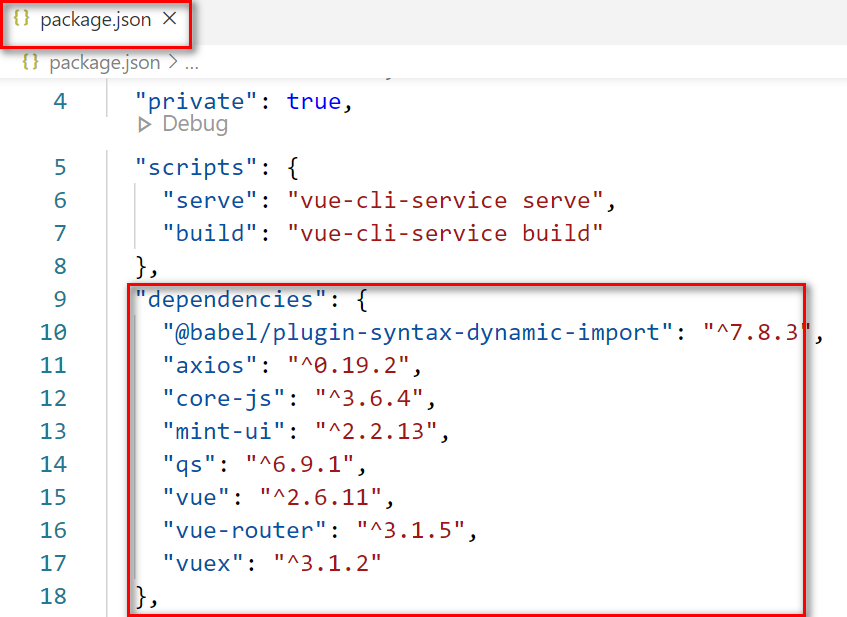
## 2.1 安装

A.启动Windows操作系统的命令行，并且切换到脚手架目录

B.输入以下命令

npm install --save mint-ui

当前脚手架安装过哪些的模块，可以在脚手架根目录下的package.json中进行查看



## 2.2 配置Mint UI

配置Mint UI需要在脚手架根目录下的src/main.js中实现

A.在脚手架中导入所有的Mint UI 组件

import MintUI from 'mint-ui';

B.导入样式表文件

import 'mint-ui/lib/style.min.css';

C.通过Vue.use()方法将Mint UI注册为Vue的插件

Vue.use(MintUI);

# 3.Mint UI组件库的使用

## · Header 组件

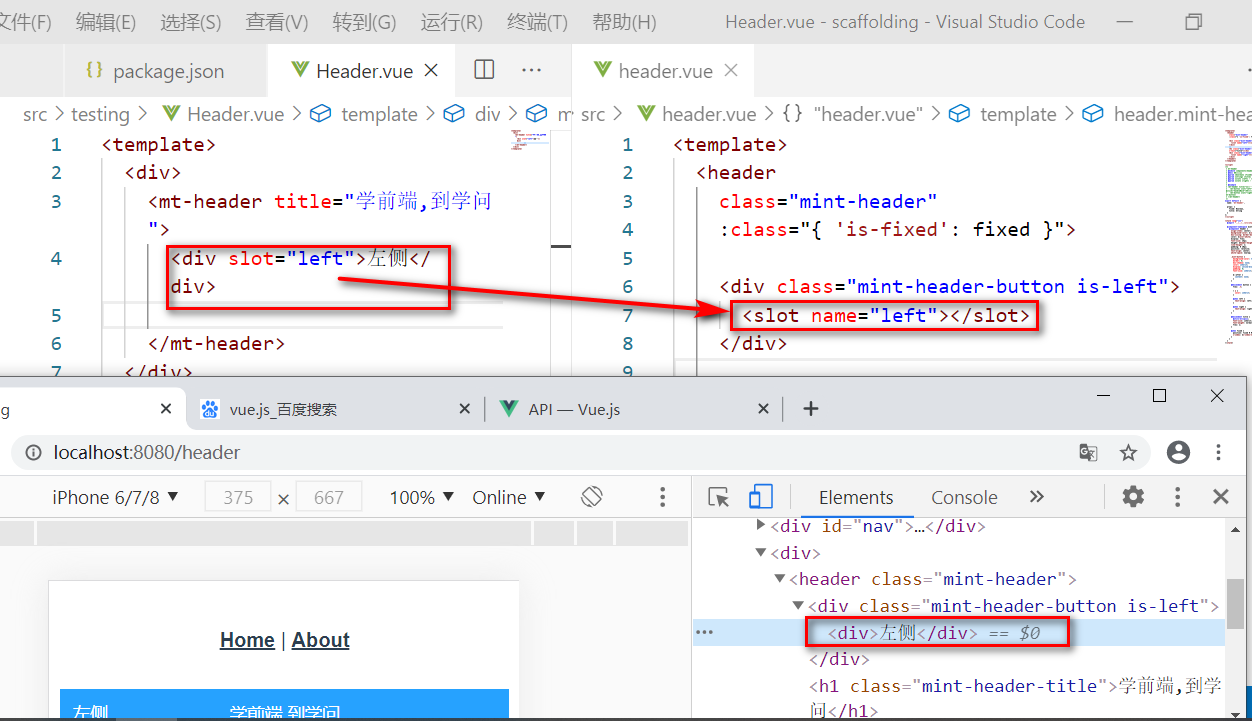
Header组件用于实现顶部导航，其语法结构是：

<mt-header title="导航文本" fixed>   
</mt-header>

fixed 属性用于控制是否固定在顶部，布尔属性

在<mt-header>中可以嵌套子元素，此时必须为子元素添加slot="left"或slot="right"属性

slot称为插槽，是预留给使用者自定义内容的区域。



## · Button 组件

Button组件为按钮组件，其语法结构是：

<mt-button type="按钮类型" size="尺寸" icon="图标" disabled plain>   
</mt-button>

按钮类型(颜色)有：default(默认)、primary(主要的—蓝色)、danger(危险的—红色)

按钮的尺寸有：small(小的)、normal(标准的)、large(大的)

图标类型有：back(返回-- < )、more(更多-- …)

disabled属性用于控制按钮是否为禁用状态，布尔属性

plain属性用于控制按钮是否为镂空按钮，布尔属性

## · Field 组件

Field组件用于实现表单编辑器，其语法结构是：

<mt-field   
 type="输入框的类型"   
 label="标签"   
 placeholder="占位提示内容"   
 state="状态"   
 :attr="{...}"   
 disableClear   
 readonly   
 disabled>   
   
</mt-field>

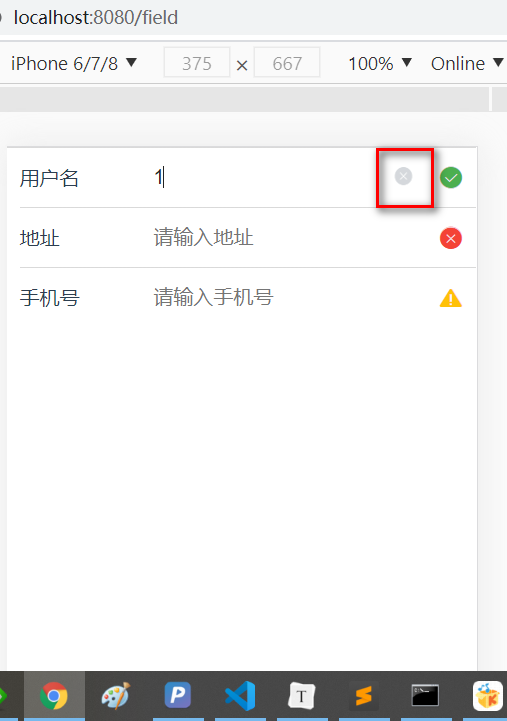
输入框的类型有：text、password、textarea、email、url等

状态包括：success(成功)、error(错误)、warning(警告)

:attr属性用于使用HTML5表单组件的原生属性，object类型，如

<mt-field   
 type="password"   
 label="密码"   
 placeholder="请输入密码"   
 :attr="{maxlength:10,autocomplete:'off'}"   
 state="error">   
</mt-field>

disableClear属性用于控制是否显示清理按钮，布尔类型



readonly属性用于控制是否只读，布尔类型

disabled属性用于控制是否禁用，布尔类型

**表单实质上就是数据表结构的映射**

**新增型的表单中(如用户注册、添加配送地址等)输入框的默认值可以为空**

**修改型的表单中(如编辑个人注册信息，编辑配送地址等)输入框"必须"存在默认值**

新增型的表单映射为SQL中的INSERT操作，修改型的表单映射为SQL中的UPDATE操作

## · Toast 组件

Toast组件用于显示短消息提示框，其语法结构是：

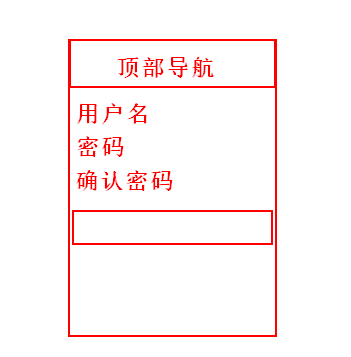
//简捷语法   
this.$toast("提示内容")   
//标准语法   
this.$toast({   
 message:"提示内容",   
 position:"位置(top|middle|bottom)",   
 duration:持续时长(3000ms) -- 数值类型   
})

示例代码如下：

methods:{   
 handle(){   
 this.$toast({   
 message:"错误提示",   
 duration:5000   
 })   
 }   
}

周未小礼物：

1.创建src/views/Register.vue以实现用户的注册业务，页面结构如下：



2.让用户名、密码及确认密码失去焦点后即可完成对自己的校验，同时改变状态(state)

友情提示:事件修饰符

3.可以再实现src/views/Login.vue以实现登录

# Mint UI - Unit02

# 1.Field组件获取/失去焦点

在Mint UI中Field组件获取/失去焦点的写法是：

<mt-field type="text" @blur.native.capture="函数名称"></mt-field>  
  
<mt-field type="text" @focus.native.capture="函数名称"></mt-field>

Vue.js中的事件修饰符有：

stop，等价于event.stopPropagation()

prevent，等价于event.preventDefault()

capture 是事件冒泡的捕获阶段

# 2.组件

# · Navbar组件

Navbar组件用于实现顶部选项卡，其语法结构是：

<mt-navbar fixed v-model="变量名称">  
 <mt-tab-item id="当前选卡的ID">...</mt-tab-item>  
 ...  
</mt-navbar>

Navbar组件绑定的变量值应该是<mt-tab-item>的id

可以在<mt-tab-item>内嵌套图像，如果为图像添加slot="icon"属性的话，图像将成为当前选项卡的图标。

<mt-tab-item> 的id 属性只需要保证在当前<mt-navbar>内唯一即可。

## · TabContainer组件

TabContainer组件用于实现面板，其语法结构是：

<mt-tab-container v-model="变量名称">  
 <mt-tab-container-item id="当前面板的ID">...</mt-tab-container-item>  
 ...  
</mt-tab-container>

TabContainer组件绑定的变量值应该是<mt-tab-container-item>的id

<mt-tab-container-item> 的id 属性只需要保证在当前<mt-tab-container>内唯一即可。

## · Tabbar组件

Tabbar组件用于实现底部选项卡，其语法结构是：

<mt-tabbar fixed v-model="变量名称">  
 <mt-tab-item id="当前选项卡的ID">...</mt-tab-item>  
 ...  
</mt-tabbar>

Tabbar组件绑定的变量值应该是<mt-tab-item>的id

<mt-tab-item> 的id 属性只需要保证在当前<mt-tabbar>内唯一即可。

可以在<mt-tab-item>内嵌套图像，如果为图像添加slot="icon"属性的话，图像将成为当前选项卡的图标。

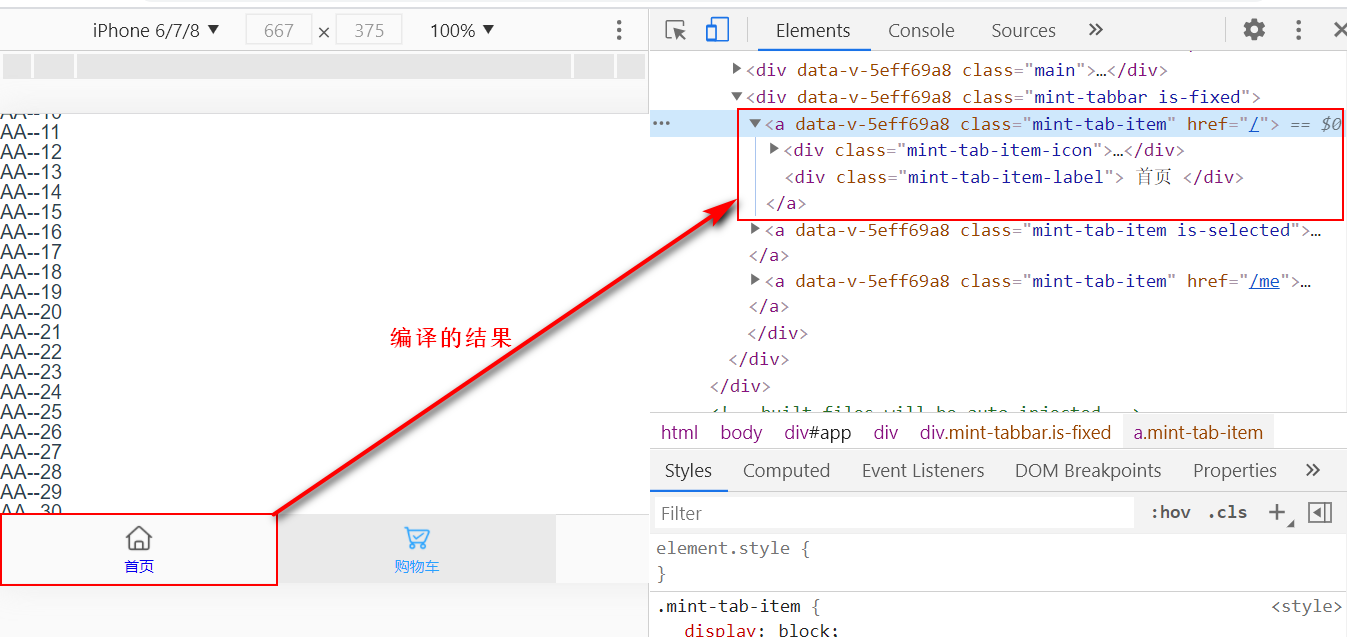
一般情况下<mt-tabbar>组件需要使用fixed属性，以固定在页面的底部；如果将其固定的话，必须为同级的兄弟元素设置样式，否则造成内容的覆盖。示例代码如下：

<template>  
 <div>  
 <div class="main">  
 ...  
 </div>  
 <mt-tabbar fixed>  
 ...  
 </mt-tabbar>   
 </div>  
</template>  
  
<style scoped>  
 .main{  
 margin-bottom:55px;  
 }  
</style>

<mt-tabbar>如果要实现链接到不同页面效果的话，其实现方法有两种：

A.<mt-tab-item>组件在编译后形成的了<a>标签，所以可以直接添加href属性来实现链接。示例代码如下：

<mt-tabbar>  
 <mt-tab-item href="/cart">购物车</mt-tab-item>  
 <mt-tab-item href="/me">我的</mt-tab-item>  
</mt-tabbar>



B.通过监听<mt-tabbar>绑定的变量，然后再使用this.$router.push()方法来实现跳转，示例代码如下：

watch(){  
 selected(newValue){  
 if(newValue == 'index'){  
 this.$router.push('/');  
 } else if(newValue == 'cart'){  
 this.$router.push('/cart');  
 } else if(newValue == 'me'){  
 this.$router.push('/me');  
 }  
 }  
}

## · Swipe组件

Swipe组件用于实现轮播图，其语法结构是：

<mt-swipe :auto="时长" :show-indicators="是否显示指示标志">   
 <mt-swipe-item>...</mt-swipe-item>   
 ...   
</mt-swipe>

因为<mt-swipe-item>采用相对的宽度和高度的样式，即:

.mint-swipe-items-wrap>div {  
 width:100%;  
 height:100%;  
}

所以在实践时，必须为swipe元素设置明确的高度

复习以下知识：

1.Node.js中的相关知识 -- Express、CORS、MySQL

2.复习MySQL数据库的相关知识 -- SELECT操作

3.必须保证XAMPP中的MySQL可以正常启动

# Mint UI - Unit02

# 1.Field组件获取/失去焦点

在Mint UI中Field组件获取/失去焦点的写法是：

<mt-field type="text" @blur.native.capture="函数名称"></mt-field>  
  
<mt-field type="text" @focus.native.capture="函数名称"></mt-field>

Vue.js中的事件修饰符有：

stop，等价于event.stopPropagation()

prevent，等价于event.preventDefault()

capture 是事件冒泡的捕获阶段

# 2.组件

# · Navbar组件

Navbar组件用于实现顶部选项卡，其语法结构是：

<mt-navbar fixed v-model="变量名称">  
 <mt-tab-item id="当前选卡的ID">...</mt-tab-item>  
 ...  
</mt-navbar>

Navbar组件绑定的变量值应该是<mt-tab-item>的id

可以在<mt-tab-item>内嵌套图像，如果为图像添加slot="icon"属性的话，图像将成为当前选项卡的图标。

<mt-tab-item> 的id 属性只需要保证在当前<mt-navbar>内唯一即可。

## · TabContainer组件

TabContainer组件用于实现面板，其语法结构是：

<mt-tab-container v-model="变量名称">  
 <mt-tab-container-item id="当前面板的ID">...</mt-tab-container-item>  
 ...  
</mt-tab-container>

TabContainer组件绑定的变量值应该是<mt-tab-container-item>的id

<mt-tab-container-item> 的id 属性只需要保证在当前<mt-tab-container>内唯一即可。

## · Tabbar组件

Tabbar组件用于实现底部选项卡，其语法结构是：

<mt-tabbar fixed v-model="变量名称">  
 <mt-tab-item id="当前选项卡的ID">...</mt-tab-item>  
 ...  
</mt-tabbar>

Tabbar组件绑定的变量值应该是<mt-tab-item>的id

<mt-tab-item> 的id 属性只需要保证在当前<mt-tabbar>内唯一即可。

可以在<mt-tab-item>内嵌套图像，如果为图像添加slot="icon"属性的话，图像将成为当前选项卡的图标。

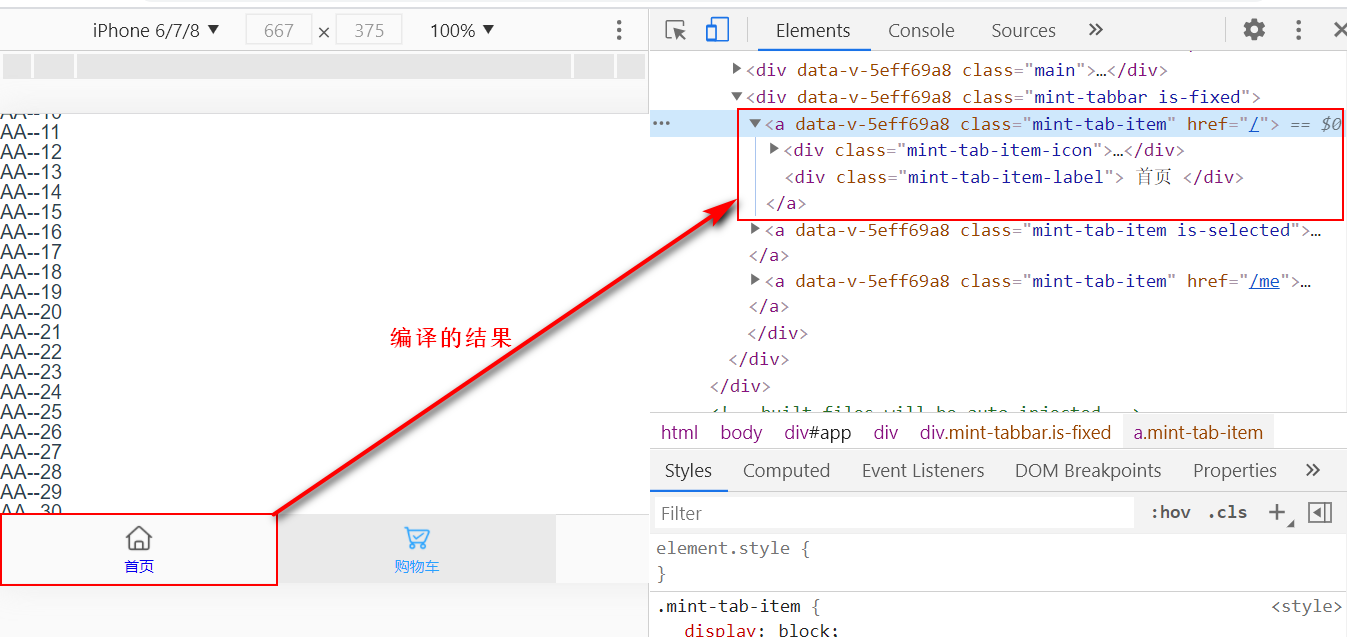
一般情况下<mt-tabbar>组件需要使用fixed属性，以固定在页面的底部；如果将其固定的话，必须为同级的兄弟元素设置样式，否则造成内容的覆盖。示例代码如下：

<template>  
 <div>  
 <div class="main">  
 ...  
 </div>  
 <mt-tabbar fixed>  
 ...  
 </mt-tabbar>   
 </div>  
</template>  
  
<style scoped>  
 .main{  
 margin-bottom:55px;  
 }  
</style>

<mt-tabbar>如果要实现链接到不同页面效果的话，其实现方法有两种：

A.<mt-tab-item>组件在编译后形成的了<a>标签，所以可以直接添加href属性来实现链接。示例代码如下：

<mt-tabbar>  
 <mt-tab-item href="/cart">购物车</mt-tab-item>  
 <mt-tab-item href="/me">我的</mt-tab-item>  
</mt-tabbar>



B.通过监听<mt-tabbar>绑定的变量，然后再使用this.$router.push()方法来实现跳转，示例代码如下：

watch(){  
 selected(newValue){  
 if(newValue == 'index'){  
 this.$router.push('/');  
 } else if(newValue == 'cart'){  
 this.$router.push('/cart');  
 } else if(newValue == 'me'){  
 this.$router.push('/me');  
 }  
 }  
}

## · Swipe组件

Swipe组件用于实现轮播图，其语法结构是：

<mt-swipe :auto="时长" :show-indicators="是否显示指示标志">   
 <mt-swipe-item>...</mt-swipe-item>   
 ...   
</mt-swipe>

因为<mt-swipe-item>采用相对的宽度和高度的样式，即:

.mint-swipe-items-wrap>div {  
 width:100%;  
 height:100%;  
}

所以在实践时，必须为swipe元素设置明确的高度

复习以下知识：

1.Node.js中的相关知识 -- Express、CORS、MySQL

2.复习MySQL数据库的相关知识 -- SELECT操作

3.必须保证XAMPP中的MySQL可以正常启动

# Mint UI - Unit03

# 1.学子问答首页实践

## 1.1 顶部选项卡的实现



### 1.1.1 数据库的基本信息

· 数据库名称：xzqa

· 数据库编码方式：utf8

### 1.1.2 xzqa\_category表结构

xzqa\_category表是文章分类表，其结构如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 是否为空 | 键 | 扩展 | 说明 |
| id | SMALLINT UNSIGNED | NO | PRIMARY KEY | AUTO\_INCREMENT | 分类ID,主键且自增 |
| category\_name | VARCHAR(30) | NO | UNIQUE KEY |  | 分类名称，唯一约束 |

### 1.1.3 xzqa\_author表结构

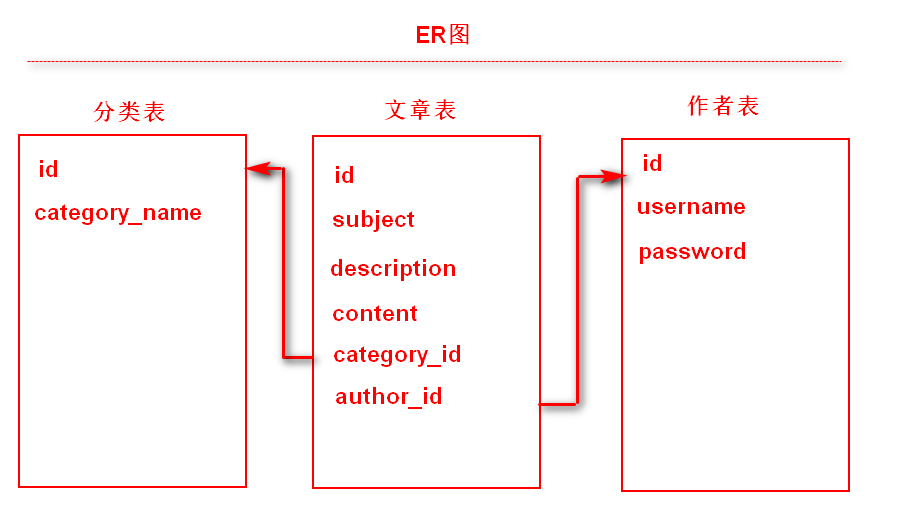
xzqa\_author是文章的作者表，其结构如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 是否为空 | 键 | 扩展 | 说明 |
| id | MEDIUMINT UNSIGNED | NO | PRIMARY KEY | AUTO\_INCREMENT | 作者ID主键且自增 |
| username | VARCHAR(30) | NO | UNIQUE KEY |  | 用户名，唯一约束 |
| password | VARCHAR(32) | NO |  |  | 密码，MD5加密 |
| nickname | VARCHAR(30) | YES |  |  | 用户昵称 |
| avatar | VARCHAR(50) | NO |  | DEFAULT 'uname.jpg' | 用户头像 |
| article\_number | MEDIUMINT UNSIGNED | NO |  | DEFAULT 0 | 用户发表的文章数量 |

