腾讯云小程序平台

开发指南

腾讯云

文档目录

开发指南
小程序介绍与开发环境

最近更新时间：2024-04-11 10:00:42

小程序是一种全新的连接用户与服务的方式，它可以在应用内被便捷地获取和传播，同时具有出色的使用体验。任何一个普通的开发者，经过简单的学习和练习后，都可以轻松地完成一个小程序的开发和发布。

1、Hello World

在详细介绍小程序历史和技术细节前，请先跟随我们的步骤完成开发 Hello World 例子。

- 第一步，请前往 IDE 操作说明 页面根据自己的操作系统下载对应的安装包进行安装。
- 第二步，打开 TCMPP 开发者工具，选择新建小程序项目，新建项目时选择无 AppID，并取消勾选“建立普通快速启动模板”的选项。
- 最后一步，我们来添加必要的代码。

在根目录下创建 app.json，其内容如下。

```
{

 "pages": ["pages/index/index"]

}
```

在根目录下新建 pages 目录，然后在 pages 目录下新建 index 目录，接着在 index 目录下创建两个文件 index.wxml 和 index.js。
index.wxml 的内容如下所示。 `<text>Hello World</text>`
index.js 的内容如下所示。 `Page({})`
通过编写以上短短的几行代码，TCMPP 开发者工具的模拟器界面上显示出 Hello World。

2、小程序介绍

2.1 小程序技术发展历史

从技术的维度看，小程序并非凭空冒出来的一个概念。当 WebView 逐渐成为移动 Web 的一个重要入口时，JS API 也应运而生了，如下代码所示：

```
WeixinJSBridge.invoke('imagePreview', {
   current: 'http://inews.gtimg.com/newsapp_bt/0/1693121381/641',
   urls: [ // 所有图片的URL列表，数组格式
      'https://img1.gtimg.com/10/1048/104857/10485731_980x1200_0.jpg',
      'https://img1.gtimg.com/10/1048/104857/10485726_980x1200_0.jpg',
```

```
        'https://img1.gtimg.com/10/1048/104857/10485729_980x1200_0.jpg'
    ]
}, function(res) {
    console.log(res.err_msg)
})
```

这是一个调用原生组件浏览图片的 JS API，相比于额外引入一个 JS 图片预览组件库，这种调用方式显得非常简洁和高效。
实际上，微信官方是没有对外暴露过如此调用的，此类 API 最初是提供给腾讯内部一些业务使用，很多外部开发者发现了之后，依葫芦画瓢地使用了，逐渐成为微信中网页的事实标准。
平台内置了一整套网页开发工具包，称之为 JS-SDK，开放了拍摄、录音、语音识别、二维码、地图、支付、分享、卡券等几十个API。给所有的 Web 开发者打开了一扇全新的窗户，让所有开发者都可以使用到宿主应用内的原生能力，去完成一些之前做不到或者难以做到的事情了。
同样是调用原生的浏览图片，调用方式如下代码所示：

```
wx.previewImage({
  current: 'https://img1.gtimg.com/10/1048/104857/10485726_980x1200_0.jpg',
  urls: [ // 所有图片的URL列表，数组格式
    'https://img1.gtimg.com/10/1048/104857/10485731_980x1200_0.jpg',
    'https://img1.gtimg.com/10/1048/104857/10485726_980x1200_0.jpg',
    'https://img1.gtimg.com/10/1048/104857/10485729_980x1200_0.jpg'
  ],
  success: function(res) {
    console.log(res)
  }
})
```

JS-SDK 解决了移动网页能力不足的问题，通过暴露宿主应用的接口使得 Web 开发者能够拥有更多的能力，然而在更多的能力之外，JS-SDK 的模式并没有解决使用移动网页遇到的体验不良的问题。
用户在访问网页的时候，在浏览器开始显示之前都会有一个的白屏过程，在移动端，受限于设备性能和网络速度，白屏会更加明显。我们团队把很多技术精力放置在如何帮助平台上的 Web 开发者解决这个问题。因此我们设计了一个 JS-SDK 的增强版本，其中有一个重要的功能，称之为“Web 资源离线存储”。
该设计类似 HTML5 的 Application Cache，但设计上规避了一些 Application Cache 的不足。
在内部测试中，我们发现离线存储能够解决了一些问题，但是对于一些复杂的页面依然会有白屏的问题，例如页面加载了大量的 CSS 或者是 JavaScript 文件，这些文件的执行时间占用了大量的 UI 线程，这种时候，即使通过离线存储快速的加载资源，但是依旧会有页面的白屏现象，同时这样分文件的 Cache 在处理代码文件更新的时候操作较为繁杂，对开发者的要求较高。
除了白屏，影响 Web 体验的问题还有缺少操作的反馈，主要表现在两个方面：页面切换的生硬和单击的迟滞感。
对于一些有经验的 Web 开发者而言，会使用一些 SPA 的框架，来模拟客户端原生的页面切换过渡。通常的方式是在一个 WebView 中去模拟多个页面，通过 CSS 处理，加之精细化的脚本代码做到单击反馈和页面切换，获得较好的体验。然而并不是所有的开发者都有足够的时间和精力来使得页面的体验变得出色。

宿主应用随之面临的问题是如何设计一个比较好的系统，使得所有开发者在宿主应用中都能获得比较好的体验。这个问题是之前的 JS-SDK 所处理不了的，需要一个全新的系统来完成，它需要使得所有的开发者都能做到：