### 1.1.4 xzqa\_article表结构

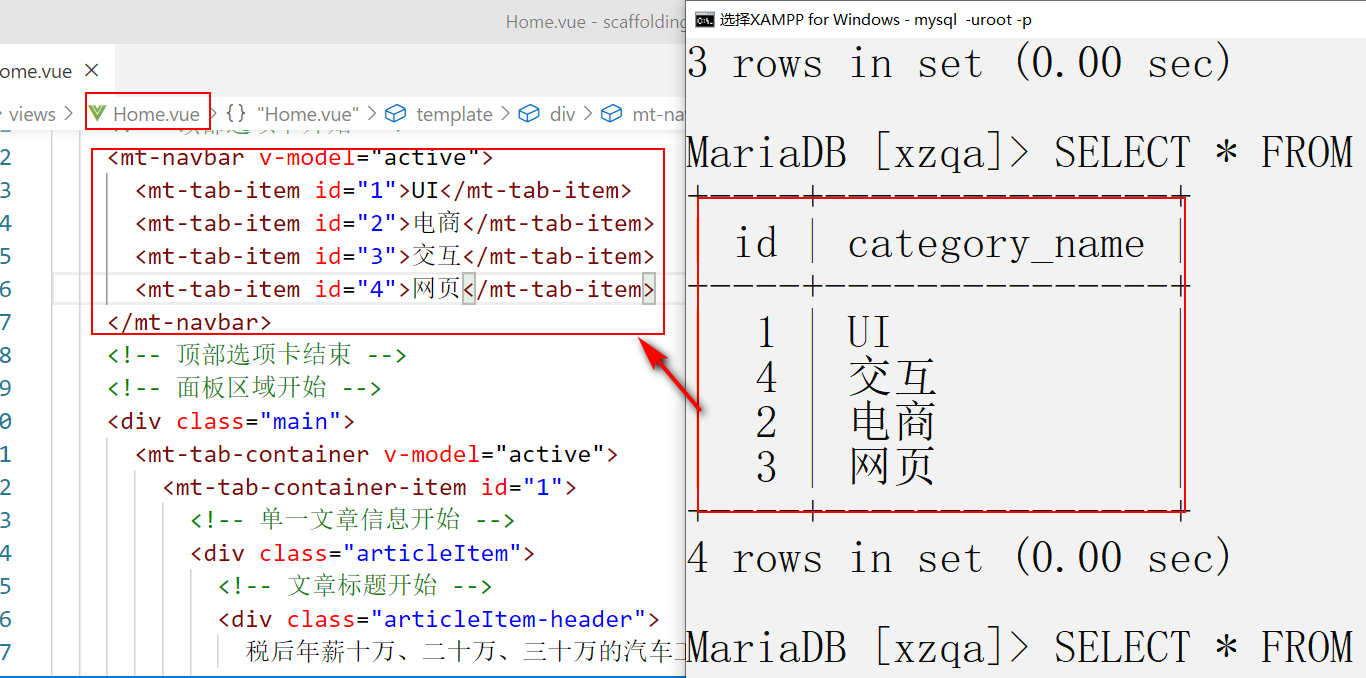
xzqa\_article表是文章表，其结构如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 是否为空 | 键 | 扩展 | 说明 |
| id | INT UNSIGNED | NO | PRIMARY KEY | AUTO\_INCREMENT | 文章ID，主键且自增 |
| subject | VARCHAR(50) | NO |  |  | 文章标题 |
| description | VARCHAR(255) | NO |  |  | 文章简介 |
| image | VARCHAR(50) | YES |  |  | 文章缩略图(重要！) |
| content | MEDIUMTEXT | NO |  |  | 文章正文 |
| category\_id | SMALLINT UNSIGNED | NO |  |  | 外键，参照分类表的ID字段 |
| author\_id | INT UNSIGNED | NO |  |  | 外键，参照作者表的ID字段 |
| created\_at | INT UNSIGNED | NO |  |  | 文章发表的日期 |



### 1.1.5 顶部选项卡的实现

原来在Home.vue中的顶部选项卡的数据是固定的，而现在已有了数据库及数据表，并且也存在了相应的记录，所以Home.vue中的顶部选项卡的数据应该从数据库中进行获取操作。在数据库中存在xzqa\_category数据表，该数据表用于存储文章的分类，所以现在需要将访表中的数据获取到后显示在Home.vue中即可。



既然现在要获取数据库的数据，那么也就代表脚手架需要向WEB服务器发送请求后才能获取数据；既然要发送请求，那么必须要通过发送请求的工具来实现，请问工具是谁呢？ -- axios

既然需要axios，那么肯定需要进行安装并且配置，所以：

第一步：脚手架上安装axios

npm install --save axios

第二步：配置axios

在src/main.js中完成

//导入axios模块  
import axios from 'axios';  
  
//设置axios基础的URL地址(在此已经决定了WEB服务器的端口号)  
axios.defaults.baseURL = 'http://127.0.0.1:3000'  
  
//将axios扩展成为Vue的属性  
Vue.prototype.axios = axios;

Apache服务器的默认端口为80

MySQL数据库服务器的默认端口号为3306

FTP 服务器的端口号默认为21

HTTPS协议默认的端口号是：443

现在已经成功安装并且配置好Axios，此时就可以通过该工具向WEB服务器发送异步的AJAX请求了，但是现在在问题是：脚手架在什么情况下发送请求呢或者说请求的代码要写在Home.vue中什么位置呢？-- 此时需要回顾Vue的生命周期的回调函数

beforeCreate、created、beforeMount、mounted、beforeUpdate、updated、beforeDestroy、destroyed

现在需要在Home.vue中调用mounted生命周期的回调函数并且发送异步的axios请求，示例代码如下：

mounted(){  
 //通过axios工具向Web服务器发送请求以获取数据   
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*坑\*\*\*\*\*坑\*\*\*\*\*\*坑\*\*\*\*\*\*\*\*\*\*  
 //以如下的结构中，发现不存在WEB服务器URL地址的话，整个请求过程根本无法进行  
 //所以跳转到1.1.6先完成WEB服务器的搭建并创建接口，然后再发送请求  
 //this.axios.get(URL地址,参数).then(res=>{  
   
 //});  
}

针对RESTFul的API，其请求方式有以下几种：

GET，主要用于从服务器获取数据

POST，主要用于向服务器添加数据

PUT/PATCH，主要用于修改服务器数据

DELETE，主要用于删除服务器数据

### 1.1.6 WEB服务器的搭建

现在需要搭建WEB服务器，所以需要创建Node.js的服务器环境，故安装并配置 Express框架

第一步：安装Express

npm install --save express

第二步：配置Express

A.在server的根目录下创建入口文件 -- index.js

B.在index.js中完成WEB服务器的创建，示例代码如下：

// 加载Express模块  
const express = require('express');  
// 创建WEB服务器实例  
const server = express();  
// 指定WEB服务器监听的端口  
server.listen(3000);

C.在Node.js环境下的Web服务器中创建接口（因为任何HTTP请求都必须请求相对应的接口地址才行，所以必须创建接口）, 接口的规定如下：

请求方式：GET

参数：无

请求地址：http://127.0.0.1:3000/category

此时必须在WEB服务器中创建名称为category的GET类型的接口才行，代码如下：

server.get('/category',(req,res)=>{  
 res.send('ok');  
});

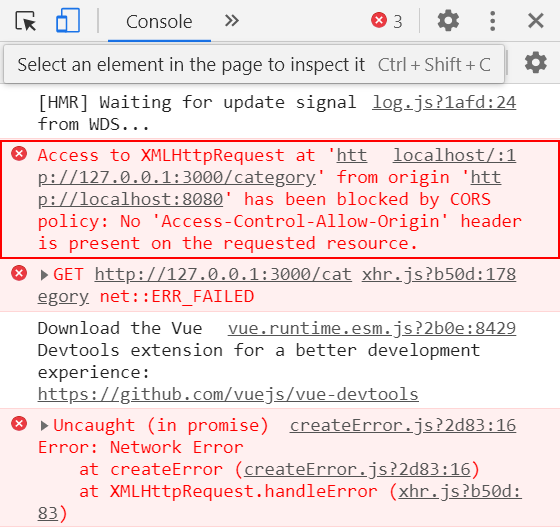
D.ctrl+反引号打开终端，输入以下命令启动WEB服务器

node index.js

如果服务器正常启动的话，那么代表的原来的"坑"已经填完了，现在可以在Home.vue中mounted生命周期回调函数中发送正常的HTTP请求了(因为现在服务器有了，接口也有了)！此时的代码如下：

mounted(){  
 //通过axios工具向Web服务器发送请求以获取数据   
 this.axios.get('/category').then(res=>{  
  
 });  
}

此时客户端的运行结果如下图所示：



现在的错误原因是：跨域错误（因为脚手架的地址是 http://127.0.0.1:8080，而WEB服务器的地址是http://127.0.0.1:3000，端口号不同，会造成跨域）

### 1.1.7 跨域问题的解决 -- CORS

跨域问题的解决有多种，如CORS、JSONP、HTTP-Proxy等

在Node.js环境中安装并配置CORS模块：

第一步：安装

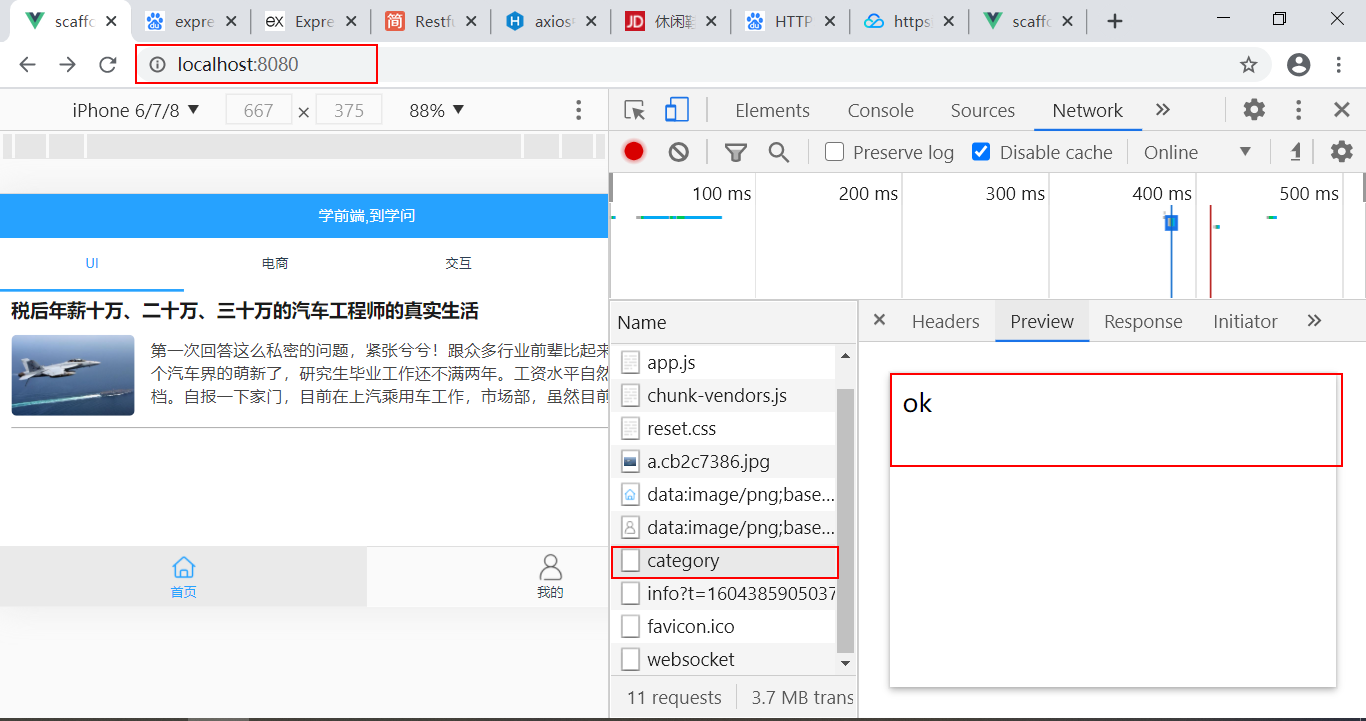
npm install --save cors

第二步：配置

//加载CORS模式  
const cors = require('cors');  
//将CORS作为服务器的中间件  
server.use(cors({  
 origin:['http://127.0.0.1:8080','http://localhost:8080']  
}));

第三步：重新启动Node.js服务器

此时脚手架的运行结果如下图所示：



到目前为目，整个应用的网络环境已经成功了，但是依然没有获取出数据库的数据！所以：

### 1.1.8 WEB服务器操作数据库的数据

既然要在WEB服务器上操作数据库的数据，所以必须在Node.js环境中安装MySQL模块才能够操作数据库中的数据：

第一步：安装

npm install --save mysql

第二步：配置

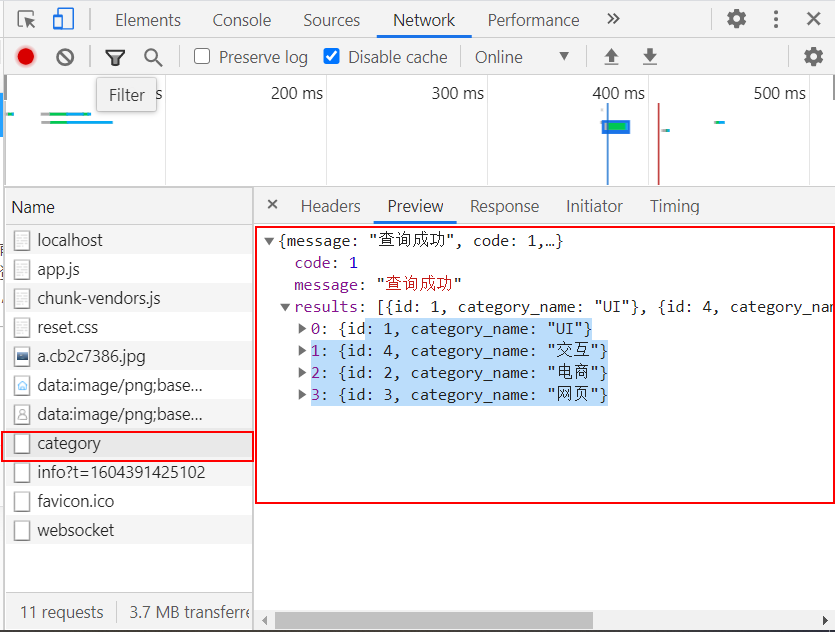
// 加载MySQL模块  
const mysql = require('mysql');  
  
// 创建MySQL连接池  
const pool = mysql.createPool({  
 // 数据库服务器的地址  
 host:'127.0.0.1',  
 // 数据库服务器的端口号  
 port:3306,  
 // 数据库用户的用户名  
 user:'root',  
 // 数据库用户的密码  
 password:'',  
 // 数据库名称  
 database:'xzqa',  
 // 编码方式  
 charset:'utf8',  
 // 最大连接数  
 connectionLimit:20  
});

第三步：修改接口 -- /category

server.get('/category',(req,res)=>{  
 //获取文章分类表中的全部数据  
 let sql = 'SELECT id,category\_name FROM xzqa\_category';  
 //通过连接池的query()方法来执行SQL语句  
 pool.query(sql,(error,results)=>{  
 if(error) throw error;  
 res.send({message:'查询成功',code:1,results:results});  
 });   
});

第四步：重新启动Node.js服务器

此时客户端的运行结果如下图所示：



此时已经证明客户端成功的向WEB服务器发送了请求，而且WEB服务器也已经将数据成功的返回到了客户端，但是目前在客户端中没有接收数据，当然也没有显示在页面中，所以：

### 1.1.9 客户端接收数据并且呈现在页面中

因为整个的请求过程是由Home.vue中的mounted发起的，所以需要修改mounted的生命周期代码，以完成接收数据的操作，具体操作过程如下：

A.需要首先在Home.vue 的data属性中声明变量，如category，因为只有将服务器返回的结果存储到变量中，才可以在页面中通过v-for指令进行循环输出，示例代码如下：

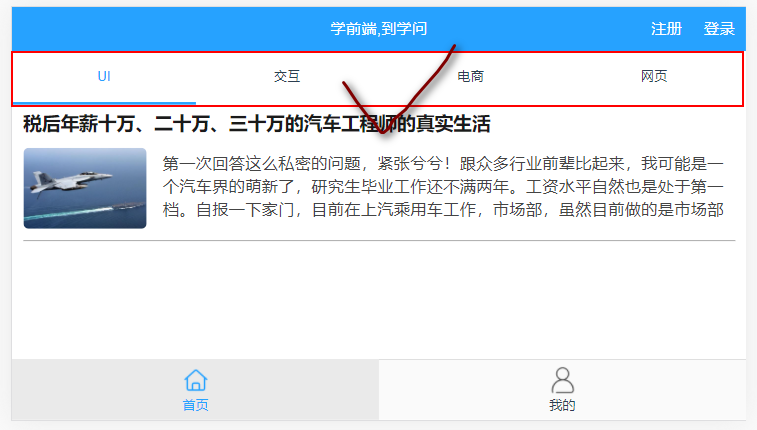
data(){  
 return {  
 //用于存储服务器返回的文章分类数据  
 category:[]  
 }  
}

B.现在已经声明了变量，于是就可以在mounted的时候把服务器返回的结果存储到该变量中了，代码如下：

mounted(){  
 //通过axios工具向Web服务器发送请求以获取数据   
 this.axios.get('/category').then(res=>{  
 this.category = res.data.results;  
 });  
}

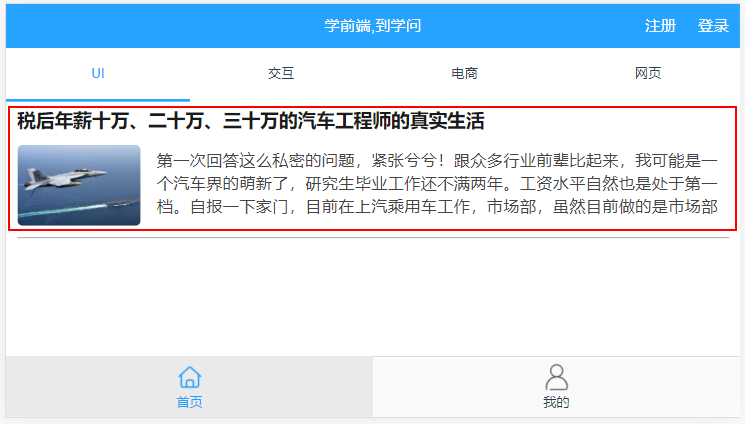
C.现在已经将服务器返回的数据存储到category变量中了，于是就可以在页面中通过v-for指令来循环输出，以决定顶部选项卡的数量及内容了，代码如下：

<!-- 顶部选项卡开始 -->  
<mt-navbar v-model="active">  
 <mt-tab-item   
 v-for="(item,index) of category"   
 :key="index"  
 :id="item.id.toString()">  
 {{item.category\_name}}   
 </mt-tab-item>  
</mt-navbar>  
<!-- 顶部选项卡结束 -->



## 1.2 首页内容的实现

### 1.2.1 首页初始化的文章列表的实现



在数据表xzqa\_article中，记录了文章数据，而且每一篇文章一定是隶属于某一个分类的。在初始的情况下，应访获取默认分类下包含的文章数据。涉及的问题有：

A.在什么情况下要获取默认分类下包含的文章数据? -- mounted

B.请问要向哪个WEB服务器接口发送请求？发送什么样的请求？是否带有参数？

## 1.2.2 在初始情况下发送请求以获取文章数据

在初始情况下在Vue.js中只能通过生命周期的回调函数实现，此时可通过mounted这一生命周期的回调函数来继续发送请求，以获取文章数据，伪代码如下：

mounted(){   
 //通过axios工具向Web服务器发送请求以获取文章数据   
 //目前不存在相关的WEB服务器的URL地址，所以代码称为伪代码   
 //如果想要真正去运行，必须保证存在web服务器且存在相关的API才行   
 //在经过获取顶部选项卡操作时(1)已经成功创建了WEB服务器(2)解决了跨域问题   
 //于是只剩下一个问题：在WEB服器下创建API接口才行   
 //this.axios.get('URL地址','参数').then(res=>{   
   
 //});   
}

于是需要修改web服务器中的index.js，为其添加API接口，接口规定如下：

请求方式：GET

参数：cid，表示的是要获取哪一类的文章数据，因为在初始情况下要只获取默认分类下的文章数据，而在后续的切换顶部选项卡时，要获取相关分类的文章数据，所以最好用一个接口来完成多项业务，故需要提供cid参数。

请求地址：http://127.0.0.1:3000/articles?cid=5

初始化的情况下可以写死:   
   
SELECT \* FROM xzqa\_article WHERE category\_id=1;   
   
在切换顶部选项卡时，仍然要获取对应分类下包含的文章数据,   
此时的SQL的结构如下：   
   
SELECT \* FROM xzqa\_article WHERE category\_id = 你刚刚选择的那个分类的ID   
   
在如上的结构是，发现只有分类的ID是不确定的，所以可以通过URL地址栏参数的形式将   
要获取的某一类的ID进行传递，在初始情况下只需要传1即可，在切换顶部选项卡的时候需要传递   
刚刚选择的那个分类的ID就行了

作业：

A.顶部选项卡重构

B.实现首页中读取初始分类的文章数据的操作

# Mint UI - Unit04

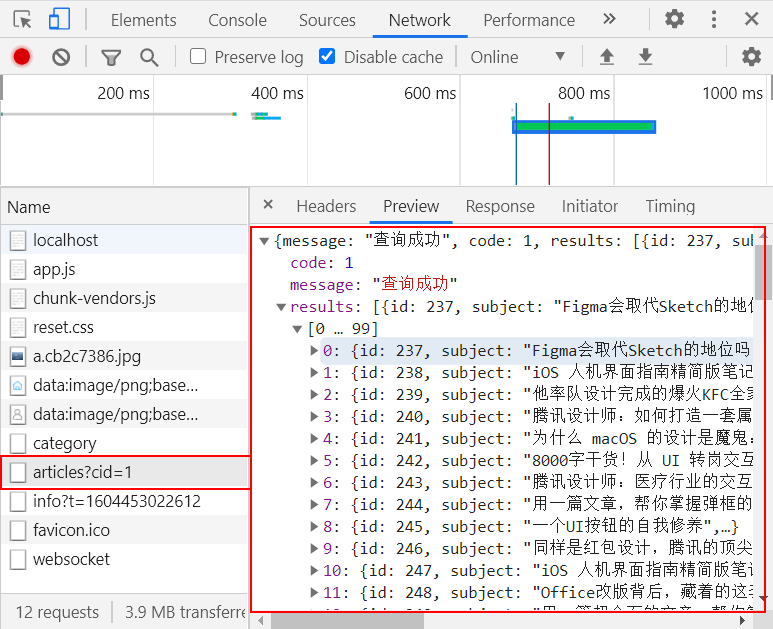
# 1.

## 1.1 在初始情况下发送请求以获取文章数据

此时已经在发送请求的时候带有了URL地址栏的参数，同时也已经在服务器接收到了该参数，所以需要在服务器上按照该参数进行文章查找即可。所以服务器的/articles接口的代码如下：

// 获取指定分类下包含的文章数据   
server.get('/articles',(req,res)=>{   
 // 获取地址栏的cid参数,该参数表示的分类的ID   
 let cid = req.query.cid;   
 // 以当前的cid为条件进行文章的查找操作   
 let sql = 'SELECT id,subject,description,image FROM xzqa\_article WHERE category\_id=?';   
 // 执行SQL查询   
 pool.query(sql,[cid],(error,results)=>{   
 if(error) throw error;   
 res.send({message:'查询成功',code:1,results:results});   
 });   
});

此时客户端的运行结果如下图所示：



此时代表服务器已经成功的返回了客户端期望的结果，但是目前还没有在页面中输出，如果要输出的话，必须将服务器返回的结果保存到变量中，然后再通过v-for指令进行输出，所以先需要在data中声明一个变量用于存储服务器返回结果，最后再通过v-for指令进行输出。此时示例代码如下：

A.在data中声明一个变量

data(){   
 return {   
 //用于存储服务器返回结果   
 articles:[]   
 }   
}

B.在客户端接收到服务器的返回结果后，将返回的结果存储到变量中去

this.axios.get('/articles?cid=' + this.active).then(res=>{   
 this.articles = res.data.results;   
});

C.在页面通过v-for指令输出

<!-- 单一文章信息开始 -->   
<div   
 class="articleItem"   
 v-for="(article,index) of articles"   
 :key="index">   
 <!-- 文章标题开始 -->   
 <div class="articleItem-header">   
 {{article.subject}}   
 </div>   
 <!-- 文章标题结束 -->   
 <!-- 文章图文信息开始 -->   
 <div class="articleItem-wrapper">   
 <!-- 文章图像开始 -->   
 <div class="articleImg">   
 <img src="../assets/images/articles/a.jpg" />   
 </div>   
 <!-- 文章图像结束 -->   
 <!-- 文章简介开始 -->   
 <div class="articleDes">   
 {{article.description}}   
 </div>   
 <!-- 文章简介结束 -->   
 </div>   
 <!-- 文章图文信息结束 -->   
</div>   
<!-- 单一文章信息结束 -->

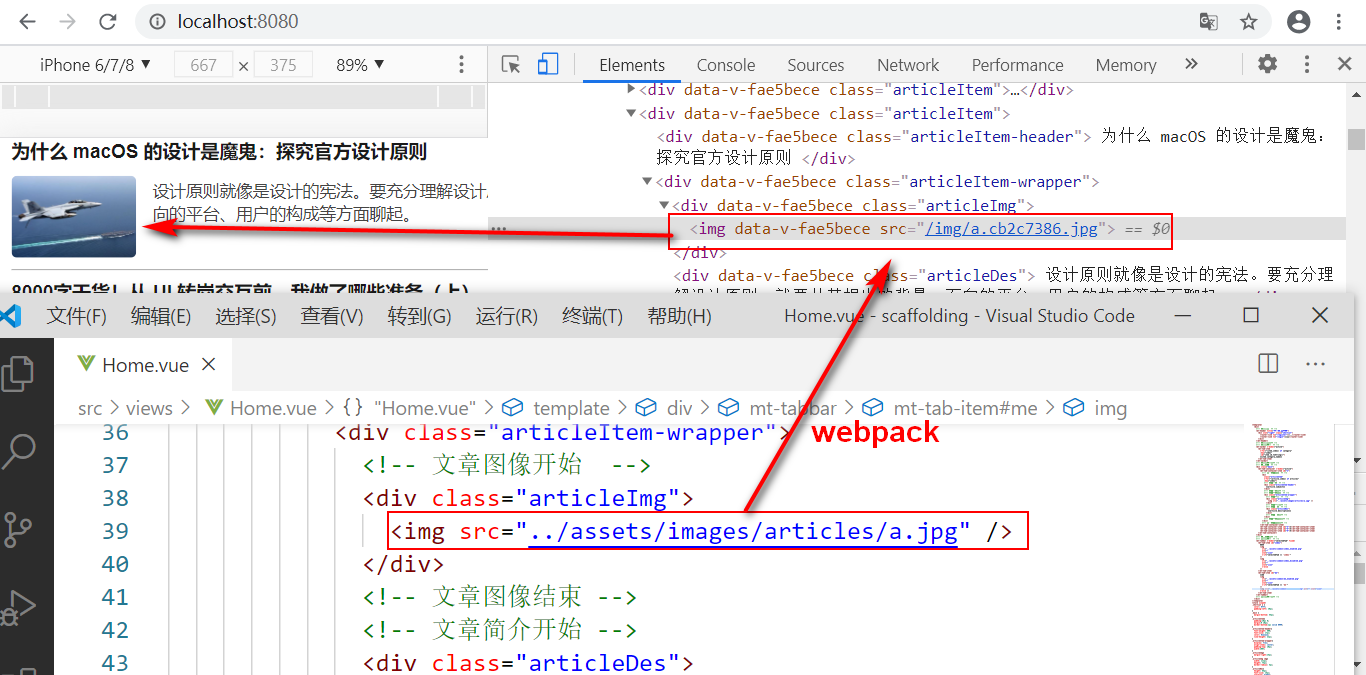
此时客户端的运行结果如下图所示：



目前关键的问题是来解决文章缩略图的问题，关于文章缩略图需要明确：

A.文章缩略图在数据表中是可以为空的

B.在输出时图片的名称是不固定的，那么就代表必须采用 :src的方式来进行输出



对于Vue.js中动态图片的处理必须要使用require语句进行加载操作

Base64是一种二进制到文本的编码方式，用ASCII方式进行方式。

采用Base64编码的最大原因是可以减少HTTP请求次数，减轻服务器的请求压力

## 1.2 切换顶部选项卡时数据发生变化

在切换顶部选项卡时页面中的数据要随之发生相应的变化，此时需要确定：

A.只有在切换顶部选项卡时数据才发生变化 -- 怎么证明顶部选项卡发生了变化呢? -- 依靠watch

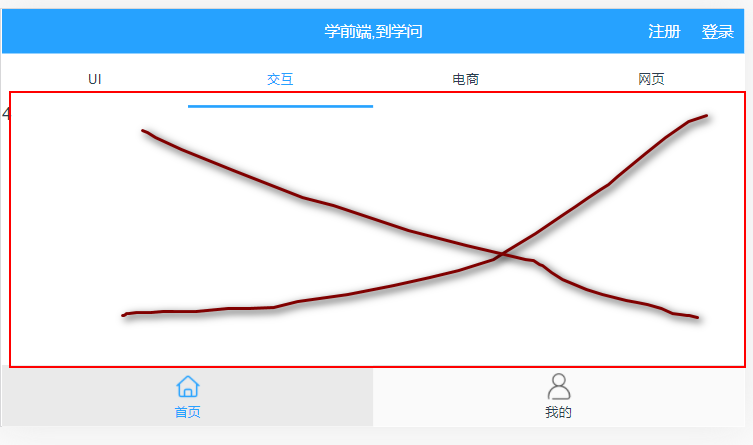
watch:{   
 active(value){   
 console.log(value);   
 }   
}

B.在切换顶部选项卡时依然要向服务器发送请求以获取数据，此时代表在该过程中仍然要向服务器发送请求，以获取该分类下包含的文章数据，在前面的过程中已经实现了获取默认分类下包含的文章数据的业务，现在要获取的是指定分类下包含的文章数据，它们只有分类的ID是不同的，其它的过程是完全一致的，所以可以使用前期的/articles接口来实现访业务功能。

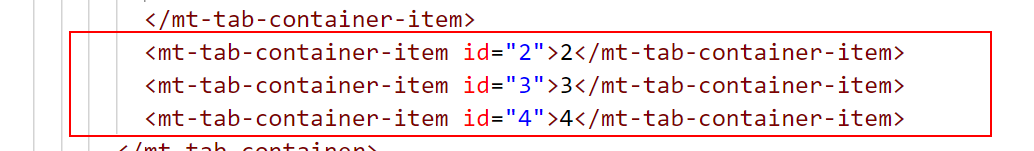
所以watch的代码修改如下：

this.axios.get('/articles?cid=' + value).then(res=>{   
 this.articles = res.data.results;   
});

此时需要在切换顶部选项卡时数据随之发生变化，但从效果看是不存在数据的。



其根本原因是因为：顶部选项卡与面板都绑定了同一个变量active，所以在切换顶部选项卡时，面板也随之发生了变化，而原来只有在第一个面板中进行数据的输出操作，故切换到第一个选项卡外都不存在数据！



**仔细试想一下，无论顶部选项卡是谁，都要显示面板，只要面板的数据发生了变化，用户感觉起来就是面板也进行了切换。**

所以最终只需要保留一个面板即可，而且不需进行双向的数据绑定，也就代表其永远都显示！

故面板的最终代码如下图所示：



但是，现在切换后的效果如下图所示：



初始情况下图片是可以正常显示的，因为进行了图片的动态加载，而在切换顶部选项卡时没有进行图片的动态加载，所以图片无法显示。



此时还需要修改切换顶部选项卡时的代码，依然需要进行图片的动态加载， 代码如下：

此时切换顶部选项卡的结果如下图所示：



看起来数据没有任何的变化 -- 实质上数据是发生了变化的。其根本原因是：



初始情况下，articles变量已经存储了服务器返回数据（假设为[1,2,3]），在切换顶部选项卡时，我们仍然采用了数组的push()方法（此时得到就是[1,2,3,4,5,6]），这样的话，就会将切换顶部选项卡后得到的数据添加到了原始数据的未尾。所以看起来没有发生变化！！！

所以在切换顶部选项卡时需要将之前数据全部清空即可！！！



## 1.3 图片的懒加载

<img v-lazy="图像的URL地址">

## 1.4 分页显示数据

利用MySQL进行分页时，其根本原理在利用了SELECT语句的LIMIT子句来实现的，其基本语法结构是：

SELECT expression[,...]   
[FROM table\_referneces]   
[WHERE condition]   
[GROUP BY column\_name]   
[HAVING condition]   
[ORDER BY column\_name [ASC|DESC][,...]]   
[LIMIT [offset,]row\_count]

SELECT语句中子句的顺序是WGHOL

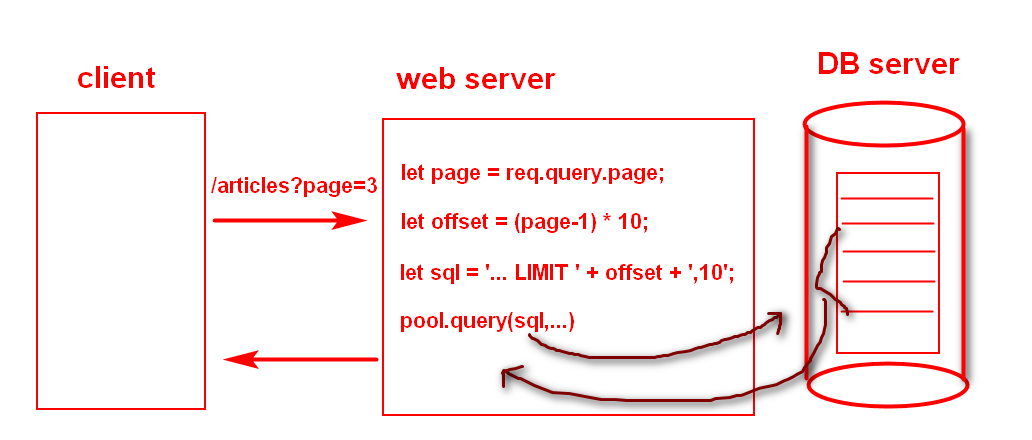
offset参数称偏移值，是指从第几条记录开始返回，其从0开始编号（与ID无任何关系）

该参数在参与分页时，其标准的计算公式是：

**(页码-1) \* 每页显示的记录数**

row\_count参数用于限制返回的数量，在参与分页时，只需要指定每页返回多少条即可。

在进行分页时，需要将页码从客户端作为URL地址栏的参数传递到服务器，然后在服务器上进行接收，并且根据上述的公式来计算需要获取的部分记录返回给客户端即可。



需要在页面初始化及切换顶部选项卡时完成初始情况下的获取部分数据的操作。

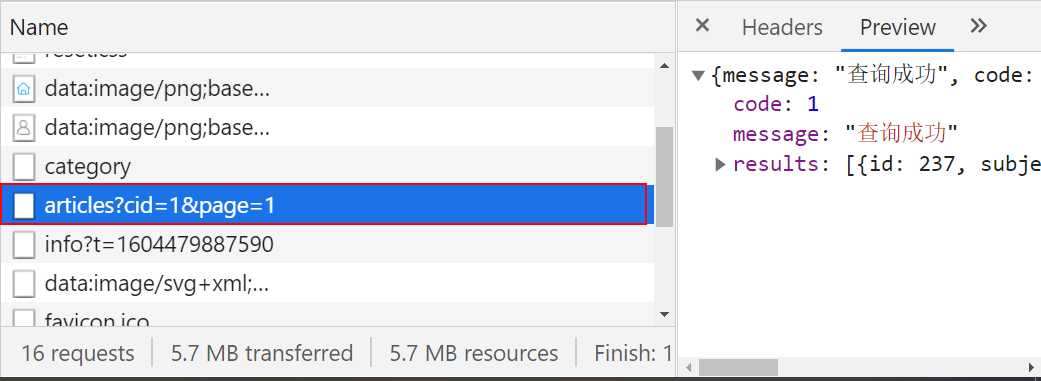
### 1.4.1 页面初始化获取部分记录

在经过刚才的描述后，我们发现在进行分页时必须要将要页码作为URL地址栏的参数告诉服务器才行，所以也就意味着在Home.vue中mounted中发送请求时，必须要带有页面的参数才行！

此时的请求代码要修改为：

//此时是告知服务器要获取哪一类(cid)的第1页的数据   
this.axios.get('/articles?cid=' + this.active + '&page=1').then(res=>{   
 //...   
});

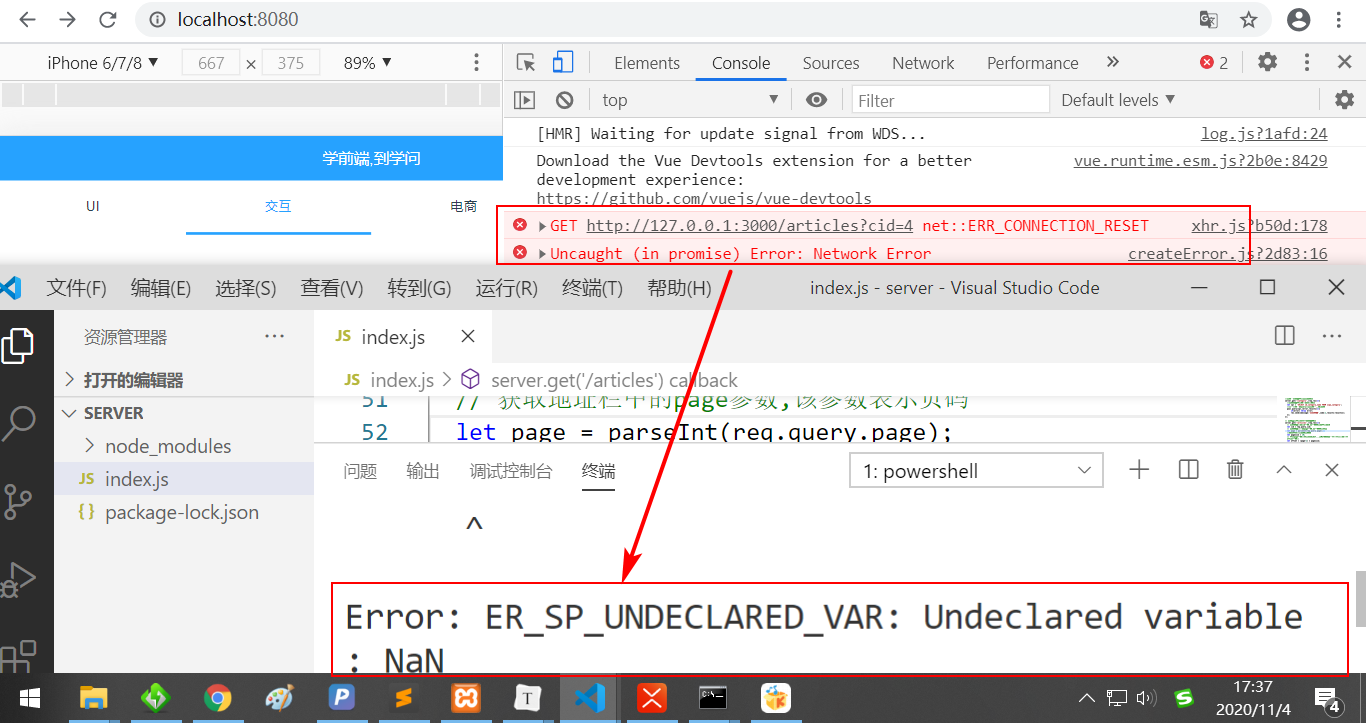
此时客户端运行的结果如下图所示：



现在已经证明请求可以正常发送了，而且发送给的是/articles的接口，但是目前在接口中没有接收，当然也就谈不上处理了。所以需要重新修改/articles接口，截图如下：

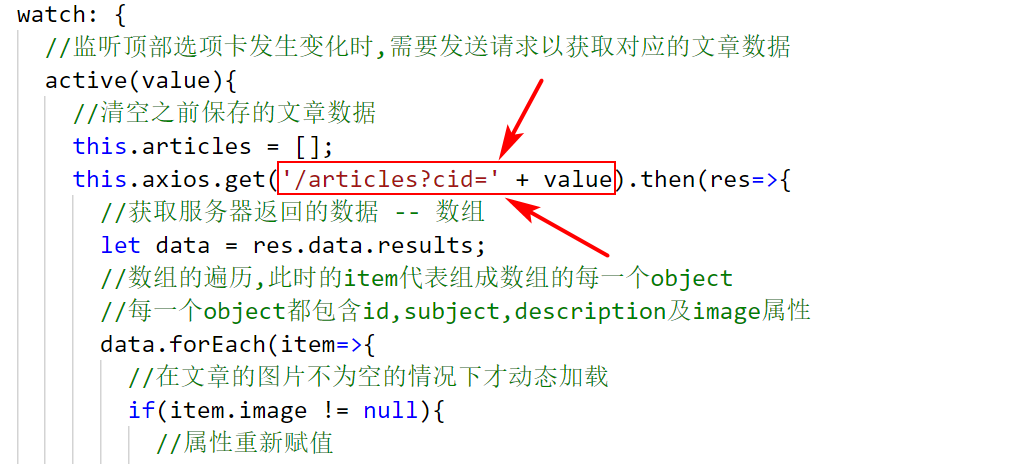


但在切换选项卡时运行效果如下图所示：



### 1.4.2 切换顶部选项卡时获取部分数据

切换顶部选项卡时调用的函数如下：



在该过程中发现，切换顶部选项卡时仍然调用的时/articles接口，而这个接口刚刚被修改过，在修改的结果中需要有两个参数进行传递，分别是cid和page，而现在只有cid，所以：

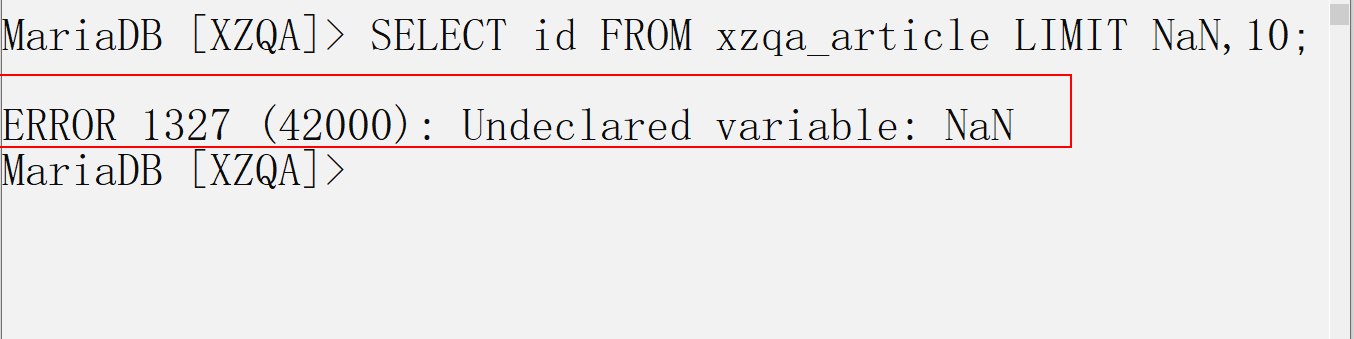
服务器在通过let page = parseInt(req.query.page)时，将返回NaN

如果此时得到的是NaN话，那么 let offset = (page-1) \* 10，也将返回NaN

如果上述过程得到了NaN，那么此时的SQL语句就成了

SELECT id,subject,description,image FROM xzqa\_article WHERE cid=3 LIMIT NaN,10

如果此SQL直接在MySQL客户端执行话，结果如下图所示：



所以，所以，所以！必须在切换顶部选项卡时传递page参数且值为1即可。



# Mint UI - Unit05

# 1.组件

## · Infinite scroll指令

Infinite scroll指令用于无限滚动，其语法结构是：

<html元素   
 infinite-scroll-distance="阈值"   
 v-infinite-scroll="触发的函数名称"   
 infinite-scroll-disabled="变量名称"   
 :infinite-scroll-immediate-check="true">   
</html元素>

infinite-scroll-distance属性用于指定距离容器底部还有多少像素时触发无限滚动的指令

v-infinite-scroll属性用于指定当触发无限滚动指定时需要调用的函数名称（因为只要触发无限滚动指令就意味着需要从服务器加载更多的数据，所以需要将加载数据的过程封装为一个函数）

infinite-scroll-disabled属性用于控制当上一次的无限滚动触发的函数没有执行完毕时，再次滚动到指定范围内时是否要继续触发无限指令，布尔类型

## · Indicator指令

Indicator指令用于显示加载提示框，其语法结构是：

//显示加载提示框   
//简捷语法   
this.$indicator.open("提示内容")   
//标准语法   
this.$indicator.open({   
 text:"提示内容",   
 spinnerType:"加载图标类型(snake|double-bounce|triple-bounce|fading-circle)"   
})   
//关闭加载提示框   
this.$indicator.close()

# 2.首页无限加载数据的实现

## 2.1 初始化选项卡的无限加载

首先确定无限滚动指令需要附加到的HTML元素，而且还需要为HTML元素添加相关的无限滚动指令。

<div   
 class="main"   
 infinite-scroll-distance="20"   
 v-infinite-scroll="loadMore"   
 infinite-scroll-disabled="busy">   
...   
</div>

既然已经在无限滚动指令中使用了busy变量和loadMore函数，所以就分别需要在data()和methods中来定义busy变量和loadMore函数，代码截图如下：

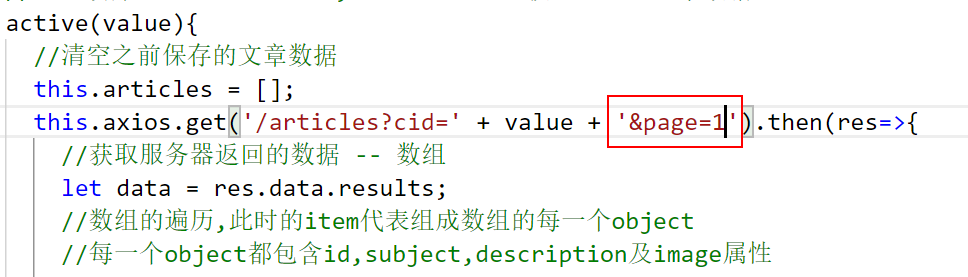


目前重要的事项就是loadMore函数的代码了，该函数中因为需要从服务器加载数据，所以一般情况下要显示加载提示框，以给用户良好的体验；除此之外，因为在服务器处理过程中，用户可以会继续向下滚动，而导致无数次触发滚动函数，所以还需要将busy变量的值修改为true，表示在服务器处理请求期间，即使用户再次进入滚动范围，也不再触发滚动函数，所以此时代码为：

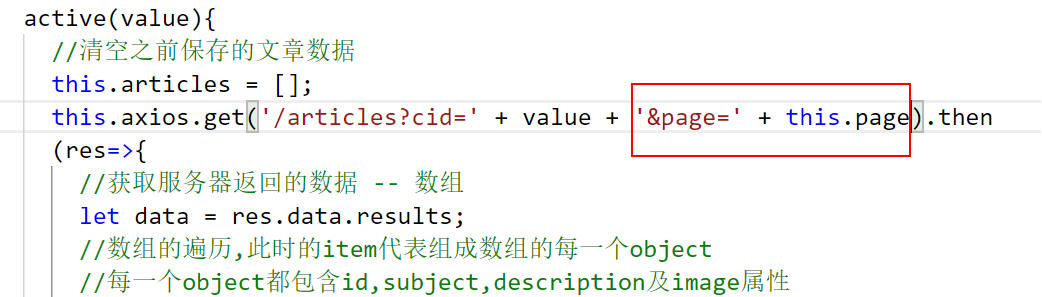
loadMore(){   
 this.$indicator.open('加载中...');   
 this.busy = true;   
 //现在就需要发送真正的请求了,而且该请求依然是获取部分的文章数据，所以   
 //可以继续使用articles接口，而在访接口中必须要存在cid和page两个参数   
 //所以此时的伪代码如下：   
 //this.axios.get('/articles?cid=当前分类的ID&page=当前的页码').then(...)   
 //在此过程中,cid的参数值为ative变量的变量值,所以代码修改为   
 //this.axios.get('/articles?cid=' + this.active +'&page=当前页的页码').then(...)   
 //在向下滚动的过程中，页码变量应该是进行累加操作的，所以需要在data()中声明页码的变量   
 //用户保存页码的初始值,在向下滚动时进行累加即可，故代码如下：   
 this.page++;   
 this.axios.get('/articles?cid=' + this.active + '&page=' + this.page).then(...);   
   
}

## 2.2 切换选项卡的无限加载

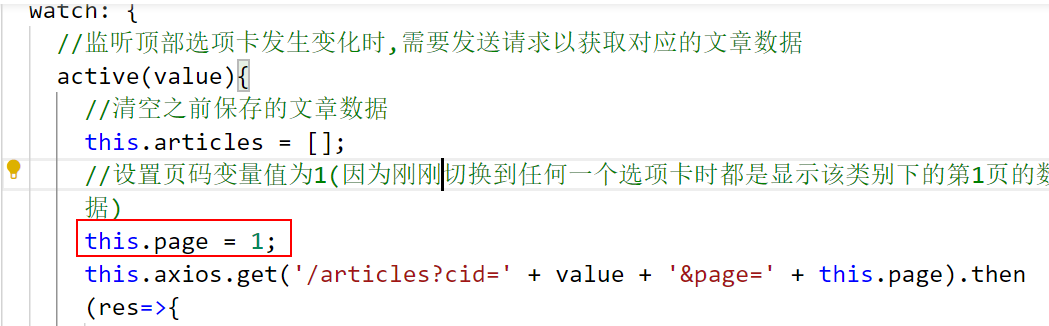
如果将切换选项卡的代码，如下图所示：



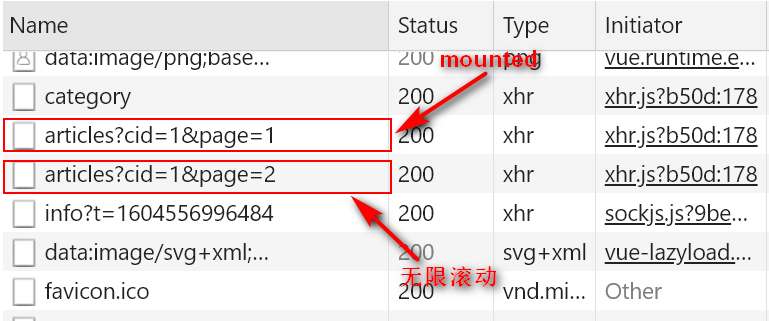
修改为如下图所示：



会造成如果在某一个（如UI）选项卡已经滚动了第9页，此时再切换到另外一个选项卡（如交互）那么此时将从服务器返回的是：交互类的第9页的文章数据，而应该获取的是这一类别的第1页的数据，所以需要强行将变量page的值修改为1



在初始化浏览时，可以在调试工具中发现竟然主动的调用了一次无限滚动指令



其根本原因是因为无限滚动会自动发送一次请求以获取更多的数据，其目的是为了防止数据数据量过少，而主动进行发送的请求(主动调用了一次无限滚动触发的函数 )。

所以必须为相关的HTML元素添加infinite-scroll-immediate-check="true"属性，截图如下：



## 2.3 自定义函数的封装

目前在三个地方都曾经向articles接口来请求数据，分别是在：

·mounted

·watch

·无限滚动

所以只需要将获取数据的过程封装为自定义函数，然后多次调用即可 -- loadData()

loadData(cid, page) {   
 //显示加载提示框   
 this.$indicator.open({   
 text:'加载中...',   
 spinnerType:'fading-circle'   
 });   
 // 修改busy变量值的值   
 this.busy = true;   
 //通过axios工具向Web服务器发送请求以获取文章数据   
 this.axios.get("/articles?cid=" + cid + "&page=" + page).then((res) => {   
 //获取服务器返回的数据 -- 数组   
 let data = res.data.results;   
 //数组的遍历,此时的item代表的是组成数组的每一个object   
 //每一个object都包含id,subject,description及image属性   
 data.forEach((item) => {   
 //在文章的图片不为空的情况下才动态加载   
 if (item.image != null) {   
 //属性重新赋值   
 item.image = require("../assets/images/articles/" + item.image);   
 }   
 //现在在无论是否图片为空都添加到以articles数组中了   
 this.articles.push(item);   
 });   
 //关闭加载提示框   
 this.$indicator.close();   
 //修改busy变量的值   
 this.busy = false;   
 });   
},