- 快速的加载
- 更强大的能力
- 原生的体验
- 易用且安全的数据开放
- 高效和简单的开发

这一系统就是本书中需要详细阐述的小程序。

2.2 小程序与普通网页开发的区别

小程序的主要开发语言是 JavaScript，所以通常小程序的开发会被用来同普通的网页开发来做对比。两者有很大的相似性，对于前端开发者而言，从网页开发迁移到小程序的开发成本并不高，但是二者还是有些许区别的。
网页开发渲染线程和脚本线程是互斥的，这也是为什么长时间的脚本运行可能会导致页面失去响应，而在小程序中，二者是分开的，分别运行在不同的线程中。网页开发者可以使用到各种浏览器暴露出来的 DOM API，进行 DOM 选中和操作。而如上文所述，小程序的逻辑层和渲染层是分开的，逻辑层运行在 JSCore 中，并没有一个完整浏览器对象，因而缺少相关的 DOM API 和 BOM API。这一区别导致了前端开发非常熟悉的一些库，例如 jQuery、Zepto 等，在小程序中是无法运行的。同时 JSCore 的环境同 NodeJS 环境也是不尽相同，所以一些 NPM 的包在小程序中也是无法运行的。
网页开发者需要面对的环境是各式各样的浏览器，PC 端需要面对 IE、Chrome、QQ浏览器等，在移动端需要面对 Safari、Chrome 以及 iOS、Android 系统中的各式 WebView 。而小程序开发过程中需要面对的是两大操作系统 iOS 和 Android 的宿主应用客户端，以及用于辅助开发的小程序开发者工具，小程序中三大运行环境也是有所区别的，如下表所示。

运行环境	逻辑层	渲染层
iOS	JavaScriptCore	WKWebView
安卓	X5 JSCore	X5浏览器
小程序开发者工具	NWJS	Chrome WebView

网页开发者在开发网页的时候，只需要使用到浏览器，并且搭配上一些辅助工具或者编辑器即可。小程序的开发则有所不同，需要经过申请小程序账号、安装小程序开发者工具、配置项目等过程方可完成。

3、小程序的特色

对于普通用户，小程序实现了应用的触手可及，只需要通过扫描二维码、搜索或者是朋友的分享就可以直接打开，加上优秀的体验，小程序使得服务提供者的触达能力变得更强。
对于开发者而言，小程序框架本身所具有的快速加载和快速渲染能力，加之配套的云能力、运维能力和数据汇总能力，使得开发者不需要去处理琐碎的工作，可以把精力放置在具体的业务逻辑的开发上。

4、小程序开发准备

开发者可以在平台创建小程序后会自动生成对应的 App ID，并且提供不同版本的 IDE 下载地址，开发者可以自行选择合适版本进行下载，具体可参考 小程序管理 和 无代码构建小程序 。

小程序代码组成
JSON 配置

最近更新时间：2024-04-10 15:12:32

JSON 是一种数据格式，并不是编程语言，在小程序中，JSON扮演的静态配置的角色。

一个例子

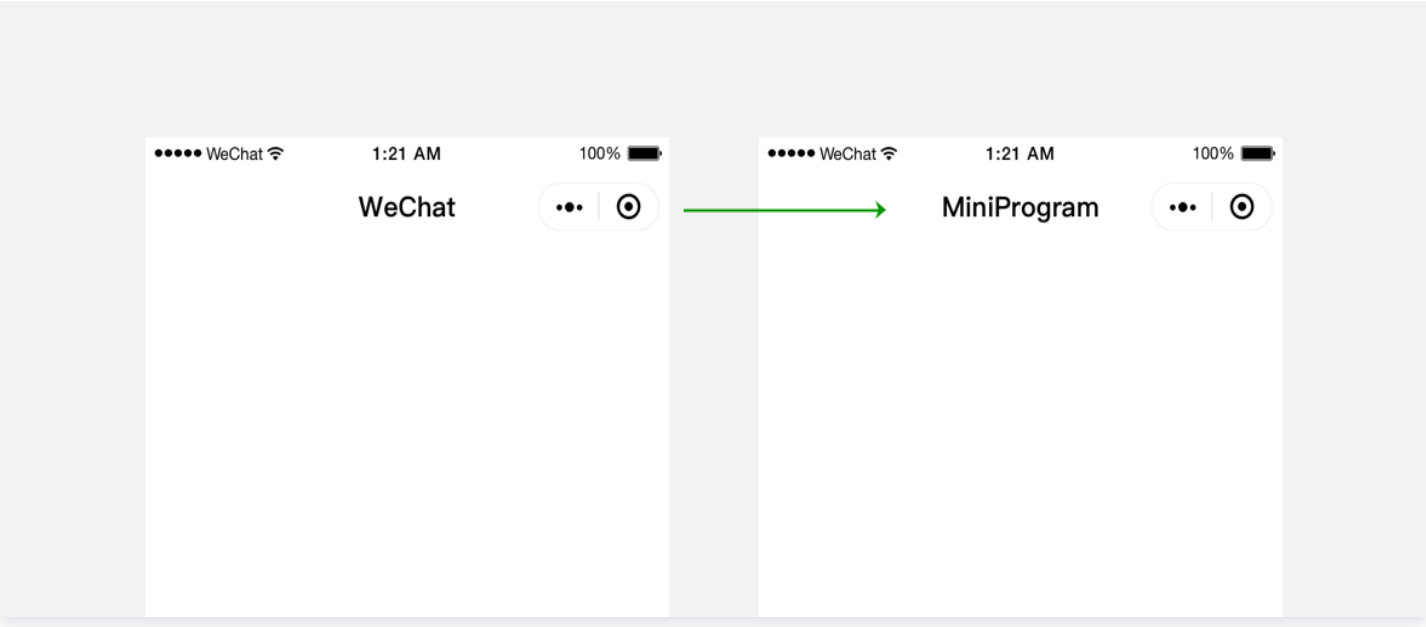
先看一个例子，打开开发工具的编辑器，在根目录下可以找到 app.json 文件，双击打开，本章代码 app.json文件代码如下：

app.json 文件代码

```
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "WeChat",
    "navigationBarTextStyle":"black"
  }
}
```

尝试修改第 9 行为 "navigationBarTitleText": "MiniProgram" ，如下图所示。

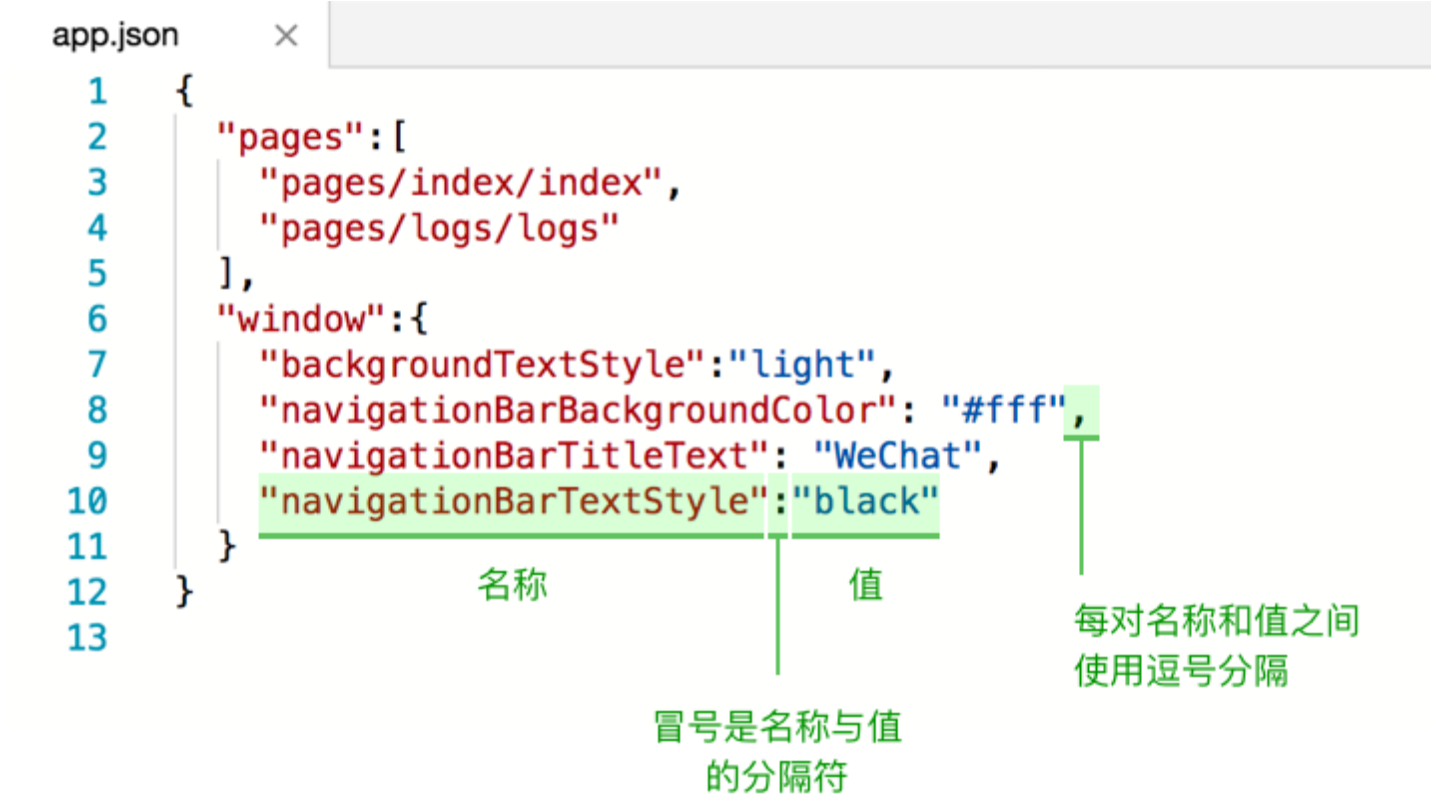
保存代码，开发者工具自动刷新后，注意到模拟器顶部 bar 的文本字段由 Wechat 变为了 MiniProgram。



JSON 文件在小程序代码中扮演静态配置的作用，在小程序运行之前就决定了小程序一些表现，需要注意的是小程序是无法在运行过程中去动态更新 JSON 配置文件从而发生对应的变化的。