最后分别在mounted、watch及无限滚动时分别调用即可，截取如下：



## 2.4 当超过总页数时不应该继续发送请求

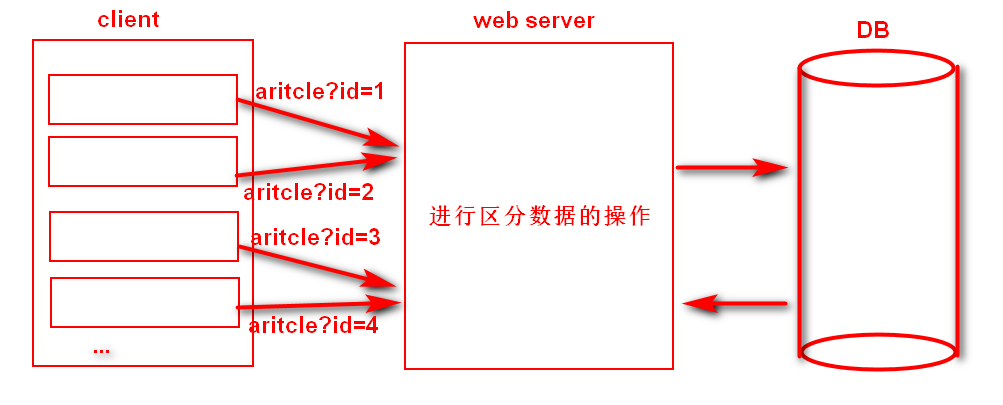
# 3.文章详情页的实现

在src/views目录创建Article.vue ，并且设置页面的结构及样式，然后在src/router/index.js中导入页面组件并且设置路由信息。

需要Home.vue中为文章添加<router-link>标签，链接刚刚的/article路由。

现在关键的问题是：要在Article.vue中显示不同的文章的详细信息！！！

既然是通过链接来实现的相关操作，那么就必须通过链接将需要查看的文章的ID传递到Article.vue，然后在Article.vue接收该数据，并且再将数据传递到服务器中才能真正的获取到相关文章的详细信息！！！



在目前为止，在通过<router-link>进行参数传递的方式有两种：

A：如

<router-link to="/article/5"></router-link>

第一：需要修改路由的规则，如path:"/article/:id"的形式

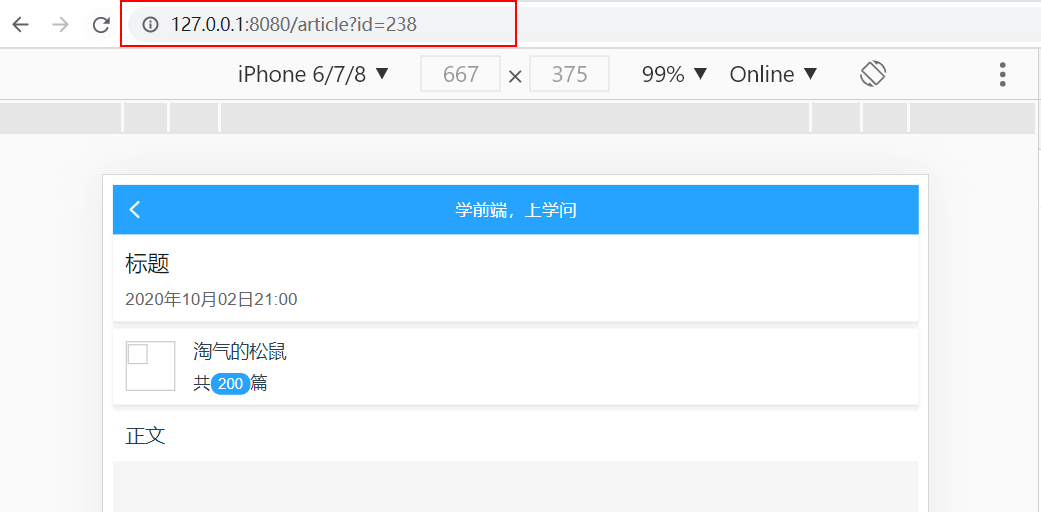
第二：目标页面通过 this.$route.params.id进行获取



B.如

<router-link to="/article?id=5"></router-link>

在目标页面，通过 this.$route.query.id 进行接收

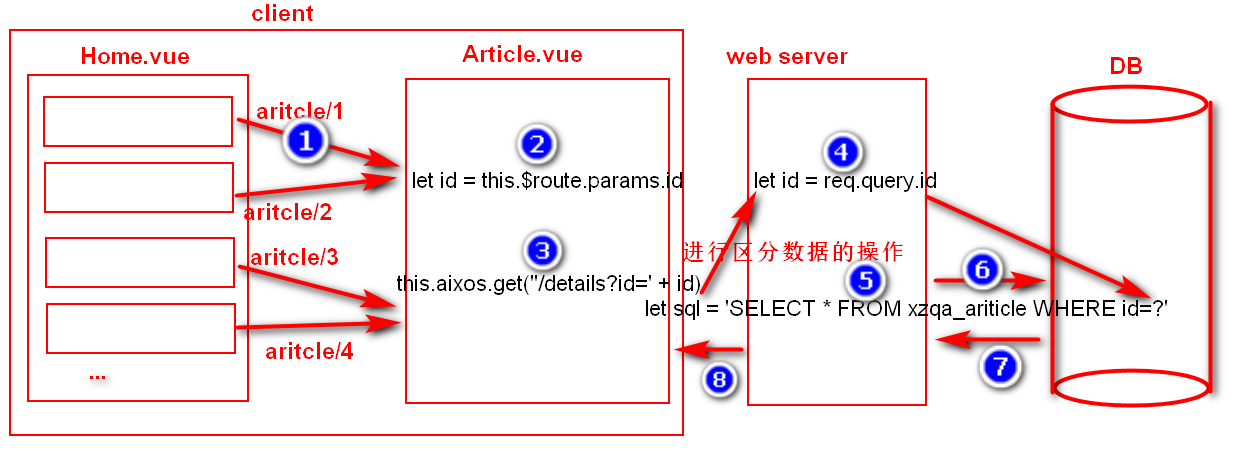


在该案例中使用第一种方式进行参数的传递，那么现在就需要在Article.vue的mounted生命周期回调函数中先取参数值，示例代码如下：

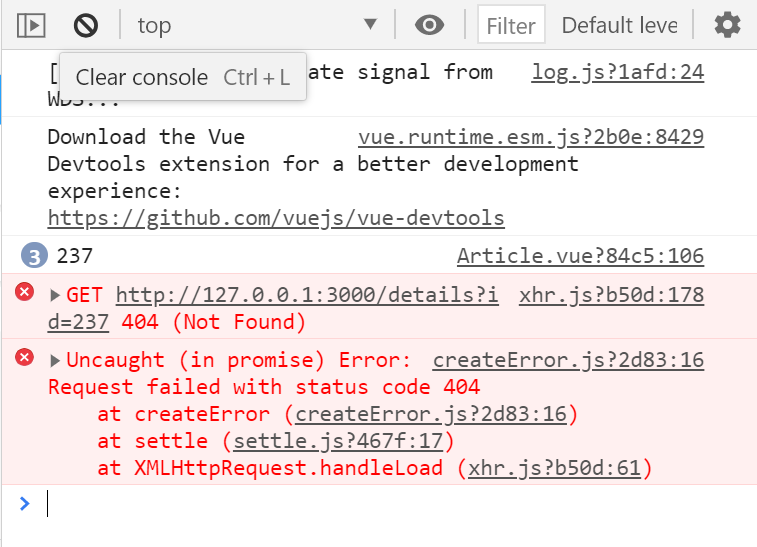
mounted(){   
 let id = this.$route.params.id;   
}

此时就需要向WEB服务器发送请求，以获取该文章ID的详细信息了，示例代码如下：

mounted(){   
 let id = this.$route.params.id;   
 //因为是要获取信息，所以请求类型一定是GET   
 //另外，现在要获取数据表中某一条记录的信息，也就代表必须给定一个可以确定唯一记录的数据   
 //才行，在整张数据表只有ID可以是唯一的！所以必须将上一个页面传递的ID在获取到后   
 //再传给服务器，并且以此为条件在数据表中进行查找。   
 this.axios.get('/details?id=' + id).then(res=>{   
   
 });   
}



现在如果运行页面的话，一定会产生404错误，根本原因是：服务器中不存在相关的接口



现在需要在index.js创建接口，规范如下：

请求类型：GET

参数:id，表示要获取的文章的ID

请求地址：http://127.0.0.1:3000/details

# Mint UI - Unit06

# 1.Moment.js

Moment.js是一个日期时间的类库，可以运行在浏览器及Node.js环境。

浏览器安装

<script src="moment.js文件URL地址"></script>

Node.js安装

npm install --save moment

Node.js配置Moment -- 在 src/main.js中实现

import moment from 'moment';   
   
Vue.prototype.moment = moment;

构建Moment对象

unix()方法

unix()方法用于将Unix时间戳转换为Moment对象，其语法结构是：

Moment Moment.unix(number)

format()方法

format()方法用于实现日期的格式化，其语法结构是：

String Moment.format([format])

常用的格式有：

Y表示四位的年份

MM表示两位的月份

DD表示两位的日期

HH表示两位小时

mm表示两位分钟

ss表示两位秒

# 2.用户注册的业务

用户注册业务的前端业务已经基本实现完成，示例代码如下：

handle(){   
 //会引发短路现象   
 if(this.checkUsername() && this.checkPassword() && this.checkConpassword()){   
 //现在要发送相关的用户名、密码到WEB服务器   
 //this.axios.请求方式(URL地址,参数).then(res=>{   
 //});   
 //因现在要需要将用户名密码等信息提交给服务器，所以现在的请求方式必须为POST   
 //提示:GET提交的信息会被浏览器缓存，且呈现在浏览器的地址栏中   
 //提示:所有的搜索引擎都是GET提交   
 //所以上述代码要修改为:(目前服务器暂不存在/register的接口)   
 //而且现在必须带有参数，因为需要将用户名和密码提交到服务器   
 //该参数要附加在HTTP请求体内，格式如： key=value&key=value&...   
 this.axios.post('/register','username=' + this.username + '&password=' + this.password).then(res=>{   
 });   
 }   
},

在index.js创建/register的接口，示例代码如下：

server.post('/register',(req,res)=>{   
 //获取用户名及密码信息   
 //因为现在是POST提交，所以只能通过req.body.username这种结构来进行获取操作   
 //但是req.body.username这种结构需要使用body-parser中间件   
 //所以现在需要在服务器端安装并且配置body-parser中间件   
 //npm install --save body-parser   
 //const bodyParser = require('body-parser');   
 //server.use(bodyParser.urlencoded({extended:false}));   
 //当完成上述三步之后，就可以获取用户名和密码的信息了   
 let username = req.body.username;   
 let password = req.body.password;   
 //现在虽然已经获取到用户名和密码，但是不能直接插入到数据表，因为在xzqa\_author表中需要   
 //保证用户名的唯一性，所以先需要进行查找操作：   
 let sql = 'SELECT COUNT(id) AS count FROM xzqa\_author WHERE username=?';   
 pool.query(sql,[username],(error,results)=>{   
 if(error) throw error;   
 if(results[0].count == 0){   
 //results代表的是结果集，为object array类型,   
 //在本案例中类似于 [{count:0}]或 [{count:1}]   
 //results[0].count的值如果为0的话，则代表用户不存在，可以进行插入操作   
 //在进行记录的插入操作时,要求密码必须采用MD5进行加密,以保证数据的安全性   
 //在Node.js的MySQL中可以通过两种方式来实现   
 //(1)直接采用MySQL数据库提供的MD5函数   
 //(2)安装使用MD5的Node.js模块   
 ////安装 npm install --save md5   
 ////配置 const md5 = require('md5');   
 ////使用,如 console.log(md5('12345678'));   
 sql = 'INSERT INTO xzqa\_author(username,password) VALUES(?,MD5(?))';   
 pool.query(sql,[username,password],(error,results)=>{   
 if(error) throw error;   
 //产生相关的信息返回到客户端   
 res.send({message:'注册成功',code:1}) ;   
 })   
 } else {   
 //否则要产生合理的"错误提示"给客户端   
 res.send({message:'注册失败',code:0})   
 }   
 })   
   
});

当完成上述操作后，需要重新启动服务器。

另外，现在服务器端已经根据注册时的情况返回信息给客户端了，那么必须在客户端接收信息，所示需要修改Register.vue中的handle函数中的代码，修改结果如下：

handle(){   
 //会引发短路现象   
 if(this.checkUsername() && this.checkPassword() && this.checkConpassword()){   
 //现在要发送相关的用户名、密码到WEB服务器   
 this.axios.post('/register','username=' + this.username + '&password=' + this.password).then(res=>{   
 //代表用户注册成功   
 if(res.data.code == 1){   
 this.$router.push('/');   
 } else {   
 //弹出提示框   
 this.$messagebox("提示信息","用户已经存在");   
 }   
 });   
 }   
}

# 3.用户登录的业务

用户登录的前端业务已经在Login.vue中实现了，此时需要完成提交用户名和密码到服务器进行检测的业务了，示例代码如下：

handle(){   
 //会引发短路现象   
 if(this.checkUsername() && this.checkPassword()){   
 //现在要发送相关的用户名、密码到WEB服务器   
 //其实现在还不存在相关的服务器接口，所以要修改index.js文件以添加login接口   
 this.axios.post('/login','username=' + this.username +'&password=' + this.password).then(res=>{   
 //   
 })   
 }   
}

服务器需要创建/login的接口，代码如下:

//用户登录的接口   
server.post('/login',(req,res)=>{   
 //获取用户名和密码   
 let username = req.body.username;   
 let password = req.body.password;   
 //以用户名和密码为条件进行查找   
 let sql = 'SELECT id,username,avatar,article\_number,nickname FROM xzqa\_author WHERE username=? AND password=MD5(?)';   
 pool.query(sql,[username,password],(error,results)=>{   
 if(error) throw error;   
 if(results.length == 0){   
 res.send({message:'登录失败',code:0});   
 } else {   
 res.send({message:'登录成功',code:1,userInfo:results[0]});   
 }   
 });   
   
});

QS模块

QS(QueryString)模块是将对象与请求字符串(a=1&b=2&c=3)之间进行相互转换。

安装

npm install --save qs

在main.js中进行配置

import qs from 'qs';   
Vue.prototype.qs = qs;

stringify()方法是将对象转换为请求字符串，语法结构是：

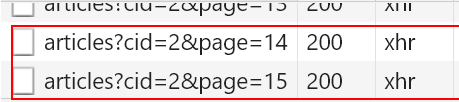
string qs.stringify(object)

parse()方法用于将请求字符串转换为对象，语法结构是：

object qs.parse(string)

# 4.无限滚动时的业务

在进行无限滚动时，即使超出了总页数，仍然会发起请求。



解决该问题方案就是：在服务器返回原始的信息之外，再返回一个总页数的信息，在客户端发起请求之前先判断一下当前页码是否超出了总页数的范畴，如果没有超出则发送请求，否则不再发送请求了。

此时必须再执行另外的一个SQL语句，以获取总记录数，为什么要获取总记录数呢？因为

总页数 = Math.ceil(总记录数 / 每页显示的记录数)

故现在最重要的事情就是获取总记录数，那么对应的SQL应该是：

SELECT COUNT(id) AS count FROM xzqa\_article WHERE category\_id = ?

此时的占位符的值已经在前期的业务中进行了获取操作：

let cid = req.query.cid;

# HTML5 新特性 - Unit01

# 1.Vuex

## 1.1 基础

Vuex是一个专门为Vue.js应用程序开发的状态管理模式，它采用集中式的存储来管理应用的所有组件的状态。

本质上来说，Vuex是Vue.js的插件

Vuex的核心就是Store，store是一个容器，包含了应用共享的状态。

所谓的状态管理实际上就是在各个页面组件中共享的信息

应用状态常见的有：

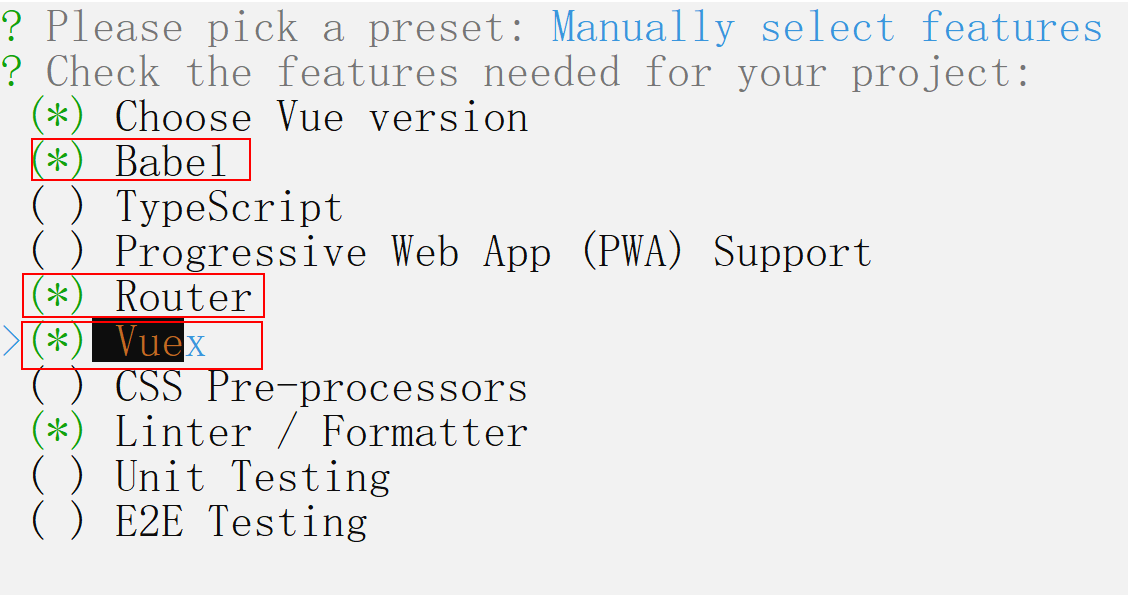
* 用户登录信息
* 购物车的相关信息

Vuex中的store之所以可以在所有的页面组件中被访问到，原因是是：Vuex自动的将状态从根组件注入到每一个子组件，而且store采用的单例模式(只能得到一个对象实例)。

## 1.2 安装

Vuex可通过以下两种方式进行安装：

A.在创建脚手架项目时，选取了Vuex插件



B.通过npm进行安装

npm install --save vuex

# 1.3 基本用法

Vuex的状态管理建议存储在store/index.js文件中

### 1.3.1 state

state中定义了应用共享的状态，其数据类型可以为string、number、boolean、array等，示例代码如下：

export default new Vuex.Store({   
 state:{   
 username:'Tom',   
 age:23,   
 sex:true,   
 products:[   
 {   
 productName:'商品1',   
 salePrice:100   
 },   
 {   
 productName:'商品2',   
 salePrice:200   
 }   
 ]   
 }   
   
})

### 1.3.2 页面组件中访问state的数据

this.$store.state.xx

示例代码如下：

<template>   
 <div>   
 <p>姓名:{{this.$store.state.username}}</p>   
 <p>年龄:{{this.$store.state.age}}</p>   
 <p>性别:{{this.$store.state.sex ? '女' : '男'}}</p>   
 <p v-for="(item,index) of this.$store.state.products" :key="index">   
 {{item.productName}}   
 {{item.salePrice}}   
 </p>   
 </div>   
</template>

### 1.3.3 mutations

mutations是改变状态的操作方法，也是vuex中修改state的唯一方法。其语法结构是：

mutations:{   
 //state参数代表是当前store内的所有state,会自动注入到当前方法的内部   
 //也就是说在提交该mutations时无需指定该参数   
 方法名称(state[,payload]){   
 //...   
 }   
}

payload意为载荷，实际上是额外的参数，数据类型可以为任意类型，建议使用object类型

示例代码如下：

export default new Vuex.Store({   
 mutations:{   
 changeAge(state){   
 state.age++;   
 }   
 login\_mutations(state,payload){   
 state.products.push(payload)   
 }   
 }   
})

<template>   
 <div>   
 <mt-button type="primary" @click="handle">改变年龄</mt-button>   
 <mt-field type="text" label="商品名称" v-model="productName">   
 </mt-field>   
 <mt-field type="text" label="商品价格" v-model="salePrice"></mt-field>   
 <mt-button type="primary" @click="add">添加商品</mt-button>   
 </div>   
</template>   
<script>   
 export default{   
 data(){   
 return {   
 productName:'',   
 salePrice:0   
 }   
 },   
 methods:{   
   
 }   
 }   
</script>

### 1.3.4 提交 mutations

this.$store.commit('方法名称'[,payload])

### 1.3.5 actions

actions与mutations的不同点在于：

·mutations只能进行同步操作，而actions可以进行异步操作

· Actions提交的是Mutations，而不能直接变更state

其语法结构是：

export default{   
 actions:{   
 方法名称(context[,payload]){   
   
 }   
 }   
}

### 1.3.5 分发actions

this.$store.dispatch("方法名称"[,payload])

# 2.WebStorage

WebStorage提供了名/值对进行数据的存储，其按存储机制可以分为：

localStorage，提供了一个独立的数据源区域，localStorage中的数据会永久存储

sessionStorage，提供了一个独立的数据源区域，sessionStorage中的数据只在当前标签页有效.

sessionStorage和localStorage可通过window对象的localStorage和sessionStorage属属性进行访问，如

window.sessionStorage   
window.localStorage   
sessionStorage   
localStorage

## setItem()方法

setItem()方法用于为指定的键名设置值，如果键名不存在，则自动创建，其语法结构是：

sessionStorage.setItem(key,value)   
   
localStorage.setItem(key,value)

getItem()方法

getItem()方法用于获取指定键名的值，如果键名不存在，则返回null，其语法结构是：

### sessionStorage.getItem(key)

### localStorage.getItem(key)

## removeItem()方法

removeItem()方法用于删除指定键名，如果键名不存在，则不进行任何操作，语法结构是：

### sessionStorage.removeItem(key)

### localStorage.removeItem(key)

clear()方法

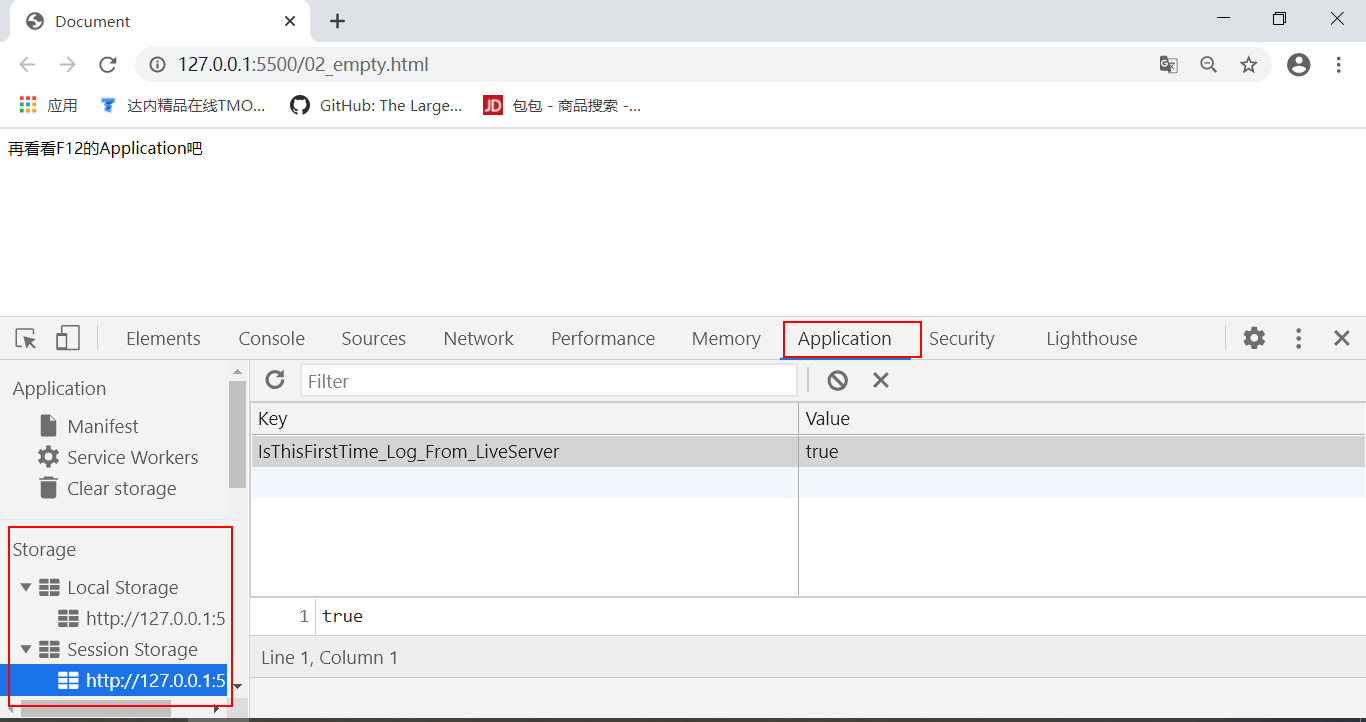
clear()方法用于清除所有的成员，其语法结构是：

sessionStorage.clear()   
   
localStorage.clear()

length方法

length属性用于获取键名的数量，其语法结构是：

sessionStorage.length   
   
localStorage.length





作业：完成用户登录状态的保持

# HTML5新特性 - Unit02

# 1.Vuex 辅助函数

辅助函数的优点：

A.将复杂的代码变得优雅

B.可以进行重命名

## · mapState()

mapState()函数用于为组件**创建计算属性**以返回state中的状态，其语法结构是：

mapState(数组 | 对象)

<template>  
 <div>  
 <p>姓名:{{username}}</p>  
 <p>年龄:{{age}}</p>  
 <p>性别:{{sex ? '女' : '男'}}</p>  
 <p>扩展:{{extension}}</p>  
 </div>  
</template>  
  
<script>  
import {mapState} from 'vuex';  
export default{  
 computed:{  
 ...mapState(['username','age','sex']),  
 ...mapState({  
 extension:'extends'  
 })  
 }  
}  
</script>

## · mapMutations

mapMutations()函数用于为**组件创建方法**以提交Mutations，其语法结构是:

mapMutations(数组 | 对象)

## · mapActions()

mapActions()函数用于为**组件创建方法**以分发actions，其语法结构是:

mapActions(数组 | 对象)