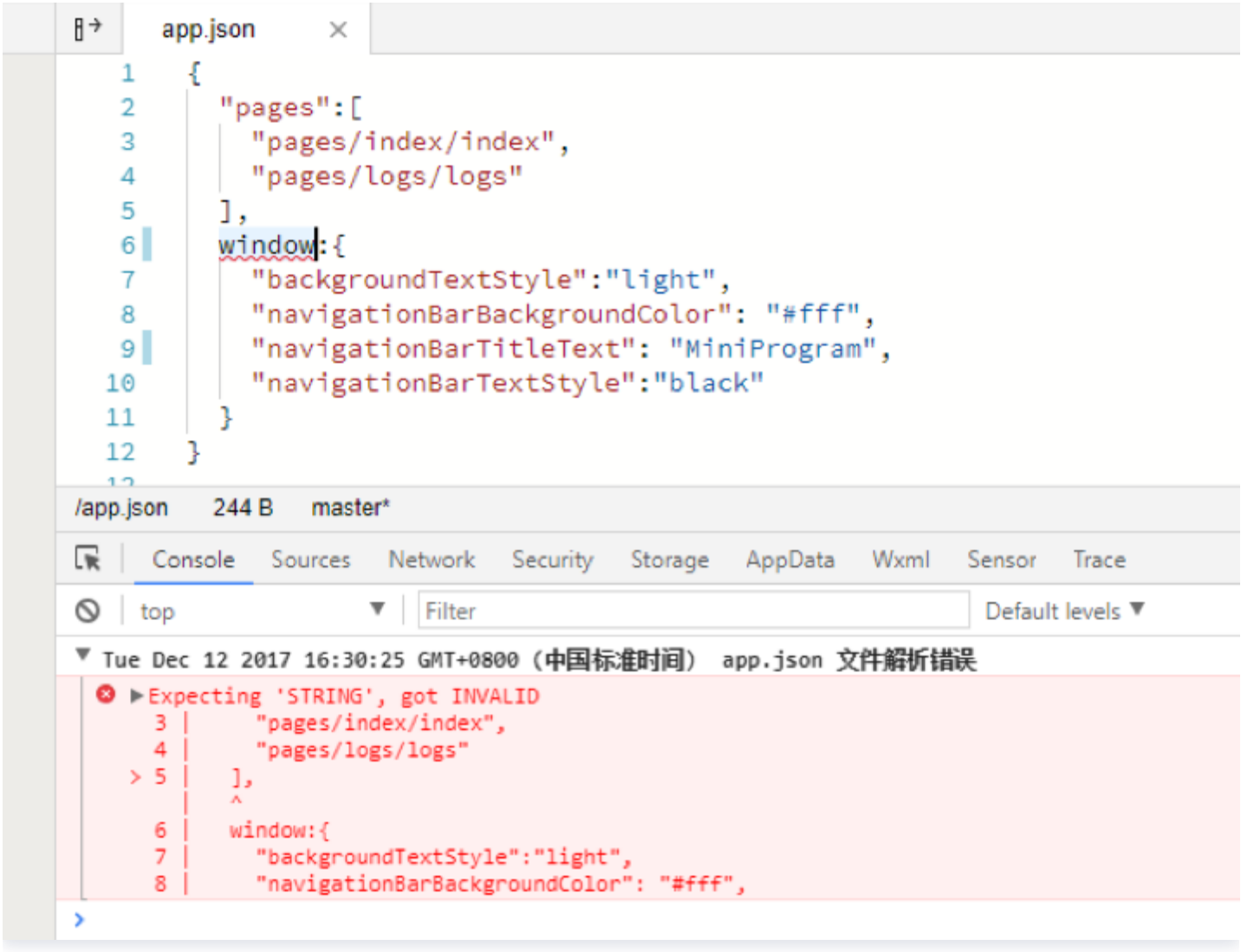
JSON 语法

相比于 XML ，JSON 格式最大的优点是易于人的阅读和编写，通常不需要特殊的工具，就能读懂和修改，是一种轻量级的数据交换格式。
JSON 文件都是被包裹在一个大括号中 {}，通过 key-value 的方式来表达数据。



看起来同 JavaScript 的对象表达方式十分相似，但是有所不同。

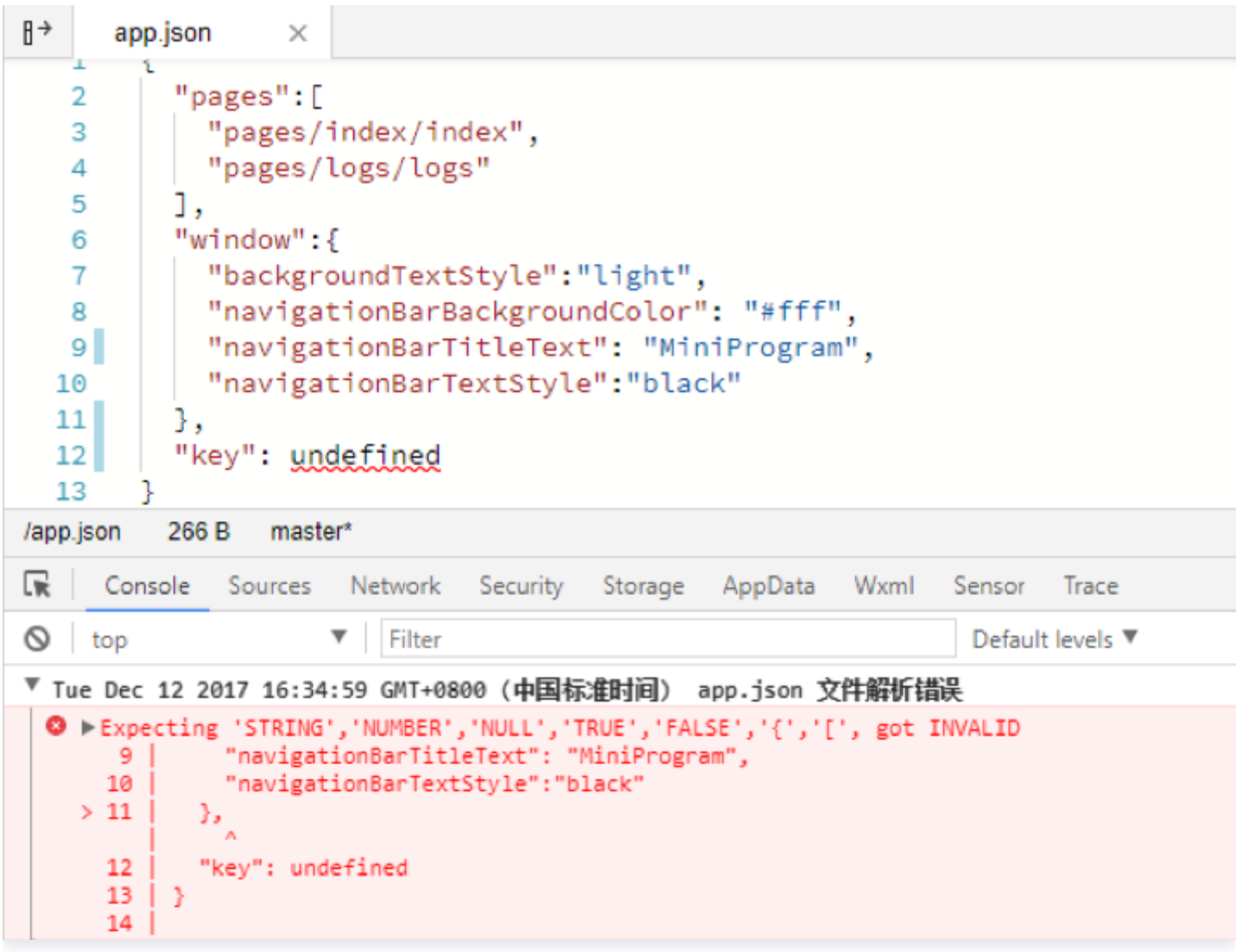
JSON 的 Key 必须包裹在一个双引号中，在实践中，编写 JSON 的时候，忘了给 Key 值加双引号或者是把双引号写成单引号是常见错误。



JSON 的值只能是以下几种数据格式：

1. 数字，包含浮点数和整数；
2. 字符串，需要包裹在双引号中；
3. Bool值，true 或者 false；
4. 数组，需要包裹在方括号中 []；
5. 对象，需要包裹在大括号中 {}；
6. Null；

其他任何格式都会触发报错，例如 JavaScript 中的 undefined 。



还需要注意的是 JSON 文件中无法使用注释，试图添加注释将会引发报错。

app.json

```
{
  // pages 定义了小程序页面
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "MiniProgram",
    "navigationBarTextStyle":"black"
  }
}
```

/app.json 282 B master*

Console Sources Network Security Storage AppData Wxml Sensor Trace

top Filter Default levels

Tue Dec 12 2017 16:50:25 GMT+0800 (中国标准时间) app.json 文件解析错误

```
Expecting 'STRING','}', got INVALID
> 1 | {
    | ^
  2 |   // pages 定义了小程序页面
  3 |   "pages":[
```

WXML 模板

最近更新时间：2024-04-10 15:12:32

WXML 全称是 WeiXin Markup Language，是小程序框架设计的一套标签语言，结合小程序的基础组件、事件系统，可以构建出页面的结构。
打开开发工具的编辑器，在根目录下找到 app.json 文件，双击打开，在 "pages/index/index" 上新增一行 "pages/wxml/index" 保存文件。模拟器刷新后，读者可以在编辑器中找到 pages/wxml/index.wxml 文件，本小结的学习通过修改这个文件来完成。

介绍

WXML 文件后缀名是 `.wxml` ，打开 pages/wxml/index.wxml 文件，有过 HTML 的开发经验的读者应该会很熟悉这种代码的书写方式，简单的 WXML 语句在语法上同 HTML 非常相似。

```
<!--pages/wxml/index.wxml-->

<text>pages/wxml/index.wxml</text>
```

不带有任何逻辑功能的 WXML 基本语法如下：

```
<!-- 在此处写注释 -->

<标签名 属性名1="属性值1" 属性名2="属性值2" ...> ...</标签名>
```

一个完整的 WXML 语句由一段开始标签和一段结束标签组成，在标签中可以是内容，也可以是其他的 WXML 语句，这一点上同 HTML 是一致的。有所不同的是，WXML 要求标签必须是严格闭合的，没有闭合将会导致编译错误，如下代码所示：

```
<text>hello world


<!--
text 没有闭合，导致编译错误：
VM148:2 ./pages/wxml/index.wxml
 end tag missing, near text
> 1 | <text>hello world
    | ^
-->
```