# 2.音频、视频

## 2.1 视频标签

浏览器支持的视频格式有：mp4、webm、ogg

简写方式为：

<video src="视频文件URL地址" width="宽度" height="高度">  
</video>

标准方式为：

<video width="宽度" height="高度">  
 <source src="视频文件URL地址" type="MIME类型">  
 ...  
</video>

mp4文件的MIME类型为：video/mp4

webp文件的MIME类型为：video/webm

ogg文件的MIME类型为：video/ogg

## 2.2 视频属性

### · controls

controls属性用于显示播放控件，布尔属性

### · autoplay

autoplay属性用于控制视频自动播放，布尔属性，须与muted属性组合使用

### · muted

muted属性用于控制视频是否静音播放，布尔属性

### · loop

loop属性用于控制视频是否循环播放，布尔属性

### · poster

poster属性用于设置视频的海报帧

### · preload

preload属性用于设置视频的加载方式，其值有：

* none，不进行任何视频加载操作
* auto，尽可能的加载视频
* metadata，只加载源数据（总时长、视频的宽度、高度等信息）

## 2.3 音频标签

浏览器支持的音频格式有：mp3、wav、ogg

简写方式

<audio src="音频文件URL地址">  
</audio>

标准方式

<audio>  
 <source src="音频文件URL地址" type="MIME类型">  
 ...  
</audio>

mp3文件的MIME类型为：video/mp3

wav文件的MIME类型为：video/wav

ogg文件的MIME类型为：video/ogg

## 2.4 音频属性

### · controls

controls属性用于显示播放控件，布尔属性

### · autoplay

autoplay属性用于控制音频自动播放，布尔属性，须与muted属性组合使用

### · muted

muted属性用于控制音频是否静音播放，布尔属性

### · loop

loop属性用于控制音频是否循环播放，布尔属性

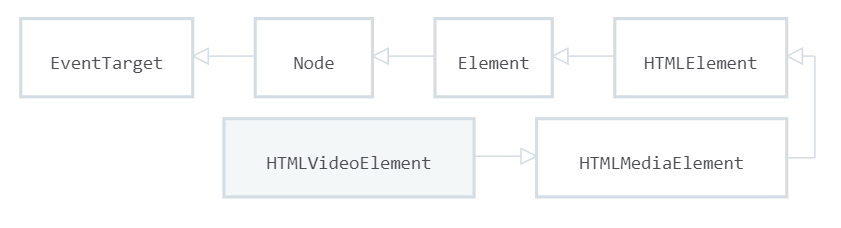
### · preload

preload属性用于设置音频的加载方式，其值有：

* none，不进行任何音频加载操作
* auto，尽可能的加载音频
* metadata，只加载元数据（总时长、音频的宽度、高度等信息）

# 3.HTMLVideoElement

HTMLVideoElement接口用于对Video元素进行控制，该接口继承自HTMLMediaElement



大部分HTML标签的单一单词属性即DOM对象的属性

HTML标签的class属性在DOM编程中使用className取代

HTML元素的style属性在DOM编程时将返回CSSStyleDeclaration或CSS2Properties对象

HTML标签的合成词属性(如body标签的bgcolor属性、table标签的cellpadding、

cellspacing属性)在DOM编程中采用驼峰标记法实现，如bgColor、cellPadding、cellSpacing等

某些HTML元素在DOM编程时拥有独立的属性或方法，如HTMLSelectElement的value属性

HTML元素的布尔属性在DOM编程时采用boolean类型表示

在DOM时对于存在构造函数的HTML元素，如(Image、Option、Audio等) 在创建时既可以通过构造函数实现，也可以通过document.createElement()方法实现；对于没有构造函数的HTML元素，如(div，p等)只能通过document.createElement()方法实现

https://www.w3.org/TR/dom41/

## · width

width属性用于获取/设置视频元素的宽度，其语法结构是：

//设置   
HTMLVideoElement.width = value   
//获取   
variable = HTMLVideoElement.width

在获取宽度属性时必须保证视频元素存在width属性

## · height

height属性用于获取/设置视频元素的高度，其语法结构是：

//设置   
HTMLVideoElement.height = value   
//获取   
variable = HTMLVideoElement.height

在获取高度属性时必须保证视频元素存在height属性

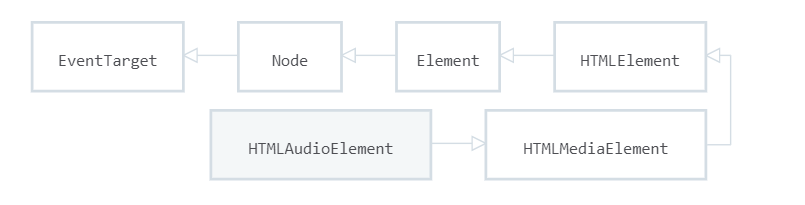
## · poster

poster属性用于获取/设置视频元素海报帧的URL地址，其语法结构是：

//设置   
HTMLVideoElement.poster = value   
//获取   
variable = HTMLVideoElement.poster

# 4.HTMLAudioElement

HTMLAudioElement接口用于对audio元素进行控制，该接口继承自HTMLMediaElement



## 4.1 构造函数

variable = new Audio([url地址])

# 5.HTMLMediaElement

HTMLMediaElement接口是HTMLAudioElement接口和HTMLVideoElement接口的父接口。

## 5.1 属性

### · controls

controls属性用于获取/设置是否显示媒体的播放控件，语法结构是：

//获取   
variable = HTMLMediaElement.controls   
//设置   
HTMLMediaElement.controls = value

### · autoplay

autoplay属性用于获取/设置媒体是否自动播放，语法结构是：

//获取   
variable = HTMLMediaElement.autoplay   
//设置   
HTMLMediaElement.autoplay = value

### · muted

muted属性用于获取/设置媒体在播放时是否静音，语法结构是：

//获取   
variable = HTMLMediaElement.muted   
//设置   
HTMLMediaElement.muted = value

### · loop

loop属性用于获取/设置是否媒体是否循环播放，语法结构是：

//获取   
variable = HTMLMediaElement.loop   
//设置   
HTMLMediaElement.loop = value

### · paused

paused属性用于获取媒体是否已经暂停，其语法结构是：

variable = HTMLMediaElement.paused

### · ended

ended属性用于获取媒体是否已经播放完毕，其语法结构是：

varaible = HTMLMediaElement.ended

### · volume

volume属性用于获取/设置媒体的音量，媒体的音量从0.0~1.0，其语法结构是：

//获取   
variable = HTMLMediaElement.volume   
//设置   
HTMLMediaElement.volume = value

### · playbackRate

playbackRate属性用于获取/设置媒体的播放速率，正常的播放速率为1.0，如果小于1.0，则慢，如果大于1.0,则快，其语法结构是：

//获取   
variable = HTMLMediaElement.playbackRate   
//设置   
HTMLMediaElement.playbackRate = value

### · currentTime

currentTime属性用于获取/设置媒体当前的播放时长，单位为秒，其语法结构是：

//获取   
variable = HTMLMediaElement.currentTime   
//设置   
HTMLMediaElement.currentTime = value;

### · duration

duration属性用于获取媒体总的播放时长，单位为秒，其语法结构是：

variable = HTMLMediaElement.duration

如果媒体不可用，将返回0

如果媒体可用，但时间长度未知，则返回NaN

## 5.2 方法

### · play()

play()方法用于实现媒体的播放，其语法结构是：

HTMLMediaElement.play()

### · pause()

pause()方法用于实现媒体的暂停，其语法结构是：

HTMLMediaElement.pause()

## 5.3 事件

### · loadedmetadata

loadedmetadata事件在元数据(如媒体的总时长、宽度、高度等)加载完毕后触发，其语法结构是：

HTMLMediaElement.onloadedmetadata = function(){   
 //...   
}   
   
HTMLMediaElement.addEventListener('loadedmetadata',function(){   
 //...   
})

### · loadeddata

loadeddata事件在媒体的第一帧加载完毕后触发，其语法结构是：

HTMLMediaElement.onloadeddata = function(){   
 //...   
}   
   
HTMLMediaElement.addEventListener('loadeddata',function(){   
 //...   
})

### · timeupdate

timeupdate事件在媒体对象的currentTime属性发生变化时触发，其语法结构是：

HTMLMediaElement.ontimeupdate = function(){   
 //...   
}   
   
HTMLMediaElement.addEventListener('timeupdate',function(){   
 //...   
})

### · pause

pause事件在媒体暂停时触发，其语法结构是：

HTMLMediaElement.onpause = function(){   
 //...   
}   
   
HTMLMediaElement.addEventListener('pause',function(){   
 //...   
})

### · play

play事件在媒体播放时触发，其语法结构是：

HTMLMediaElement.onplay = function(){   
 //...   
}   
   
HTMLMediaElement.addEventListener('play',function(){   
 //...   
})

# 6.全屏API

全屏API可以让任何的HTML元素及其子元素占满整个屏幕。

## requestFullscreen()方法

requestFullscreen()方法用于进入全屏，其语法结构是：

HTMLElement.requestFullscreen()

## exitFullscreen()

exitFullscreen()方法用于退出全屏，其语法结构是：

document.exitFullscreen()

# HTML5新特性 - Unit04

# 1.地理定位

## 1.1 基础

地理定位API允许用户向WEB应用程序提供他们的位置，出于隐私的考虑，在报告地理位置之前会请求用户进行授权。

从chrome 50开始，地位定位只能使用HTTPS协议。

地理定位API通过navigator对象geolocation属性进行获取，其语法结构是：

Geolocation window.navigator.geolocation   
   
Geolocation navigator.geolocation

## 1.2 Geolocation对象

### · getCurrentPosition()方法

getCurrentPosition()方法用于获取设备的当前位置，其语法结构是：

Geolocation.getCurrentPosition(success[,error[,options]])

success表示成功获取位置信息后的回调函数，使用GeolocaionPosition对象作为唯一参数

error表示获取位置信息失败后的回调函数，使用GeolocationPositionError对象作为唯一参数

options参数用于对获取位置时进行相关的设置，object类型， 包括：

◎ timeout属性，用于设置多长时间内（单位为毫秒）返回一个位置，如果在指定时间内没有获取到位置信息，则产生错误

◎ enableHighAccuracy属性，表示是否启用高精度结果

#### · GeolocationPosition对象

GeolocationPosition对象作为Geolocation对象的getCurrentPosition()方法的第一个参数，该对象表示设备的相关位置，包含有timestamp 和coords属性

##### ◎ timestamp属性

timestamp属性返回的是获取位置时的时间戳

##### ◎ coords属性

coords属性将返回GeolocationCoordinates对象，表示坐标的相关信息，包括：

※ longitude

longitude属性表示经度

※ latitude

latitude属性表示纬度

#### · GeolocationPositionError对象

##### ◎ code

code表示错误代码

##### ◎ message

message表示错误信息

## · watchPosition()方法

watchPosition()方法用于注册监听器，只要设备的位置发生变化，该监听器将自动调用，其语语法结构是：

id = Geolocation.watchPosition(success[,error[,options]])

## · clearWatch()方法

clearWatch()方法用于清理制指定的监听器，其语法结构是：

Geolocation.clearWatch(id)

# 2.百度地图JSAPI

百度地图JS API是一套基于JavaScript开发的应用程序接口，支持PC端和移动端。

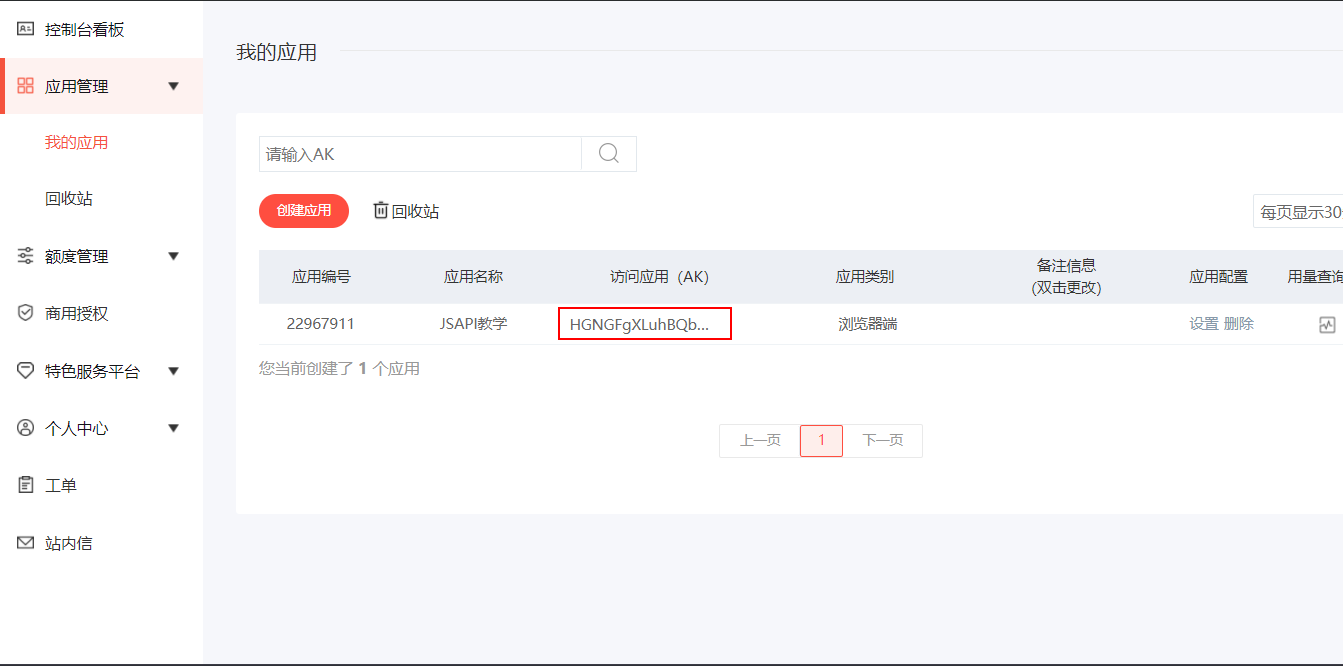
## 2.1 账号和获取密钥

A.注册百度账号

B.申请成为百度地图开放平台开发者

C.获取服务密钥





## 2.2 使用百度地图JSAPI

A.创建HTML5的页面，并且书写div元素，该元素将作为地图容器元素，并且设置访元素的样式

B.在HTML5页面中引入百度地图JS脚本文件，示例代码如下：

<script src="http://api.map.baidu.com/api?v=3.0&ak=您的密钥"></script>

C.创建地图实例并且设置中心点坐标

//获取地图容器的DIV元素   
let divEle = document.getElementById('map');   
   
//创建地图实例   
let map = new BMap.Map(divEle);   
   
//创建Point对象(中心点坐标对象) -- new BMap.Point(经度,纬度)   
let point = new BMap.Point(116.39528,39.930332);   
   
//设置中心点坐标及缩放层次   
map.centerAndZoom(point,15);

### 2.3 百度地图JSAPI

命名空间 API使用BMap作为命名空间，所有类均在该命名空间之下

自学一下高德地图JS API

# 3.拖放

拖放是将对象从一个位置拖拽到另外的一个位置，任何的HTML元素都可以进行拖放，但为保证浏览器的兼容性，建议在拖放的文本对象(如p，h1等)上添加draggable="true"属性，如:

<p draggable="true">段落</p>

## 3.1 DragEvent

DragEvent指拖放事件，其继承自MouseEvent和Event接口。

## 3.2 事件类型

### 3.2.1 源对象事件

#### · dragstart

dragstart事件在开始拖放时触发，只触发一次，其语法结构是：

HTMLElement.ondragstart = function(){   
 //...   
}   
HTMLElement.addEventListener('dragstart',function(){   
 //...   
})

#### · drag

drag事件在源对象拖放过程中触发，其触发无数次，其语法结构是：

HTMLElement.ondrag = function(){   
 //...   
}   
HTMLElement.addEventListener('drag',function(){   
 //...   
})

#### · dragend

dragend事件在释放鼠标左键时触发(也可能在目标区域内，也可能在目标区域外)，其语法结构是：

HTMLElement.ondragend = function(){   
 //...   
}   
HTMLElement.addEventListener('dragend',function(){   
 //...   
})

### 3.2.2 目标对象事件

#### · dragenter

dragenter事件在源对象进入目标对象范围时触发，其触发一次，语法结构是：

**目标对象**  
HTMLElement.ondragenter = function(){   
 //...   
}   
HTMLElement.addEventListener('dragenter',function(){   
 //...   
})

#### · dragover

dragover事件当源对象在目标对象范围内悬停时触发，其语法结构是：

HTMLElement.ondragover = function(event){   
 event.preventDefault();   
 //...   
}   
HTMLElement.addEventListener('dragover',function(event){   
 event.preventDefault();   
 //...   
})

在dragover事件中必须要带有

event.preventDefault()

否则不会触发目标对象的drop事件

#### · dragleave

dragleave事件在源对象拖离目标对象范围时触发，其语法结构是：

HTMLElement.ondragleave = function(){   
 //...   
}   
HTMLElement.addEventListener('dragleave',function(){   
 //...   
})

#### · drop

drop事件在源对象在目标对象范围内释放时触发，其语法结构是：

HTMLElement.ondrop = function(event){   
 event.preventDefault();   
 event.stopPropagation();   
 //...   
}   
HTMLElement.addEventListener('drop',function(event){   
 event.preventDefault();   
 event.stopPropagation();   
 //...   
})

拖放事件正常的触发顺序：

dragstart->drag->dragenter->dragover->drop->dragend

# HTML5新特性 - Unit05

# 1.拖放 -- DragEvent

## 1.1 dataTransfer属性

dataTransfer属性用于获取/保存拖放过程中的数据，返回DataTransfer对象，语法结构是：

DataTransfer DragEvent.dataTransfer

## 1.2 DataTransfer对象

DataTransfer对象用于获取/保存拖过程中的数据

### · setData()方法

setData()方法用于为一个指定类型设置数据，如果类型不存在，则自动添加，其语法结构是：

DataTransfer.setData(type,value)

### · getData()方法

getData()方法用于获取指定类型的数据，其语法结构是：

DataTransfer.getData(type)

· files属性

files属性将返回FileList的对象，该对象可能来源于表单内的<input type="file">元素，也可能来源于用户的拖放，其语法结构是：

FileList DataTransfer.files

## 1.3 FileList对象

FileList的对象，该对象可能来源于表单内的<input type="file">元素，也可能来源于用户的拖放。

### · length属性

length属性用于获取FileList对象中包含的File对象的数量，其语法结构是：

FileList.length

### · item()方法

item()方法用于返回FileList中指定索引值的File 对象，其语法结构是：

File FileList.item(index)

## 1.4 FormData对象

FormData对象提供键/值表示的表单数据，经过它的数据可以通过AJAX提交

### · 构造函数

variable = new FormData()

### · set()方法

set()方法用于给FormData设置属性值，如果对应的属性已经存在，则产生覆盖，其语法结构是：

FormData.set(key,value)

### · get()方法

get()方法用于获取指定属性的值，其语法结构是：

FormData.get(key)

### · append()方法

append()方法用于添加一个新值到已经存在的属性中，如果属性不存在，则自动创建，其语法结构是：

FormData.append(key,value)

# 2.Multer中间件

Multer中间件用于处理enctype="multipart/form-data"类型表单（实质上就是用于实现文件的上传操作）。

Multer是一个路由级别的中间件。

## 2.1 安装与配置

npm install --save multer

配置(简化)

//加载Multer模块   
const multer = require('multer');   
//创建Multer对象   
const upload = multer({   
 dest:'upload'   
});

简化配置造成的结果是文件只有主文件名(Multer随机生成的)，没有扩展名

配置(标准) -- 自定义上传规则

第一步：定义规则

variable = multer.diskStorage({   
 //用于设置上传时目录的相关规则   
 destination:function(req,file,cb){   
 // req,当前的HTTP请求对象

// file,当前上传的文件对象

// cb(callback),回调函数,让multer按照指定的规则进行相关的操作

},   
 //用于设置上传时文件名称的相关规则   
 filename:function(req,file,cb){   
   
 }   
})

第二步：使用规则

const upload = multer({   
 storage:规则   
});

## 2.2 使用Multer中间件

### · single()方法

single()方法用于实现单文件上传，其语法结构是：

Multer对象.single('浏览框名称')

### · array()方法

array()方法用于实现多文件上传，其语法结构是：

Multer对象.array('浏览框名称')

## 2.3 req.files和req.file

如果进行多文件上传时，所以成功上传的文件信息存储req.file**s**数组对象中；

如果进行单文件上传时，所以成功上传的文件信息存储req.file对象中；

文件信息将得到一个object,该object称其为文件对象

## 2.4 文件对象

· originalname，文件原始名称（包括扩展名）

· destination，文件上传后的保存位置，也就是传到了哪个目录内

· filename，文件上传后的名称

· path，文件上传后的保存位置及名称（destiname + filename）

例如：

|  |
| --- |
| [  {  fieldname: 'avatar',  originalname: '216322.jpg',  encoding: '7bit',  mimetype: 'image/jpeg',  destination: 'upload',  filename: '43a36f40-27aa-11eb-8789-972f13e73cf6.jpg',  path: 'upload\\43a36f40-27aa-11eb-8789-972f13e73cf6.jpg',  size: 272523  }  ] |

# 3.UUID

UUID，通用唯一标识符，其目的是为了让分布式系统中的所有元素都有唯一的标识符。

UUID是一组32位的16进制字符组成的字符串，格式为8-4-4-4-12

安装

npm install --save uuid

配置

const uuid = require('uuid')

使用

//产生基于时间戳的UUID   
uuid.v1();   
//产生基于随机数的UUID   
uuid.v4();

FileReader

https://developer.mozilla.org/zh-CN/docs/Web/API/FileReader

# HTML5新特性 - Unit06

# 1.WebSocket

## 1.1 什么是WebSocket?、Socket.io?

WebSocket是一个网络通信协议，其最大的特点是：服务器可以向客户端推送消息；客户端也可以向服务器发送消息，是真正平等的双向对话，属于服务器推送技术的一种。

WebSocket协议诞生于2008年，`2011年成为国际标准。

Socket.io是一个为浏览器与服务器之间提供实时、双向的、基于事件网络通信库。 Socket.io实现了WebSocket协议，抹平一些技术细节及兼容性。

Socket.io包括：

· Node.js服务器

· 浏览器的JavaScript客户端

## 1.2 WebSocket 的优点

· 数据格式比较轻量、通信高效

· 既可以发送文本信息，也可以发送二进制数据

· 没有同源的限制，可以与任何的服务器进行数据交互

· 协议的前缀是ws(如果加密，则为wss)

WebSocket协议是基于TCP的一种新的网络协议。它实现了浏览器与服务器全双工(full-duplex)通信——允许服务器主动发送信息给客户端。

## WebSocket的使用场景

社交聊天、弹幕、多玩家游戏、协同编辑、股票基金实时报价、体育实况更新、视频会议/聊天、基于位置的应用、在线教育、智能家居等需要高实时的场景

## 由轮询到WebSocket

## （1） 轮询

客户端和服务器之间会一直进行连接，每隔一段时间就询问一次。客户端会轮询，有没有新消息。这种方式连接数会很多，一个接受，一个发送。而且每次发送请求都会有Http的Header，会很耗流量，也会消耗CPU的利用率。

## （2） 长轮询

长轮询是对轮询的改进版，客户端发送HTTP给服务器之后，有没有新消息，如果没有新消息，就一直等待。当有新消息的时候，才会返回给客户端。在某种程度上减小了网络带宽和CPU利用率等问题。但是这种方式还是有一种弊端：例如假设服务器端的数据更新速度很快，服务器在传送一个数据包给客户端后必须等待客户端的下一个Get请求到来，才能传递第二个更新的数据包给客户端，那么这样的话，客户端显示实时数据最快的时间为2×RTT（往返时间），而且如果在网络拥塞的情况下，这个时间用户是不能接受的，比如在股市的的报价上。另外，由于http数据包的头部数据量往往很大（通常有400多个字节），但是真正被服务器需要的数据却很少（有时只有10个字节左右），这样的数据包在网络上周期性的传输，难免对网络带宽是一种浪费。

## （3）  WebSocket

现在急需的需求是能支持客户端和服务器端的双向通信，而且协议的头部又没有HTTP的Header那么大，于是，Websocket就诞生了！流量消耗方面，相同的每秒客户端轮询的次数，当次数高达数万每秒的高频率次数的时候，WebSocket消耗流量仅为轮询的几百分之一

**WebSocket协议原理**

Websocket是应用层第七层上的一个应用层协议，它必须依赖 HTTP 协议进行一次握手 ，握手成功后，数据就直接从 TCP 通道传输，与 HTTP 无关了。

Websocket的数据传输是frame形式传输的，比如会将一条消息分为几个frame，按照先后顺序传输出去。这样做会有几个好处：

1 大数据的传输可以分片传输，不用考虑到数据大小导致的长度标志位不足够的情况。

2 和http的chunk一样，可以边生成数据边传递消息，即提高传输效率。

## （4）WebSocket和Socket的区别与联系

首先，Socket 其实并不是一个协议。它工作在 OSI 模型会话层（第5层），是为了方便大家直接使用更底层协议（一般是 TCP 或 UDP ）而存在的一个抽象层。Socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口(API)。

Socket通常也称作”套接字”，用于描述IP地址和端口，是一个通信链的句柄。网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个Socket，一个Socket由一个IP地址和一个端口号唯一确定。应用程序通常通过”套接字”向网络发出请求或者应答网络请求。

Socket在通讯过程中，服务端监听某个端口是否有连接请求，客户端向服务端发送连接请求，服务端收到连接请求向客户端发出接收消息，这样一个连接就建立起来了。客户端和服务端也都可以相互发送消息与对方进行通讯，直到双方连接断开

## 1.3 安装

浏览器

<script src="socket.io.js文件URL地址"></script>

https://cdn.jsdelivr.net/npm/socket.io-client@2/dist/socket.io.js

当在浏览器引入socket.io客户端脚本后会自动暴露名称为io的函数

创建Socket客户端以连接到Socket服务器

variable = io('ws://socket服务器地址')

服务器

npm install --save socket.io

## 1.4 客户端API

· emit()方法

emit()方法用于客户端向服务器广播事件，其语法结构：

client.emit('eventName'[,arguments[,...]])

eventName译为事件名称，其实质上就是一个key名称

arguments参数指向服务器广播事件时需要传递的参数，可以为任意数据类型

## 1.5 服务器端API

· on()方法

on()方法用于实现侦听客户端的事件,并且在侦听到之后触发相应的回调函数，其语法结构是：

server.on('eventName',callback)

· emit()方法

emit()方法用于服务器向客户端广播事件，其语法结构：

server.emit('eventName'[,arguments[,...]])