标签可以拥有属性，属性提供了有关的 WXML元素更多信息。属性总是定义在开始标签中，除了一些特殊的属性外，其余属性的格式都是key="value" 的方式成对出现。需要注意的是，WXML 中的属性是大小写敏感的，也就是说 class 和 Class 在 WXML 中是不同的属性，下方代码是一个文本标签的示例。

```
<!--一个简单的文本标签 -->
<text>hello world</text>

<!-- view 中包含了 text 标签 -->
<view>
  <text>hello world</text>
</view>
```

带属性的图片标签的例子如下所示：

```
<image class="userinfo-avatar" src="./image/a.png" ></image>
```

数据绑定

用户界面呈现会因为当前时刻数据不同而有所不同，或者是因为用户的操作发生动态改变，这就要求程序的运行过程中，要有动态的去改变渲染界面的能力。在 Web 开发中，开发者使用 JavaScript 通过 Dom 接口来完成界面的实时更新。在小程序中，使用 WXML 语言所提供的数据绑定功能，来完成此项功能。
先看一个简单的例子。
将 pages/wxml/index.wxml 文件的内容做一些简单的修改，如下代码所示：

```
<!--pages/wxml/index.wxml-->
<text>当前时间：{{time}}</text>
```

保存后工具刷新，模拟器并没有显示出当前的时间，这是因为我们并没有给 time 设置任何初始值，请打开 pages/wxml/index.js 文件，在 data 的大括号中加入： `time: (new Date()).toString()` ，如下代码所示：

```
// pages/wxml/index.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    time: (new Date()).toString()
  },
})
```

保存，模拟器刷新后正确的展示了当前时间，并且每次编译时间都会被更新。

WXML 通过 {{变量名}} 来绑定 WXML 文件和对应的 JavaScript 文件中的 data 对象属性。
后文中为了保持简单，通过以下格式来展示上述的代码逻辑，使用第一段注释来表明 WXML 对应的脚本文件中的 data 结构，如下代码所示：

```
<!--
{
  time: (new Date()).toString()
}
-->
<text>当前时间：{{time}}</text>
```

属性值也可以动态的去改变，有所不同的是，属性值必须被包裹在双引号中，如下代码所示：

```
<!-- 正确的写法 -->
<text data-test="{{test}}"> hello world</text>


<!-- 错误的写法  -->
<text data-test={{test}}> hello world </text >
```

需要注意的是**变量名是大小写敏感的，也就是说 {{name}} 和 {{Name}} 是两个不同的变量**，如下代码所示：

```
<!--
{
  w: 'w',
  W: 'W'
}
-->


<view>{{w}}</view>
<view>{{W}}</view>


<!-- 输出
w
W
-->
```

还需要注意，**没有被定义的变量的或者是被设置为 undefined 的变量不会被同步到 wxml 中**，如下代码所示：

```
<!--
```

```
{
  var2: undefined,
  var3: null,
  var4: "var4"
}
-->


<view>{{var1}}</view>
<view>{{var2}}</view>
<view>{{var3}}</view>
<view>{{var4}}</view>


<!--
输出:
null
var4
-->
```

关于数据绑定的概念在第三章中有更为详细的介绍。

逻辑语法

通过 {{ 变量名 }} 语法可以使得 WXML 拥有动态渲染的能力，除此外还可以在 {{ }} 内进行简单的逻辑运算。

三元运算：

```
<!-- 根据 a 的值是否等于 10 在页面输出不同的内容 -->
<text>{{ a === 10? "变量 a 等于10": "变量 a 不等于10"}}</text>
```

算数运算：

```
<!--
{ a: 1,  b: 2, c: 3 }
-->


<view> {{a + b}} + {{c}} + d </view>


<!-- 输出 3 + 3 + d -->
```

类似于算数运算，还支持字符串的拼接，如下代码所示：

```
<!--
{ name: 'world' }
-->


<view>{{"hello " + name}}</view>


<!-- 输出 hello world -->
```

{{ }}中还可以直接放置数字、字符串或者是数组，如下代码所示：

```
<text>{{[1,2,3]}}</text>

<!-- 输出 1,2,3 -->



<text>{{"hello world"}}</text>

<!-- 输出 hello world -->
```

条件逻辑

WXML 中，使用 wx:if="{{condition}}" 来判断是否需要渲染该代码块：

```
<view wx:if="{{condition}}"> True </view>
```

使用 wx:elif 和 wx:else 来添加一个 else 块：

```
<view wx:if="{{length > 5}}"> 1 </view>
<view wx:elif="{{length > 2}}"> 2 </view>
<view wx:else> 3 </view>
```

因为 wx:if 是一个控制属性，需要将它添加到一个标签上。如果要一次性判断多个组件标签，可以使用一个 <block/> 标签将多个组件包装起来，并在上边使用 wx:if 控制属性。

```
<block wx:if="{{true}}">
  <view> view1 </view>
  <view> view2 </view>
</block>
```

列表渲染

在组件上使用 wx:for 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该组件。默认数组的当前项的下标变量名默认为 index，数组当前项的变量名默认为 item，如下代码所示：

```
<!-- array 是一个数组 -->
<view wx:for="{{array}}">
 {{index}}: {{item.message}}
</view>

<!-- 对应的脚本文件
Page({
 data: {
  array: [{
   message: 'foo',
  }, {
   message: 'bar'
  }]
 }
})
-->
```

使用 wx:for-item 指定数组当前元素的变量名，使用 wx:for-index 指定数组当前下标的变量名：

```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
 {{idx}}: {{itemName.message}}
</view>
```

类似 block wx:if ，也可以将 wx:for 用在 <block/> 标签上，以渲染一个包含多节点的结构块，如下代码所示：

```
<block wx:for="{{[1, 2, 3]}}">
 <view> {{index}}: </view>
 <view> {{item}} </view>
</block>
```

如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（如 `<input/>` 中的输入内容， `<switch/>` 的选中状态），需要使用 `wx:key` 来指定列表中项目的唯一的标识符。wx:key 的值以两种形式提供：

1. 字符串，代表在 for 循环的 array 中 item 的某个 property，该 property 的值需要是列表中唯一的字符串或数字，且不能动态改变。
2. 保留关键字 this 代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数字，如：

当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率，如下代码所示：

```
<switch wx:for="{{objectArray}}" wx:key="unique" > {{item.id}} </switch>
<button bindtap="switch"> Switch </button>
<button bindtap="addToFront"> Add to the front </button>


<switch wx:for="{{numberArray}}" wx:key="*this" > {{item}} </switch>
<button bindtap="addNumberToFront"> Add Number to the front </button>
```

```
Page({
  data: {
    objectArray: [
      {id: 5, unique: 'unique_5'},
      {id: 4, unique: 'unique_4'},
      {id: 3, unique: 'unique_3'},
      {id: 2, unique: 'unique_2'},
      {id: 1, unique: 'unique_1'},
      {id: 0, unique: 'unique_0'},
    ],
    numberArray: [1, 2, 3, 4]
  },
  switch: function(e) {
    const length = this.data.objectArray.length
    for (let i = 0; i < length; ++i) {
      const x = Math.floor(Math.random() * length)
      const y = Math.floor(Math.random() * length)
      const temp = this.data.objectArray[x]
      this.data.objectArray[x] = this.data.objectArray[y]
      this.data.objectArray[y] = temp
    }
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addToFront: function(e) {
    const length = this.data.objectArray.length
    this.data.objectArray = [{id: length, unique: 'unique_' +
length}].concat(this.data.objectArray)
    this.setData({
      objectArray: this.data.objectArray
    })
  },
```

```
  addNumberToFront: function(e){
    this.data.numberArray = [ this.data.numberArray.length + 1
].concat(this.data.numberArray)
    this.setData({
      numberArray: this.data.numberArray
    })
  }
})
```

模板

WXML提供模板（template），可以在模板中定义代码片段，然后在不同的地方调用。使用 name 属性，作为模板的名字。然后在 <template/> 内定义代码片段，如下代码所示：

```
<template name="msgItem">
  <view>
    <text> {{index}}: {{msg}} </text>
    <text> Time: {{time}} </text>
  </view>
</template>
```

使用 is 属性，声明需要的使用的模板，然后将模板所需要的 data 传入，如下代码所示：

```
<!--
item: {
  index: 0,
  msg: 'this is a template',
  time: '2016-06-18'
}
-->


<template name="msgItem">
  <view>
    <text> {{index}}: {{msg}} </text>
    <text> Time: {{time}} </text>
  </view>
</template>


<template is="msgItem" data="{{...item}}"/>

<!-- 输出
0: this is a template Time: 2016-06-18
```

```
-->
```

使用 is 可以动态决定具体需要渲染哪个模板，如下代码所示：

```
<template name="odd">
  <view> odd </view>
</template>


<template name="even">
  <view> even </view>
</template>


<block wx:for="{{[1, 2, 3, 4, 5]}}">
  <template is="{{item % 2 == 0 ? 'even' : 'odd'}}"/>
</block>



<!-- 输出
odd
even
odd
even
odd
-->
```

引用

WXML 提供两种文件引用方式 import 和 include。
import 可以在该文件中使用目标文件定义的 template，如：在 item.wxml 中定义了一个叫 item 的 template。

```
<!-- item.wxml -->
<template name="item">
  <text>{{text}}</text>
</template>
```

在 index.wxml 中引用了 item.wxml，就可以使用 item 模板：

```
<import src="item.wxml"/>
```

```
<template is="item" data="{{text: 'forbar'}}"/>
```

需要注意的是 import 有作用域的概念，即只会 import 目标文件中定义的 template，而不会 import 目标文件中 import 的 template，简言之就是 import 不具有递归的特性。
例如：C 引用 B，B 引用 A，在 C 中可以使用 B 定义的 template，在 B 中可以使用 A 定义的 template ，但是 C 不能使用 A 定义的 template，如下代码所示：

```
<!-- A.wxml -->
<template name="A">
  <text> A template </text>
</template>
```

```
<!-- B.wxml --><import src="a.wxml"/><template name="B">  <text> B template
</text></template>
```

```
<!-- C.wxml -->
<import src="b.wxml"/>

<template is="A"/>  <!-- 这里将会触发一个警告，因为 b 中并没有定义模板 A -->

<template is="B"/>
```

include 可以将目标文件中除了 `<template/> <wxs/>` 外的整个代码引入，相当于是拷贝到 `include` 位置，如下代码所示：

```
<!-- index.wxml -->
<include src="header.wxml"/>

<view> body </view>

<include src="footer.wxml"/>
```

```
<!-- header.wxml -->
<view> header </view>
```

```
<!-- footer.wxml -->
<view> footer </view>
```