# 2.WebWorker

JavaScript采用单线程模型，也就是说所有的任务只能在一个线程上完成，一次只能做一件事。WebWorker就是为JavaScript创造多线程环境，通过主线程来创建worker线程，可以将繁重的计算工作交由worker线程来实现，此时可以减轻主线程的工作压力。

## 2.1 WebWorker使用时的限制

· 同源策略，必须保证主线程文件与worker线程的JS文件同源

· DOM限制，因为主线程和worker线程是两个相对独立的线程，所以在worker线程不能进行任何的DOM操作，但可以访问location、navigator对象。

· 通信联系，主线程与Worker线程不能直接通信，只能通过消息实现

· 脚本限制，Worker线程可以使用XMLHttpRequest对象发送AJAX请求

· 文件限制，worker线程无法直接读取本地文件，必须来源于网络路径

## 2.2 WebWorker线程(主线程)

### · 创建webworker线程

variable = new Worker('脚本文件URL地址')

· postMessage()方法

postMessage()方法用于实现主线程向worker线程发送消息，其语法结构是：

Worker线程.postMessage(message)

## 2.4 WebWorker线程内部

· onmessage属性

onmessage属性用于监听主线程发送的消息，其语法结构是：

this.onmessage = function(e){   
   
}   
self.onmessage = function(e){   
   
}

# HTML5新特性 - Unit07

# 1.WebWorker

## 1.1 主线程

### · 创建webworker线程

variable = new Worker('脚本文件URL地址')

### · postMessage()方法

postMessage()方法用于实现主线程向worker线程发送消息，其语法结构是：

Worker线程.postMessage(message)

### · onmessage属性

onmessage属性用于监听worker线程发送的消息，其语法结构是：

Worker线程.onmessage = function(e){   
 //e代表MessageEvent事件对象,包含有data属性,其值为worker线程向主线程发送的消息   
}

## 1.2 WebWorker线程

### · onmessage属性

onmessage属性用于监听主线程发送的消息，其语法结构是：

this.onmessage = function(e){   
 //e代表MessageEvent事件对象,包含有data属性,其值为主线程向worker线程发送的消息   
   
}   
self.onmessage = function(e){   
 //e代表MessageEvent事件对象,包含有data属性,其值为主线程向worker线程发送的消息   
}

### · postMessage()方法

postMessage()方法用于实现worker线程向主线程发送消息，其语法结构是：

this.postMessage(message)   
   
self.postMessage(message)

# 2.SVG

## 2.1 什么是SVG?

SVG(Scalable Vector Graphics)，可缩放的矢量图形，是基于XML语法的图像格式。

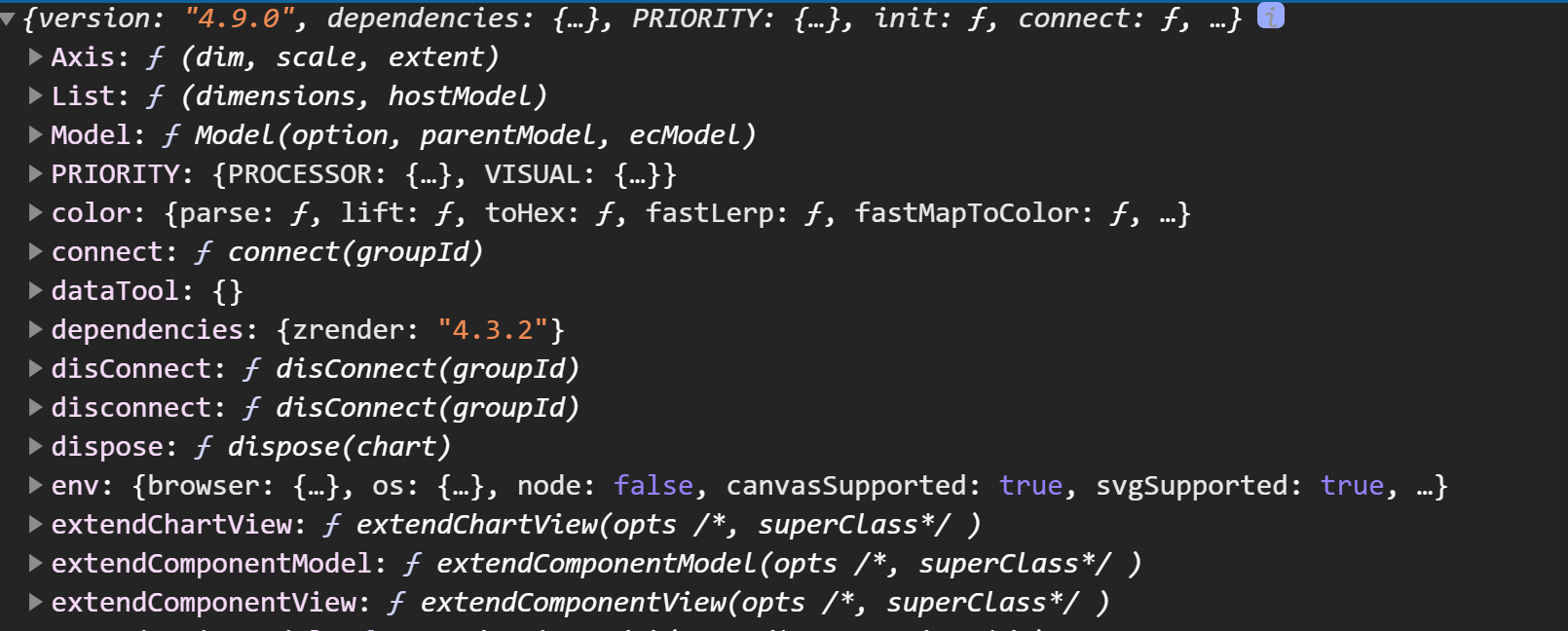
SVG的标准由w3c制定和维护。

# 3. SVG是什么？

<https://www.php.cn/html5-tutorial-413456.html>

SVG是一种基于XML的矢量图形格式，用于在Web和其他环境中显示各种图形；它允许我们编写可缩放的二维图形，并可通过CSS或JavaScript进行操作。

|  |
| --- |
| <script src="echarts/echarts.js"></script>    <script>      // echart类型      console.log(echarts);    </script> |



## 2.2 SVG的使用方式

### · <img>标签

<img src="SVG文件的URL地址">

### · background 样式

selector{   
 background-image:url(SVG文件URL地址)   
}   
   
selector{   
 background:url(SVG文件URL地址)   
}

### · object标签

object标签可以引入一个外部资源，其语法结构是：

<object type="MIME类型" data="SVG文件URL地址">   
 浏览器不支持该类型的文件时的提示文本   
</object>

SVG文件的MIME类型是image/svg+xml

扩展：

CSS文件的MIME类型是： text/css

html文件的MIME类型是： text/html

gif文件的MIME类型是： image/gif

jpg文件的MIME类型是： image/jpeg

png文件的MIME类型是： image/png

### · embed标签

<embed type="MIME类型" src="SVG文件URL地址">   
 浏览器不支持该类型的文件时的提示文本   
</embed>

### · iframe标签

<iframe src="SVG文件URL地址" frameborder="0/1" scrolling="yes/no/auto">   
</iframe>

扩展：

之前的iframe实现的网页嵌套 -- 在多个页面中共享的广告

现在一般在要服务器编程时采用SSI技术实现

### · svg标签

<svg version="1.1" xmlns="http://www.w3.org/2000/svg">   
 ...   
</svg>

version代表的SVG的版本，值只能为1.1

xmlns代表XML 命名空间(XML Namespace)，XML的命名空间用于解决标记名称冲突。

扩展：

HTML中标签的含义及作用是由W3C来制定的，如p表示段落

而在XML中标签的含义及作用由定义者来规定，如用户A用table表示图表的信息也是完全允许的。

XML是一种允许用户自定义标记的语言，如用户A用table标记来表示桌子的相关信息，而用户B用table来表示表格的相关信息，现在有用户C需要将用户A和用户B的文档进行整合，如果此时直接书写table的话，那么table到底代表的是用户A的桌子还是用户B的表格已经无法分清了。所以在XML中就引入了命名空间的概念，也就是将不同的文档的放置在不同的空间下（空间可以想像成目录），此时就可以通过空间下的标签名称进行有效的区分了！

## 2.3 SVG应用场景

· 地图

· 图表

## 2.4 图形元素

### · 外观属性

外观属性既可以直接书写于SVG元素，也可以书写于SVG元素的CSS属性

<line x1="10" y1="10" x2="200" y2="10" stroke="#000" stroke-width="5"/>

<style>   
 .line{   
 stroke:#000;   
 stroke-width:5px;   
 }   
</style>   
<line x1="10" y1="10" x2="200" y2="10" class="line"/>

stroke ，控制描边颜色

stroke-width，控制描边宽度

stroke-opacity，控制描边透明度

fill，控制填充颜色

fill-opacity，控制填充透明度

### · <line>元素

<line>元素用于绘制线条，其语法结构是：

<line x1="起点X轴" y1="起点Y轴" x2="终点X轴" y2="终点Y轴"></line>

### · <polyline>元素

<polyline>元素用于绘制开放的折线，其语法结构是：

<polyline points="x1,y1,x2,y2,x3,y3,...">   
</polyline>

### · <rect>元素

<rect>元素用于绘制(圆角)矩形，其语法结构是：

<rect x="起点X轴" y="起点Y轴"   
 width="宽度" height="高度"   
 rx="水平圆角半径" ry="垂直圆角半径">   
</rect>

## 2.5 SVGElement接口

W3C DOM分为Core DOM、HTML DOM、`XML DOM``

`Core DOM ：核心DOM，针对任何结构化文档(如HTML文档、XML文档等)的标准模型

XML DOM：用于XML文档的标准模型，对XML元素进行操作

HTML DOM： 用于HTML文档的标准模型，对HTML元素进行操作

## · createElementNS()

createElementNS()方法用于创建指定命名空间下的元素，其语法结构是：

Element document.createElementNS(string namespace,string tagName)

## · setAttribute()方法

## · getAttribute()方法

https://c.runoob.com/more/svgeditor/

# 3.ECharts

ECharts是百度公司开源的数据可视化工具，其基于JavaScript开发，可运行在浏览器及移动端。

## 3.1 安装

浏览器

直接在网页中通过 <script>标签来引入外部的ECharts脚本文件即可，如：

<script src="ECharts脚本文件URL地址"></script>

当引用ECharts的脚本文件后将自动暴露名称为echarts的全局对象

NPM安装

npm install --save echarts

## 3.2 ECharts的使用步骤

A.在HTML中通过<script>标签引入ECharts的脚本文件

B.在HTML页面中创建DIV元素，并且设置该元素的宽度和高度的样式(该DIV元素将作为渲染容器)

C.书写<script>标签，并且在其中调用echars对象的init()方法来**创建图表实例**

//获取DIV元素   
let divEle = document.getElementById('container');   
//创建图表实例,并且指定DIV元素将作为图表的容器   
let chart = echarts.init(divEle);

D.调用**图表实例**的setOption()方法对图表进行配置，如：

chart.setOption({...})

注册全新的邮箱

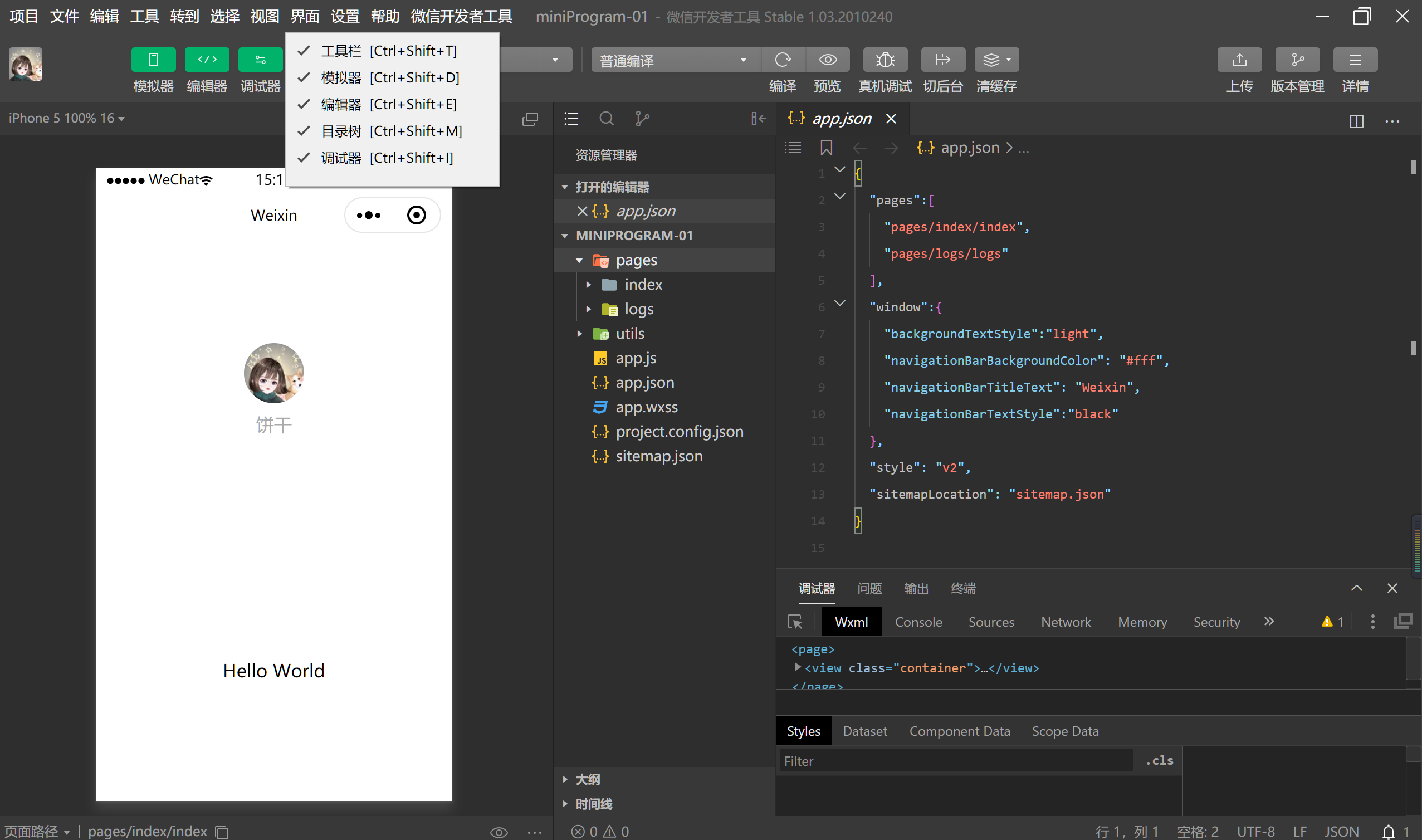
下载并安装微信开发者工具

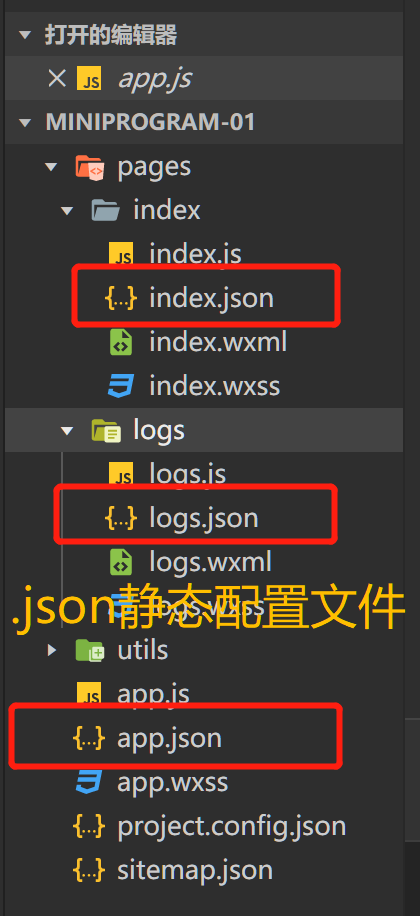
<https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html>

# 小程序









全局配置信息文件

# 微信小程序 - Unit01

# 1.微信公众平台

## 1.1 概述

微信公众平台，是基于微信服务器给个人、企业和组织提供用户管理及业务服务的服务平台。

微信公众平台的账号类型包括：

· 服务号，为企业和组织提供强大的业务服务和用户管理能力，主要偏向于服务类交互，如中国移动、中国联通、招商银行等。

· 订阅号，为媒体和个人提供一种全新的信息传播方式，主要偏向于咨讯传达，如CSND、前端大全、达内教育等。

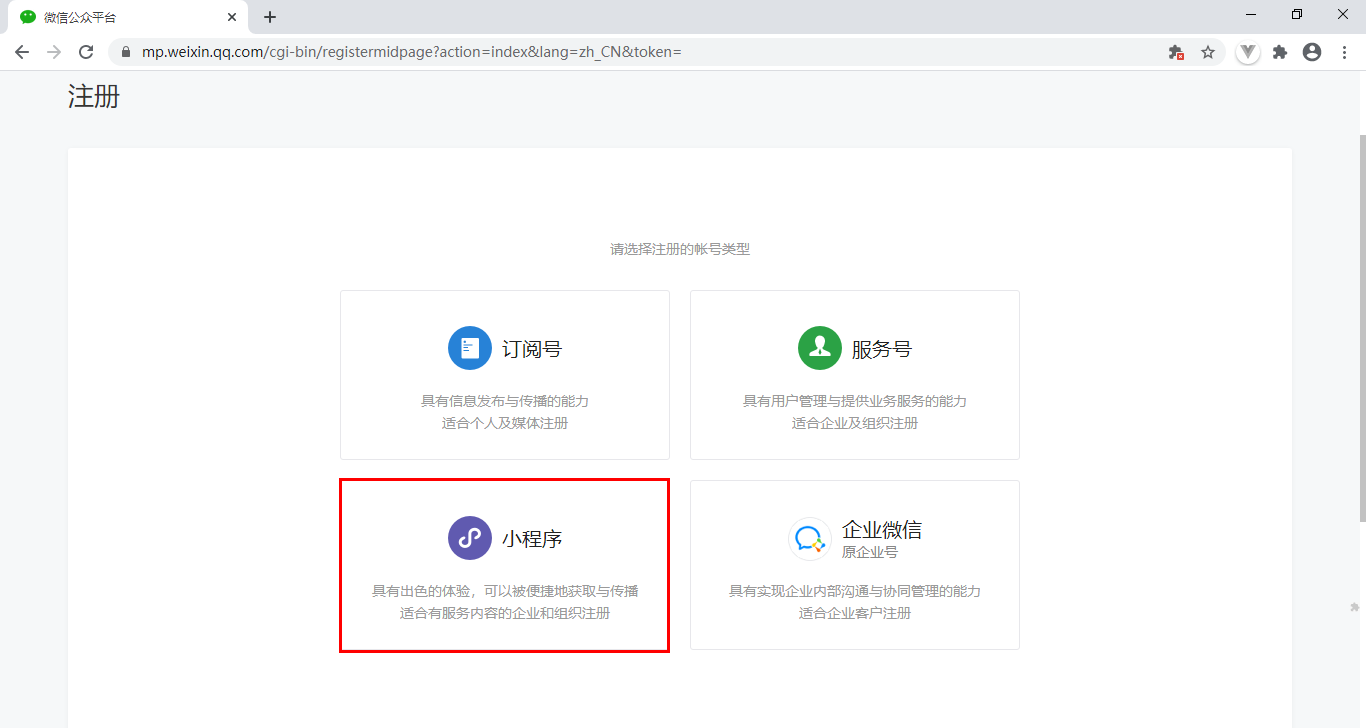
· 小程序，为企业、组织和个人提供一种无需安装即可使用的、具有原生APP体验的服务，达到"用完即走"的目的，如大众点评、KFC等

微信公众平台官网： https://mp.weixin.qq.com

## 1.2 小程序接入

A.微信公众平台注册开发者 (https://mp.weixin.qq.com)

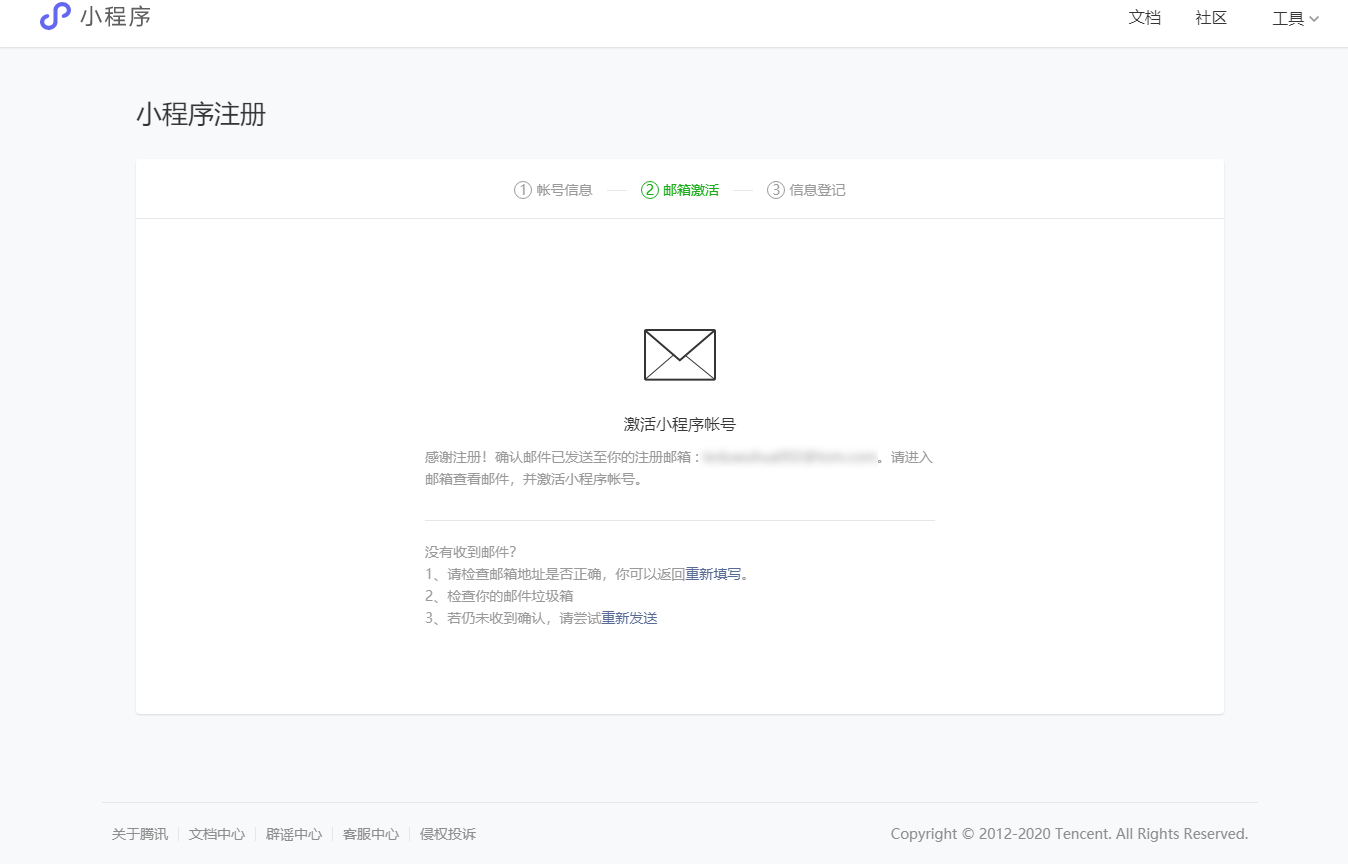




B. 填写注册信息

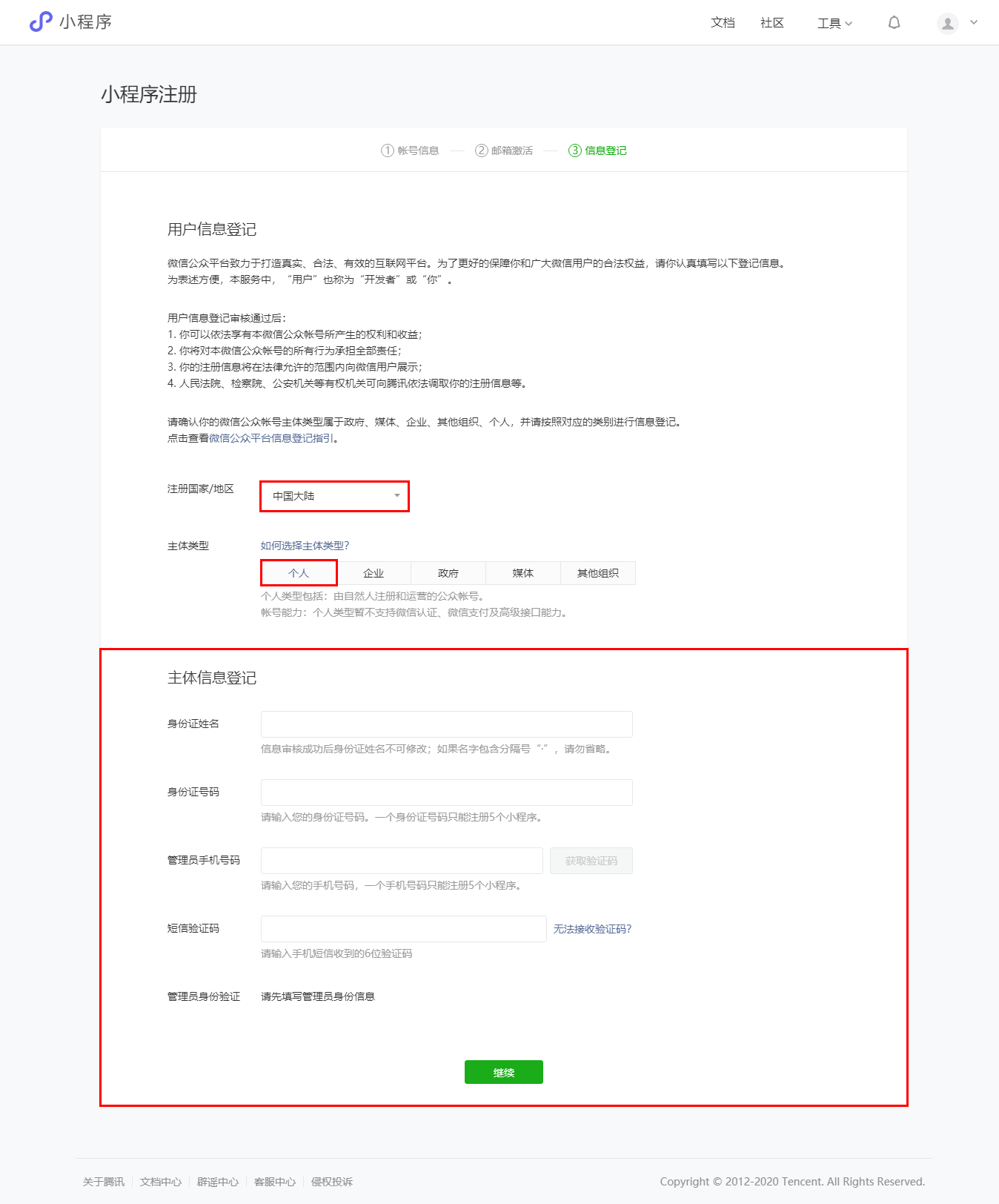


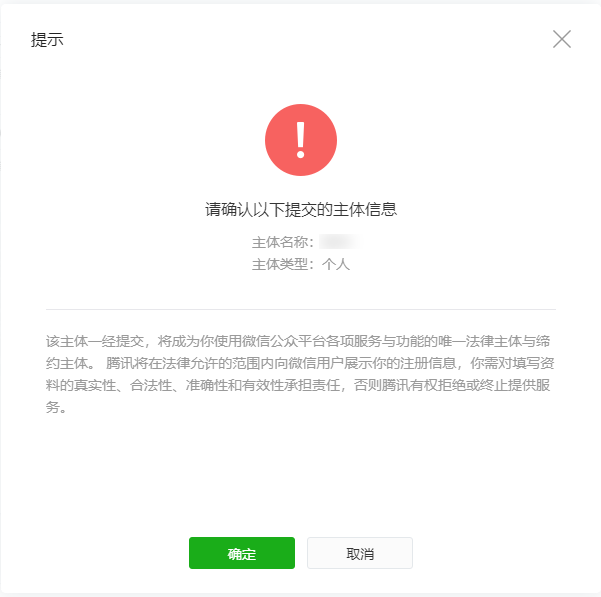
C.邮箱激活

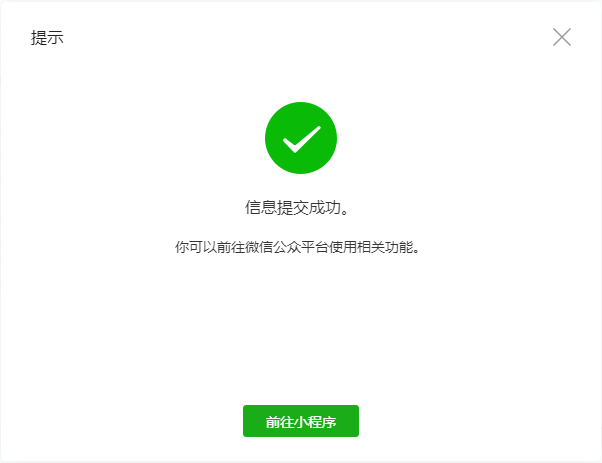




D.登记主体信息





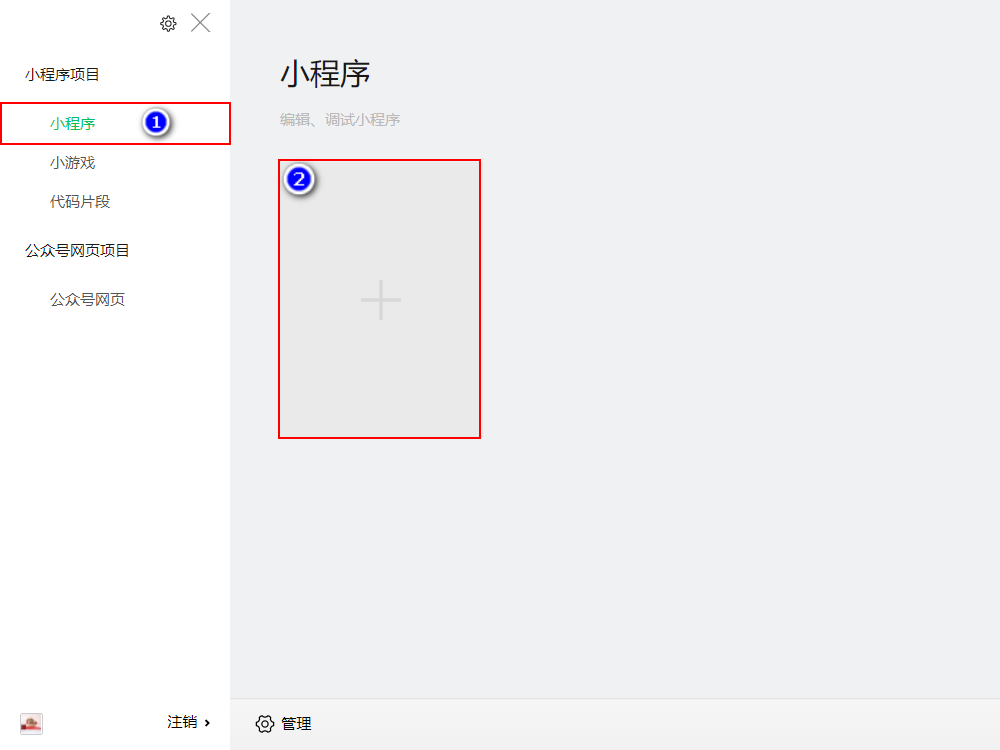


## 1.3 创建第一个小程序项目

A.下载并且安装开发者工具(https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html)

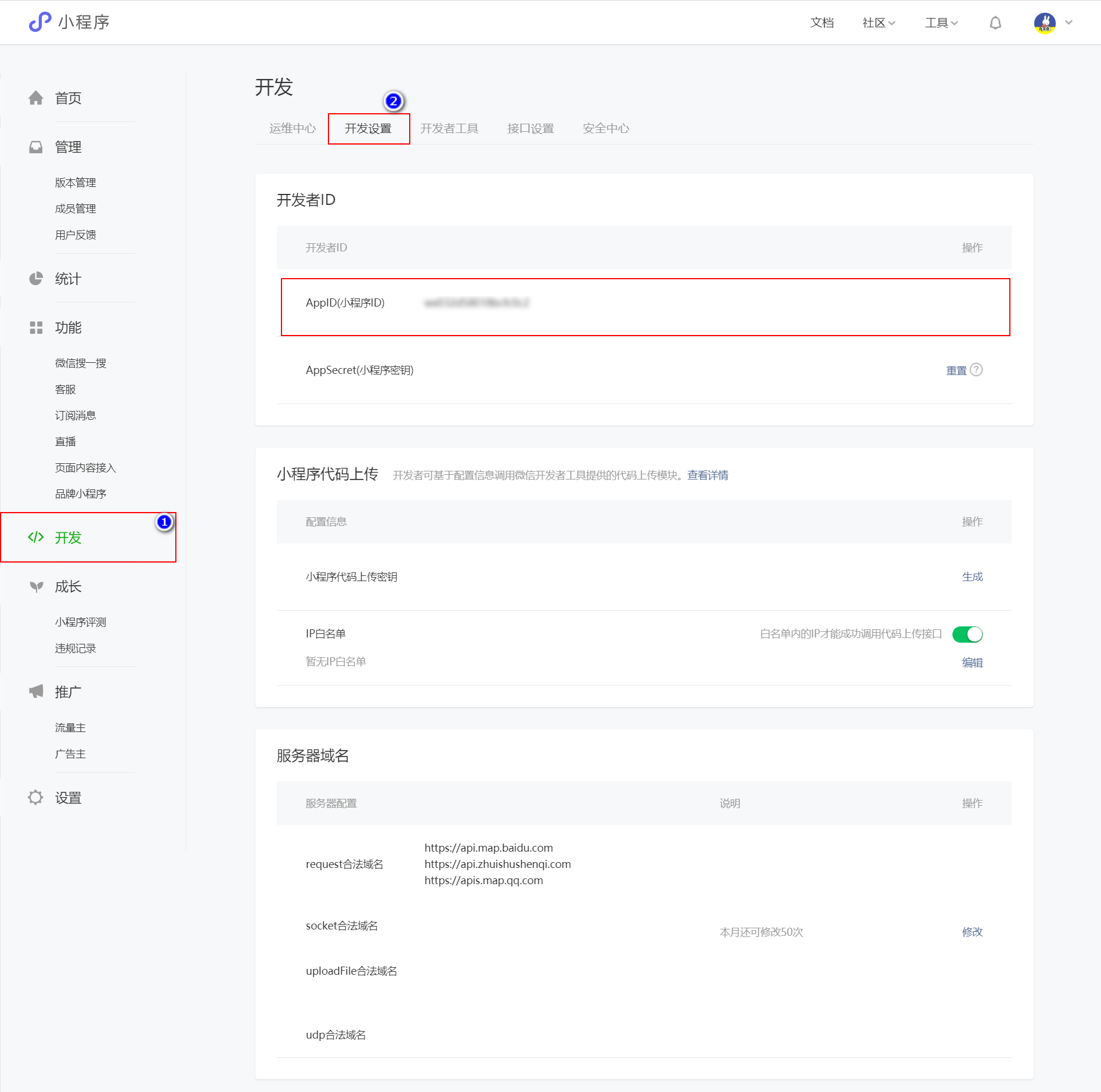
B.登录开发者工具

C.新建小程序项目

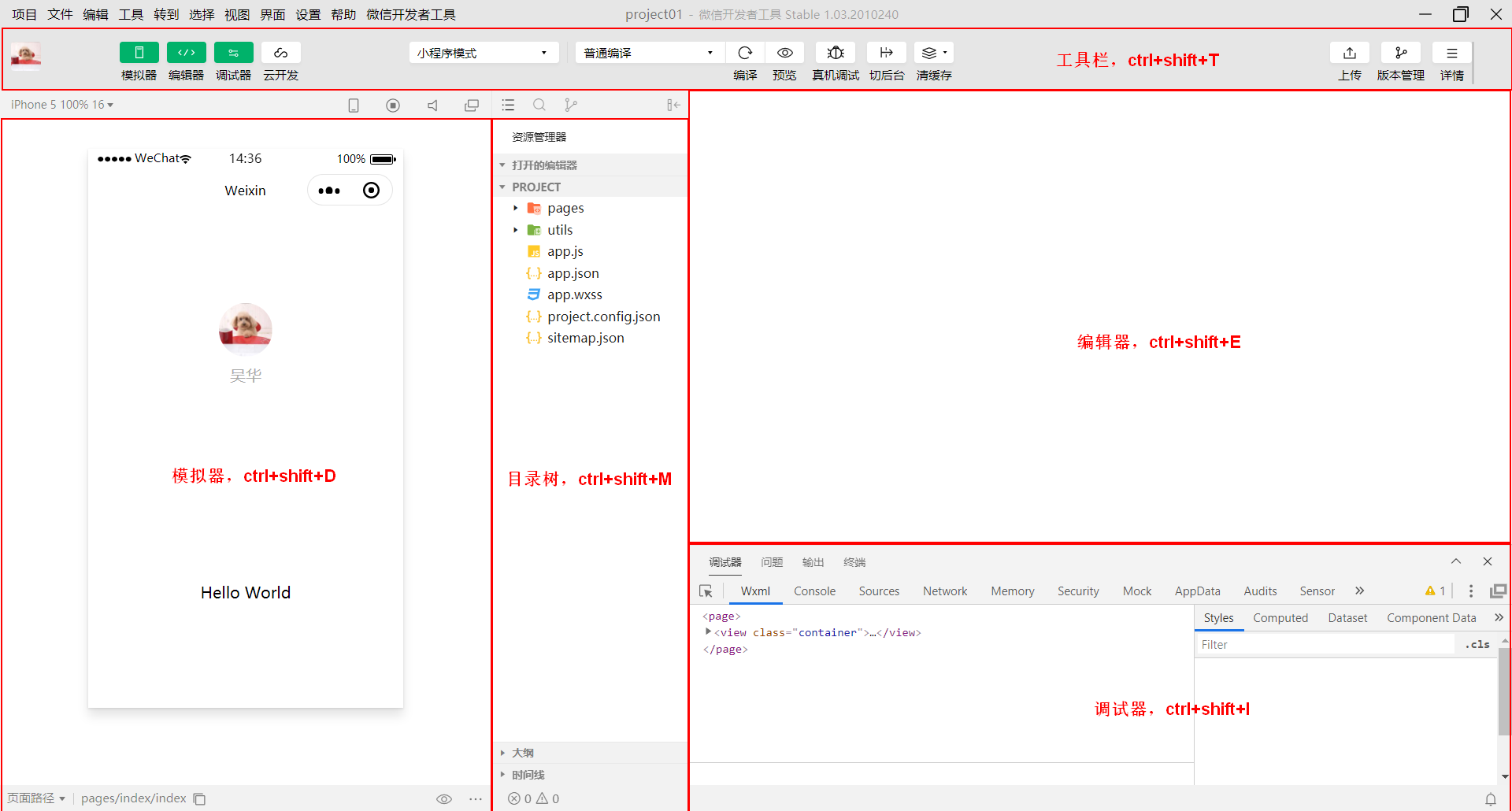




AppID在小程序开发者控制台->开发->开发设置->AppID



微信开发者工具的界面组成：



# 2.小程序的文件构成

小程序(MiniProgram)在不分包的情况下最大2MB(是所有小程序内所以资源(包括图片、音频、视频等)的字节数)；在分包的情况下最大20MB

小程序代码由以下文件类型构成：

· .json文件

.json文件是静态配置信息文件

app.json，全局配置信息文件

page.json，页面配置信息文件，页面配置优于全局配置。

· .wxml文件

.wxml文件在小程序开发中充当HTML角色，.wxml文件用于实现微信页面的结构。

.wxml文件中不能使用任何的HTML标签，只能使用小程序专用的标签。

· .wxss文件

.wxss文件在小程序开发中充当CSS角色，.wxss文件用于实现微信页面元素的修饰。

.wxss基本上与CSS相同

app.wxss，作为全局样式，作用于当前小程序所有页面。

page.wxss，作为页面样式，仅作用于当前页面，页面样式优于全局样式

· .js/.ts文件

.js/.ts文件是脚本文件，用于实现交互逻辑。

app.js文件是小程序的入口脚本文件

page.js文件是页面的脚本文件

app.json、app.wxss、app.js必须存储在小程序的根目录下！

# 3. app.json

app.json文件是全局配置信息文件，其必须存储在小程序的根目录内。

app.json中规定了所有页面的路径、界面表现、底部tabBar等信息。其结构形如：

{   
 "pages":[   
 "pages/index/index",   
 "pages/list/list",   
 "pages/me/me"   
 ]   
}

## 3.1 pages属性

pages属性用于指定小程序由哪些页面组成以及每个页面的存储位置，array类型

在默认情况下，数组的第一项代表小程序的初始页面(首页)

## 3.2 window属性

window属性用于设置小程序的导航栏信息、窗口背景颜色，object类型

· navigationBarTitleText属性用于设置导航栏标题文本，string类型

· navigationBarBackgroundColor属性用于设置导航栏背景颜色，string类型

· navigationBarTextStyle属性用于设置导航栏标题文本颜色，string类型 ，其值只能为black或white

· enablePullDownRefresh属性用于设置是否开启下拉刷新，boolean类型

· backgroundTextStyle属性用于设置下拉loading的样式，string类型，其值只能为light或dark

· backgroundColor属性用于设置窗口的背景颜色，string类型

## 3.3 tabBar属性

tabBar属性用于设置底部选项卡，object

tabBar至少包含两个、至多五个选项卡

tabBar的icon的字节数不能超过40KB

tabBar的icon尺寸建议81x81，且不支持网络图片

· color属性用于设置文本的默认颜色

· selectedColor属性用于设置选定时文本的颜色

· list属性用于设置tabBar的数量及链接路径，object array类型，每一个选项卡是object类型，包括：

· text属性用于设置底部选项卡的显示文本， string类型

· pagePath 属性用于设置底部选项卡的链接页面，string类型

必须保证页面已经在pages中存在

不能使用绝对路径

· iconPath属性用于设置正常显示的图标的路径，string类型

· selectedIconPath属性用于设置选定时显示的图标的路径，string类型

预习小程序的组件(view、image、navigator、form、input等)

https://developers.weixin.qq.com/miniprogram/dev/component/

微信小程序 - Unit02

# 1.WXSS

WXSS(Weixin Style Sheets)，是一套样式语言，用于描述WXML的组件样式。

WXSS具备CSS的大部分特性，但其进行扩展：

· 尺寸单位

· 样式导入

## 1.1 尺寸单位

rpx(Response Pixel)，响应式像素，可以根据屏幕宽度进行自适应，规定屏幕宽度为750rpx。

|  |  |  |
| --- | --- | --- |
| 设备 | rpx换算px (屏幕宽度/750) | px换算rpx (750/屏幕宽度) |
| iPhone5 | 1rpx = 0.42px | 1px = 2.34rpx |
| iPhone6 | 1rpx = 0.5px | 1px = 2rpx |
| iPhone6 Plus | 1rpx = 0.552px | 1px = 1.81rpx |

## 1.2 样式导入

样式导入的语句是：

@import "wxss文件的路径及名称";

# 2.组件

## · 关于组件(属性)的说明

A.组件属性既可以采用短横线分隔也可以采用驼峰标记法

B.针对布尔属性时，无论其值书写为任何结果都认为是true，除非该属性不存在(如果在进行双向数据绑定时，则需要true或 false)

C.小程序组件严格遵守XML语法结构，单标记必须以/>结尾

## · <view>组件

<view>组件是视图容器组件，其语法结构是：

<view   
 hover-class="按下去的样式类名称"   
 hover-start-time="按住后多久出现点击态(毫秒)"   
 hover-stay-time="点击态保留时间(毫秒)"   
 hover-stop-propagation="是否阻止冒泡">   
 ...   
</view>

## · <image>组件

<image>组件为图片组件，支持GIF、JPG、PNG、SVG、WEBP格式的图片，语法结构是：

<image   
 src="图片的URL地址"   
 lazy-load="是否采用懒加载"   
 show-menu-by-longpress="是否开启长按显示识别小程序码的菜单">   
</image>

webp图像格式是Google推荐使用的一种图像格式， 其体积约为JPG文件的2/3

## · <swiper>组件

<swiper>组件为滑块视图容器，其语法结构是：

<swiper   
 autoplay="是否自动播放"   
 indicator-dots="是否显示指示点"   
 indicator-color="指示点正常显示的颜色"   
 indicator-active-color="指示点选定时的颜色"   
 circular="是否采用衔接滑动"   
 interval="停留时间(毫秒)"   
 duration="切换的时长(毫秒)">   
 <swiper-item>...</swiper-item>   
 ...   
</swiper>

## · <text>组件

<text>组件是文本组件，其语法结构是：

<text decode="是否解码" user-select="文本是否可选">...</text>

解码支持实体(HTMLEntity)的有：&gt;、 &lt;、&amp;、&apos;、&nbsp;、&ensp;和&emsp;

## · <navigator>组件

<navigator>组件是页面链接组件，其语法结构是：

<navigator url="当前小程序内的页面路径" open-type="跳转方式">   
 ...   
</navigator>

跳转方式有：

· navigate，默认跳转方式，可以跳转到非tabBar页面，存在"返回"图标

· navigateBack，作用与"返回"图标是相同的，返回上一级页面

switchTab，可以跳转到tabBar页面

redirect 关闭当前页面，跳转到非tabBar页面

relaunch reLaunch 关闭所有页面(包括tabBar和非tabBar页面)

## · <button>组件

<button>组件为按钮组件，其语法结构是：

<button   
 type="按钮类型"   
 size="按钮尺寸"   
 plain="是否镂空"   
 disabled="是否禁用"   
 loading="是否带有加载图标">   
 ...   
</button>

按钮类型有：primary(主要)、default(默认的)、warn(警告)

按钮尺寸有：default(默认)、mini(小的)

# 3.事件绑定

事件是视图层到逻辑层的通讯方式，事件可以将用户的行为反馈到逻辑层进行处理。

## 1.1 事件列表

·touchstart，手指触屏开始

· touchmove，手指触屏后移动

· touchend，手指触屏动作结束

· touchcancel，手指触屏动作被打断，如来电提醒

· tap，轻敲（手指触屏后马上离开）

· longtap，长敲(手指触屏350ms后离开)

· longpress，长敲(手指触屏350ms后离开)

## 1.2 事件绑定

<组件名称 bind事件名称="函数名称"></组件名称>   
<组件名称 bind:事件名称="函数名称"></组件名称>   
<组件名称 catch事件名称="函数名称"></组件名称>

**小程序进行事件绑定时，函数名称严禁出现小括号！！！**

bind:事件名称不能应用于原生组件，而bind事件名称可应用于所有组件。

catch事件名称也不能应用于原生组件。

bind事件名称和bind:事件名称不能阻止事件冒泡；而catch事件名称可以阻止事件冒泡。

longpress不会触发tap事件；而longtap会触发tap事件，建议使用longpress取代longtap。

原生组件是手机操作系统自带的组件（键盘、照像机等）。

如：

<button bindtap="f1">按钮</button>   
   
<button bind:tap="f2">按钮</button>   
   
<button catchtap="f3">按钮</button>

Vue.js与小程序页面的结构对比

Vue.js

<script>   
 export default{   
 data:{   
 return {   
 //初始化数据   
 }   
 },   
 methods:{   
 //方法列表   
 }   
 }   
</script>

小程序页面

Page({   
 //初始化数据   
 data:{   
   
 },   
 //方法列表   
})

## 1.3 数据传递

当对组件进行事件绑定时，事件处理函数**严禁出现小括号**，形如：

<button type="primary" bindtap="f5">按钮</button>

现在的问题就是：如何将参数传递到自定义函数的内部？

针对**原生事件(如touchstart、tap等)**只能通过以两步实现：

第一步：首先组件上通过data-\*来实现自定义属性

<button   
 type="primary"   
 bindtap="f1"   
 data-username="张三">   
 原生组件传参   
</button>

第二步：在事件处理函数中传入event参数，代表事件对象，然后在函数体内部通过：event.target.dataset.\*进行获取操作

Page({   
 f1(event){   
 let username = event.target.dataset.username;   
 console.log(username);   
 }   
})

# 微信小程序 - Unit03

# 1.组件

## · <input>组件

<input>组件为输入框组件，该原生是原生组件，其语法结构是：

<input   
 type="输入框的类型"   
 placeholder="占符位"   
 password="是否为密码"   
 value="初始内容"   
 maxlength="最大长度"   
 focus="是否自动获得焦点"   
 bindinput="输入时触发的函数"   
 bindfocus="获得焦点时触发的函数"   
 bindblur="失去焦点时触发的函数"   
 bindconfirm="单击完成时触发的函数">   
</input>

输入框的类型包括：

· text，文本输入框

· number，数字输入框

· digit，带有小数点的数字输入框

· idcard，身份证号输入框

原生组件：由手机操作系统自带的组件，其特点有：

· 无论其他组件的z-index设置多大，都无法覆盖原生组件

· 部分样式(如position:fixed)无法应用于原生组件

· 事件绑定只支持 bind事件名称 的方法

原生组件有：

· input -- 仅在focus时表现为原生组件

· camera

· video等

扩展：

表单元素的value属性在编辑时"必须"进行指定，因为需要显示的用户原始填写的相关信息。

## · <checkbox-group>组件

<checkbox-group>组件为复选框群组组件，其语法结构是：

<checkbox-group bindchange="选定项目发生改变时触发的函数">   
 <checkbox value="值" checked="是否被选定">...</checkbox>   
</checkbox-group>

## · <radio-group>组件

<radio-group>组件为单选框群组组件，其语法结构是

<radio-group bindchange="选定项目发生改变时触发的函数">   
 <radio value="值" checked="是否被选定">...</radio>   
</radio-group>

## · <switch>组件

<switch>组件是开关组件，其语法结构是：

<switch   
 checked="是否被选定"   
 type="类型"   
 bindchange="状态发生改变时触发的函数">   
 ...   
</switch>

类型包括：switch和checkbox

## · <picker>组件

<picker>组件是从底部弹出的选择器

### ◎selector类型

selector指普通的选择器，其语法结构是：

<picker   
 mode="selector"   
 range="选择的范围(array类型)"   
 bindchange="选项发生变化时触发函数"   
 bindcancel="取消选择时触发的函数">   
</picker>

### ◎region类型

# 2.组件自定义事件

组件自定义事件是指除原生事件(如touchstart、touchmove、tap等)外的事件。

无论组件绑定的事件是原生事件还是自定义事件，在事件处理函数中一定不能使用小括号，如：

<button type="primary" bindtap="f1">按钮</button>   
   
<input type="text" bindinput="f2"></input>

如果原生事件的处理函数中需要参数时，可通过自定义属性进行传递。

如果在自定义事件的处理函数中需要获取当前组件的相关数据怎么办?

直接在事件处理函数中带有event 参数(代表事件对象)，然后通过event.detail.value进行获取，如:

Page({   
 f1(event){   
 console.log(event.detail.value)   
 }   
})

# 3.WXML

## 3.1 内容绑定

WXML 中的数据来源于逻辑层中的data

Page({   
 //初始化数据   
 data:{   
   
 }   
})

内容绑定直接通过双花括号语法进行输出即可，如

<view>姓名:{{username}}</view>

内容绑定：Vue.js VS 小程序

Vue.js

<div>{{username}}</div>

小程序

<view>{{username}}</view>

属性绑定：Vue.js VS 小程序

Vue.js

<img :src="imageURL">

小程序

<image src="{{imageURL}}"/>

CSS类的绑定：Vue.js VS 小程序

Vue.js

<div :class="className">动态CSS类</div>

小程序

<view class="{{className}}">动态CSS类</view>

内联样式的绑定：Vue.js VS 小程序

Vue.js

<div :style="inlineStyle">动态行内样式</div>

小程序

<view style="{{inlineStyle}}">动态行内样式</view>

## 3.2 列表渲染

<组件名称   
 wx:for="{{变量名称}}"   
 wx:for-item="成员变量名称"   
 wx:for-index="下标变量名称"   
 wx:key="成员下标变量名称">   
 ...   
</组件名称>

默认的成员变量名称是item  
默认的成员下标变量名称是index

列表渲染 --Vue.js VS 小程序

Vue.js

<div v-for="(item,index) of books" :key="index">   
 ...   
</div>

小程序

<view   
 wx:for="{{books}}"   
 wx:for-item="item"   
 wx:for-index="index"   
 wx:key="index">   
 ....   
</view>

## 3.3 条件渲染

<组件名称 wx:if="{{条件表达式}}">   
 ...   
</组件名称>

<组件名称 wx:if="{{条件表达式}}">   
 ...   
</组件名称>   
<组件名称 wx:else>   
 ...   
</组件名称>

<组件名称 wx:if="{{条件表达式}}">   
 ...   
</组件名称>   
<组件名称 wx:elif="{{条件表达式}}">   
 ...   
</组件名称>   
<组件名称 wx:elif="{{条件表达式}}">   
 ...   
</组件名称>   
...   
<组件名称 wx:else>   
 ...   
</组件名称>

条件渲染 -- Vue.js VS 小程序

Vue.js

<div v-if="条件表达式">   
 ...   
</div>   
<div v-else-if="条件表达式">   
 ...   
</div>   
<div v-else-if="条件表达式">   
 ...   
</div>   
<div v-else>   
 ...   
</div>

小程序

<view wx:if="{{条件表达式}}">   
 ...   
</view>   
<view wx:elif="{{条件表达式}}">   
 ...   
</view>   
<view wx:elif="{{条件表达式}}">   
 ...   
</view>   
<view wx:else>   
 ...   
</view>

<block>...</block>标签仅是一个包装元素的标签，其不会在页面中进行渲染操作，只接收控制属性

# 4.修改data中的变量值

在小程序的页面中，要修改data中的变量值的话，只能通过Page对象的setData()方法来实现，其语法结构是：

this.setData({   
 ...   
}[,callback])

this 关键字指当前的页面对象 -- Page

关于JSON:https://www.json.org/json-zh.html

# 5. API

## · wx.showToast()方法

wx.showToast()方法用于显示短消息提示框，其语法结构是：

wx.showToast({   
 title:"提示文本",   
 icon:"图标类型(success|loading|none)",   
 image:"本地图标路径",   
 duration:延迟时长(1500ms),   
 mask:是否显示透度的遮罩层   
})

## · wx.hideToast()方法

wx.hideToast()方法用于隐藏消息提示框，其语法结构是：

wx.hideToast({   
 success:res=>{   
 //...   
 }   
})

## · wx.showModal()方法

wx.showModal()方法用于显示模态对话框 ，其语法结构是：

wx.showModal({   
 title:"提示标题",   
 content:"提示内容",   
 showCancel:是否显示取消按钮,   
 confirmText:"确认按钮的文本",   
 cancelText:"取消按钮的文本",   
 success:res=>{   
 //res.confirm代表是否单击了确认按钮   
 //res.cancel代表是否单击了取消按钮   
 }   
})

## · wx.showLoading()方法

wx.showLoading()方法用于显示加载提示框，其语法结构是：

wx.showLoading({   
 title:"提示的内容",   
 mask:是否显示透明的遮罩层,   
 success:res=>{   
 //...   
 }   
})

## · wx.hideLoading()方法

wx.hideLoading()方法用于隐藏加载提示框，其语法结构是：

wx.hideLoading({   
 success:res=>{   
 //...   
 }   
})

# 微信小程序 - Unit04

# 1.小程序的生命周期

小程序的生命周期由小程序的生命周期和页面的生命周期来构成。

## 1.1 小程序的生命周期

小程序的生命周期存储在app.js中。

App函数用于注册小程序，接收object类型的参数，在其中可以指定小程序的生命周期回调、全局变量等。

App({...})

### · onLaunch()函数

onLaunch()函数在小程序初始化完成时自动触发，全局只触发一次，其语法结构是：

App({   
 onLaunch(){   
 //...   
 }   
})

可以在onLaunch()函数内完成用户是否登录的检测以及获取用户信息等操作。

### ·onShow()

onShow()方法在小程序启动或小程序从后台切入到前台时触发，其语法结构是：

App({   
 onShow(){   
 //...   
 }   
})

### ·onHide()

onHide()方法在小程序从前台切入到后台时触发，其语法结构是：

App({   
 onHide(){   
 //...   
 }   
})

## 1.2 页面的生命周期

页面的生命周期存储在page.js中。

Page()函数用于注册页面，其中包含在当前页面动态的数据 -- data、生命周期回调、页面的事件等。

### · onLoad(options)

onLoad(options)函数在页面加载时触发，一个页面只触发一次，其语法结构是：

Page({   
   
 onLoad(options){   
 //...   
 }   
})

options参数指由链接的URL地址栏参数所形成的对象！

### · onShow()

onShow()函数在页面显示/切入前台时触发，其语法结构是：

Page({   
 onShow(){   
 //...   
 }   
})

### · onReady()

onReady()函数在页面初次渲染完成时触发，一个页面只会触发一次，代表页面已经准备妥当，可以与视图层进行交互，其语法结构是：

Page({   
 onReady(){   
 //...   
 }   
})

小程序的生命周期回调中，先触发onShow() 即先显示静态的WXML页面，再触发onReady()，此时可以与视图层进行交互，这样做的目的是为了防止用户长时间面对空白页面。

### · onHide()

onHide()函数在页面隐藏或小程序切入到后台时触发，其语法结构是：

App({   
 onHide(){   
 //...   
 }   
})

小程序在切入后台时，先触发页面onHide()，再次小程序的onHide()

小程序在切入前台时，先触发小程序的onShow()，再触发页面的onShow()

### · onUnload()

onUnload()函数在页面卸载时触发，其语法结构是：

App({   
 onUnload(){   
 //...   
 }   
})

# 2.页面跳转

在小程序中，实现页面跳转的方式有两种：

A.通过<navigator>组件实现

B.通过相关的API实现

## 2.1 <navigator>

<navigator>组件为链接组件，其语法结构是：

<navigator url="当前小程序内的页面地址" open-type="跳转方式">   
...   
</navigator>

跳转方式有：

· navigate

称为保留跳转，只能跳转到非tabBar页面

触发当前页面的onHide()

可以带有URL地址栏参数

· navigateBack

称为关闭当前页面跳转

触发当前页面的onUnload()

· switchTab

称为关闭所有非tabBar页面

触发所有页面的onUnload()

不能带有URL参数

· redirect

称为关闭当前页面，跳转到非tabBar页面

触发当前页面的onUnload()

可以带有 URL参数

· reLaunch

称为关闭所有页面

触发所有页面的onUnload()

可以带有URL参数

## 2.2 API

### · wx.navigateTo()

wx.navigateTo()方法的作用等同于<navigator open-type="navigate">，其语法结构是：

wx.navigateTo({   
 url:'当前小程序内的非tabBar页面路径,可带有参数',   
 success:res=>{   
 //...   
 }   
 fail:error=>{   
 //...   
 },   
 complete:res=>{   
 //...   
 }   
});

### · wx.navigateBack()

wx.navigateBack()方法等同于<navigator open-type="navigateBack">，其语法结构是：

wx.navigateBack({   
 delta:层级数   
})

### · wx.switchTab()

wx.switchTab({   
 url:'当前小程序内的tabBar页面路径',   
 success:res=>{   
 //...   
 }   
 fail:error=>{   
 //...   
 },   
 complete:res=>{   
 //...   
 }   
});

### · wx.redirectTo()

wx.redirectTo({   
 url:'当前小程序内的非tabBar页面路径,可带有参数',   
 success:res=>{   
 //...   
 }   
 fail:error=>{   
 //...   
 },   
 complete:res=>{   
 //...   
 }   
});

### · wx.reLaunch()

wx.reLaunch({   
 url:'当前小程序内的页面路径,可带有参数',   
 success:res=>{   
 //...   
 }   
 fail:error=>{   
 //...   
 },   
 complete:res=>{   
 //...   
 }   
});

navigate和redirect只能跳转到非tabBar页面

switchTab只能跳转到tabBar页面

reLaunch可以跳转到任意页面

navigate、redirect、reLaunch可以带有URL参数

switchTab不能带有URL参数

redirect关闭当前页面；switchTab关闭所有非tabBar页面；reLaunch关闭所有页面

# 3.获取服务器数据

## 3.1 服务器域名的限制

A.域名只支持 HTTPS协议

B.域名必须经过ICP备案

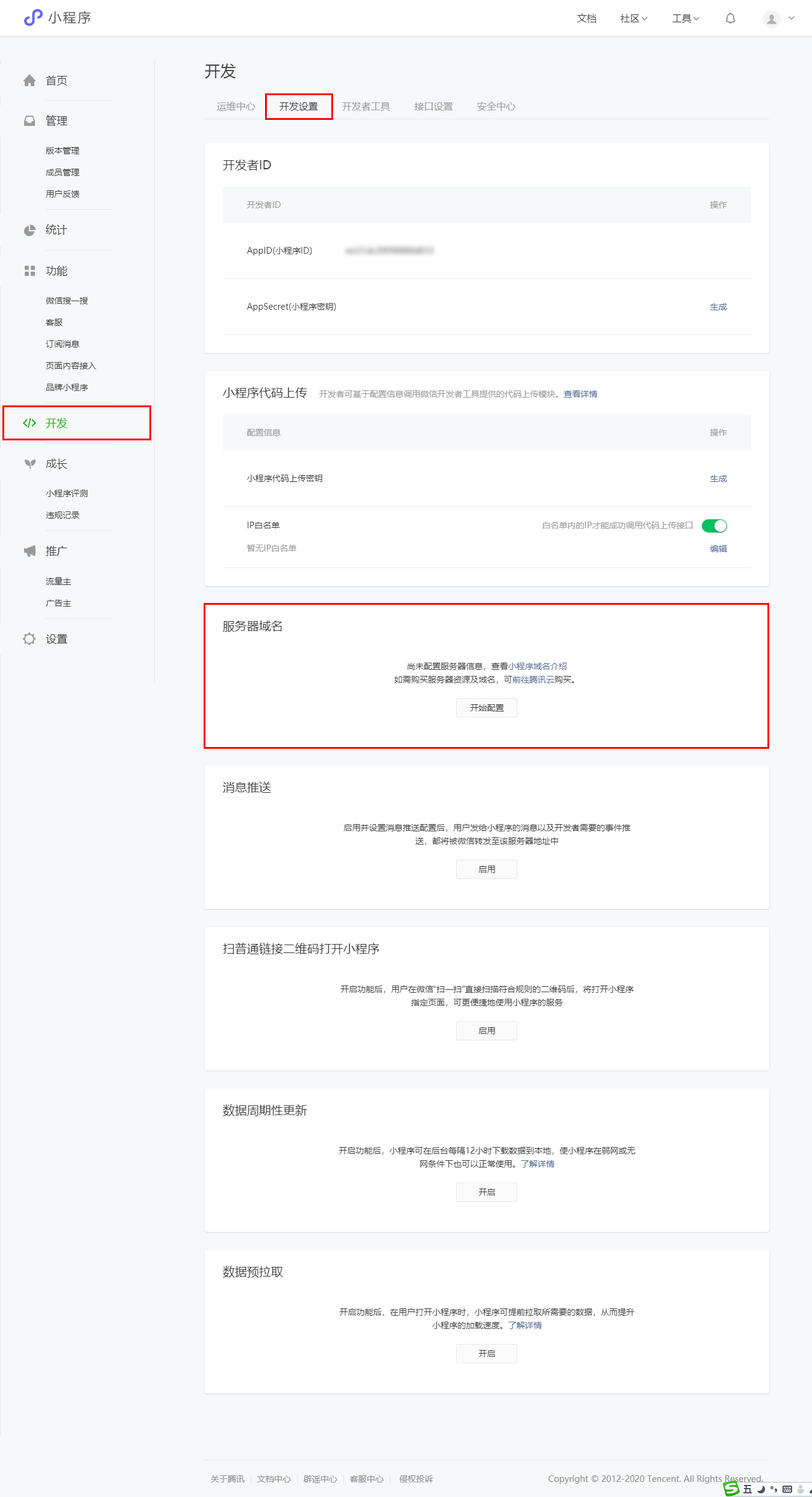
C.域名不能使用IP地址

D.不支持配置父域名使用子域名

E.对于每个接口最多可以配置20个域名

## 3.2 服务器域名的配置

小程序后台-开发-开发设置-服务器域名







## 3.3 wx.request()

wx.request()方法用于发起 HTTPS 网络请求，其语法结构是：

wx.request({   
 url:"服务器API接口地址",   
 data:"请求参数,object类型",   
 method:"请求方式"，   
 success:res=>{   
 //...   
 }   
})

# 微信小程序 - Unit05

# 1.云开发

## 1.1 概述

开发者可以使用云开发来开发微信小程序、小游戏，而无需搭建服务器。

云开发提供的能力：

· 云存储，类似网盘，在小程序客户端可以实现文件上传/下载的能力。

· 云数据库，是一个JSON类型的数据，实现数据的增、删、改、查等操作。

· 云函数，在云端运行的代码。

## 1.2 开通云开发

单击开发者工具栏上的"云开发"按钮

## 1.3 创建云开发小程序

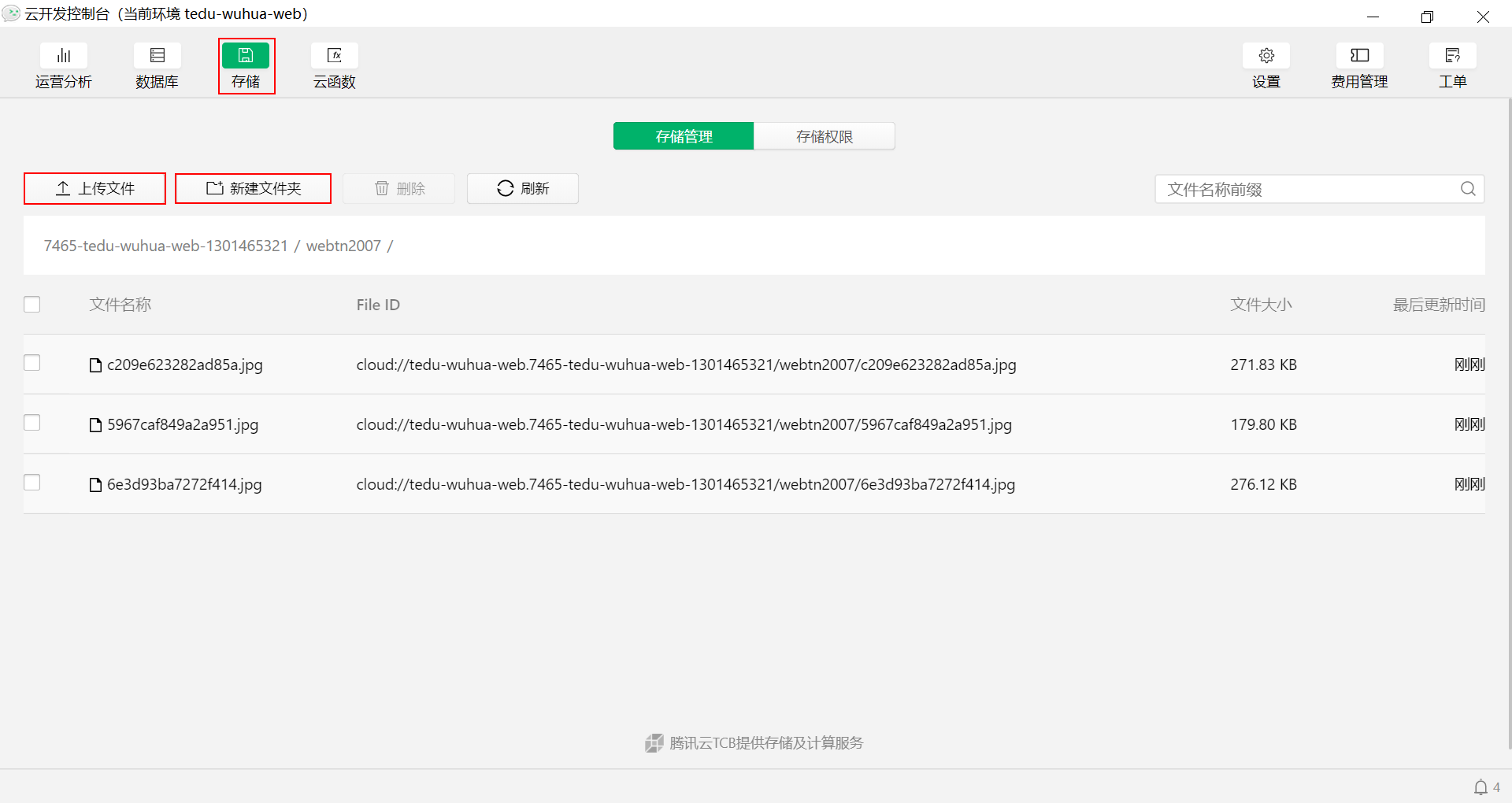


在创建云开发的小程序项目后，miniprogram才是当前项目的根目录。

# 2.云存储

云存储的管理可以通过图形化界面或API来实现。

## 2.1 云开发控制台管理界面



## 2.2 API

### · wx.cloud.init()

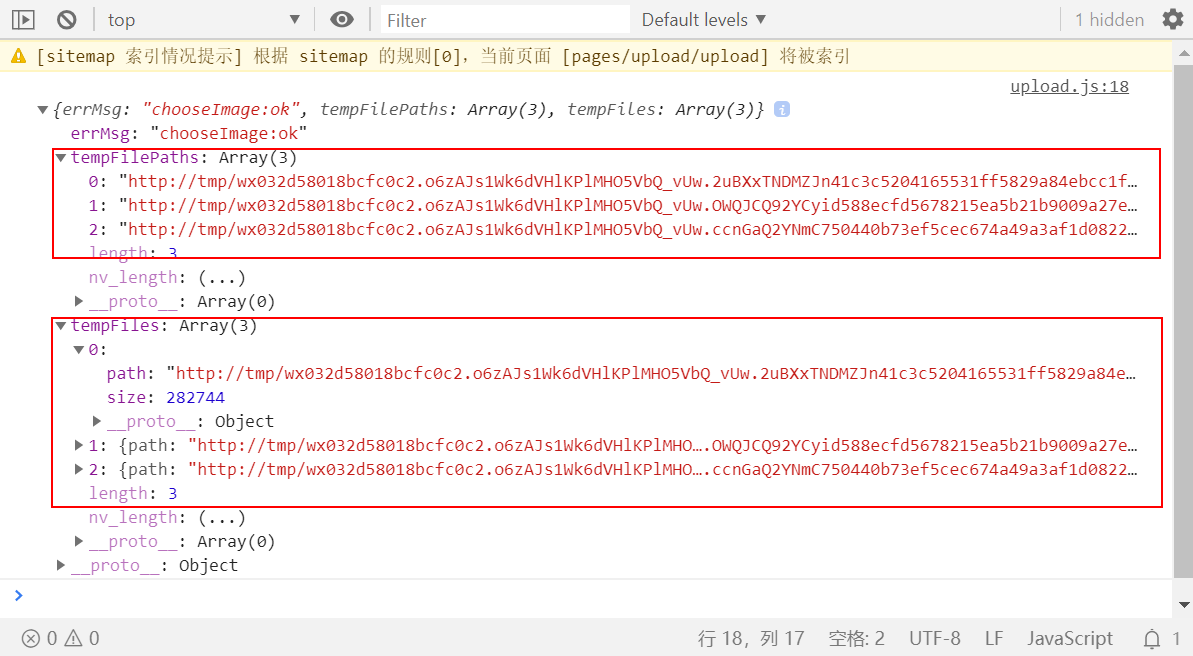
wx.cloud.init()方法用于实现云环境的初始化，其语法结构是：

wx.cloud.init({   
 env:"云环境ID",   
 traceUser:是否跟踪用户   
})

### · wx.chooseImage()

wx.chooseImage()方法用于从本地相册选择图片或使用相机拍照，其语法结构是：

wx.chooseImage({   
 count:最多可选择的图片数量(9),   
 //album,相册;camera,相机   
 sourceType:图片来源["album","camera"],   
 success:res=>{   
 //res.tempFilePaths,为字符串数组,用于存储临时文件的路径   
 //res.tempFiles,为对象数组，用于存储临时文件的信息   
 //每个临时文件包括path及size属性   
 }   
})



### · wx.previewImage()

wx.previewImage()方法用于全屏预览图片，其语法结构是：

wx.previewImage({   
 urls:"需要预览的图表列表(字符串数组类型)",   
 current:"当前显示的图片的地址(字符串类型)",   
 success:res=>{   
 //...   
 }   
})

### · wx.uploadFile()

wx.uploadFile()方法用于将本地文件上传到HTTPS服务器，其语法结构是：

wx.uploadFile({   
 filePath:"本地文件路径",   
 url:"服务器URL地址",   
 name:"文件名称key",   
 success:res=>{   
 //...   
 }   
})

### · wx.cloud.uploadFile()

wx.cloud.uploadFile()方法用于将本地文件上传到云存储空间，其语法结构是：

wx.cloud.uploadFile({   
 filePath:"本地文件路径",   
 cloudPath:"云存储路径",   
 config:{   
 env:"云环境ID"   
 },   
 success:res=>{   
 //res.fileID返回文件上传成功后的fileID   
 //该fileID可以被使用于<image>的src属性中   
 }   
})

# 3.云数据库

## 3.1 概述

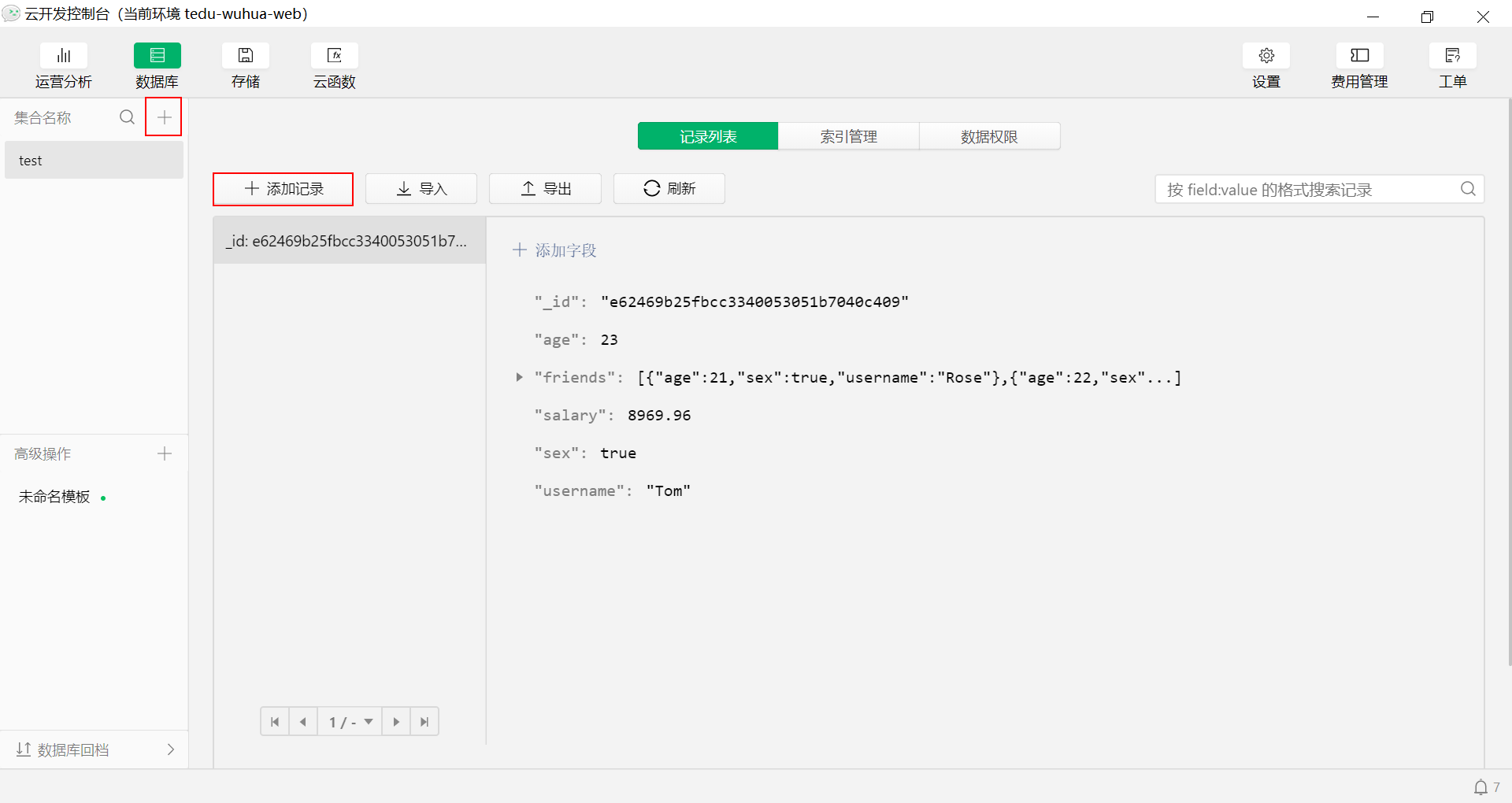
云数据库是一个既可以在小程序客户端，也可以在云函数中读写的JSON类型的数据库。

小程序提供的云数据库是基于MongoDB 的非关系型数据库。

## 3.2 关系型数据库与非关系型数据库的区别

|  |  |  |
| --- | --- | --- |
|  | 关系型数据库 | 非关系型数据库 |
| 核心概念 | 数据库、数据表、列(字段)、行(记录) | 数据库、集合、字段、记录(文档) |
| 数据类型 | 字符型、整型、浮点型、布尔型 | 字符型、数值型、布尔型、数组、对象 |
| 操作命令 | SQL | 不尽相同 |
| 关系的维护 | 外键 | - |
| 典型的代表 | MySQL、MS SQL Server等 | MongoDB、Redis |

## 3.3 创建集合



## 3.4 API

### · wx.cloud.database()

wx.cloud.database()方法用于获取数据库对象实例，其语法结构是：

wx.cloud.database({   
 env:"云环境ID"   
})

wx.cloud.database()方法可以想像成USE 数据库名称

### · collection()

collection()用于获取集合对象实例，其语法结构是：

Database.collection(string name)

### · add()

add()方法用于添加文档(记录)，其语法结构是：

Collection.add({   
 data:{   
 //要插入的数据,数据类型包括：string、number、boolean、array、object等   
 },   
 success:res=>{   
 //res.\_id将返回新插入记录的\_id号   
 }   
})

\_id为系统自动生成的唯一的ID

\_openid指当前文档(记录)所有者的OpenID

课外作业：

1.手动创建两个集合 -- category、books，分别用来存储图书的分类及图书信息

2.将上传图片的fileID写入到test集合