共同属性

所有 wxml 标签都支持的属性称之为共同属性，如下表所示。

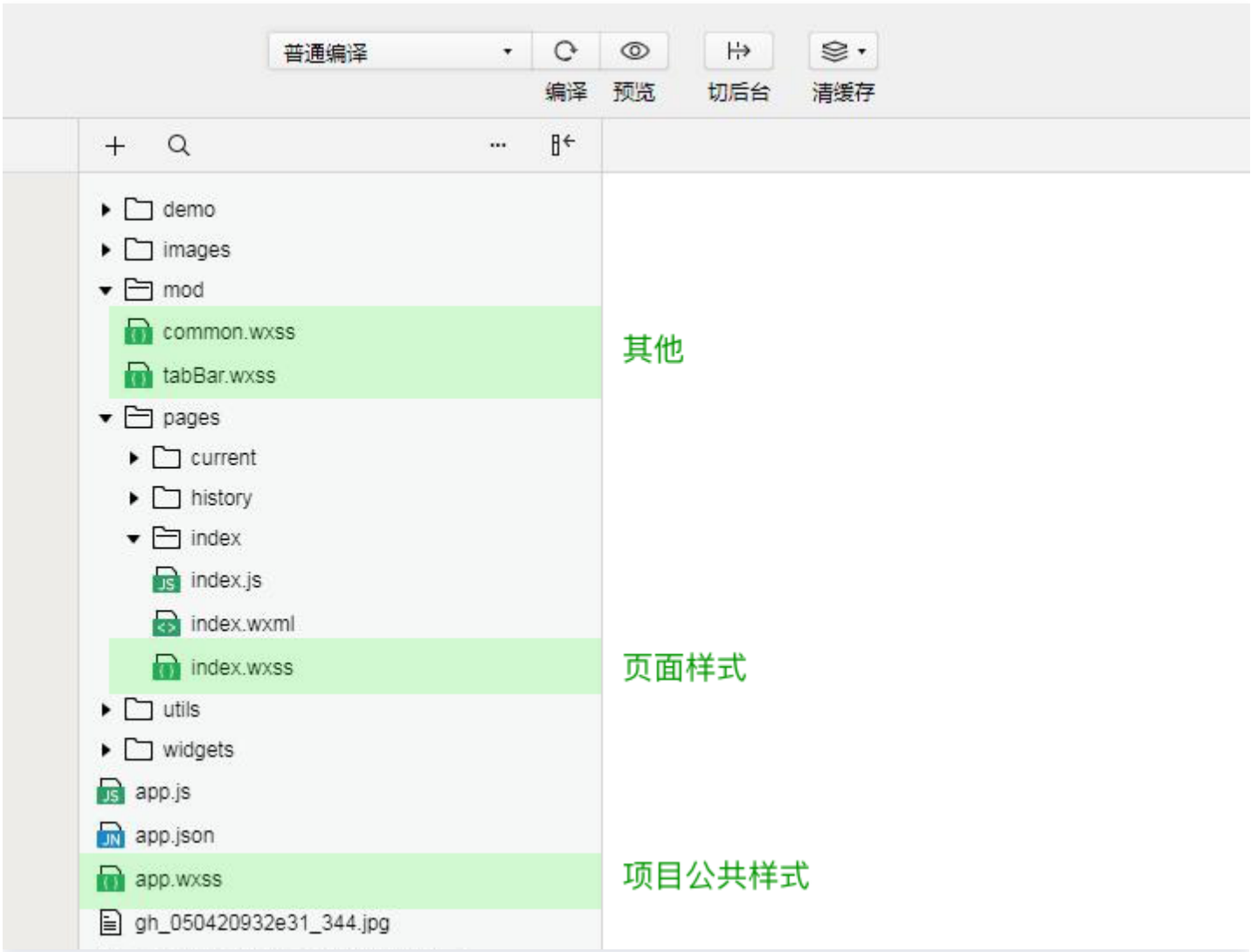
属性名	类型	描述	注解
id	String	组件的唯一标识	整个页面唯一
class	String	组件的样式类	在对应的 WXSS 中定义的样式类
style	String	组件的内联样式	可以动态设置的内联样式
hidden	Boolean	组件是否显示	所有组件默认显示
data-*	Any	自定义属性	组件上触发的事件时，会发送给事件处理函数
bind*/catch*	EventHandler	组件的事件	-

WXSS 样式

最近更新时间：2024-05-11 17:21:31

WXSS（WeiXin Style Sheets）是一套用于小程序的样式语言，用于描述 WXML 的组件样式，也就是视觉上的效果。
WXSS 与 Web 开发中的 CSS 类似。为了更适合小程序开发，WXSS 对 CSS 做了一些补充以及修改。

文件组成



app.json

```
{
  "pages":[
    "pages/rpx/index",
    "pages/box/index",
    "pages/inlinestyle/index",
    "pages/flex/index"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": " ",
    "navigationBarTextStyle":"black"
  }
}
```

项目公共样式：根目录中的 app.wxss 为项目公共样式，它会被注入到小程序的每个页面。页面样式：与 app.json 注册过的页面同名且位置同级的 WXSS 文件。例如上图注册了pages/rpx/index 页面，那 pages/rpx/index.wxss 为页面 pages/rpx/index.wxml 的样式。

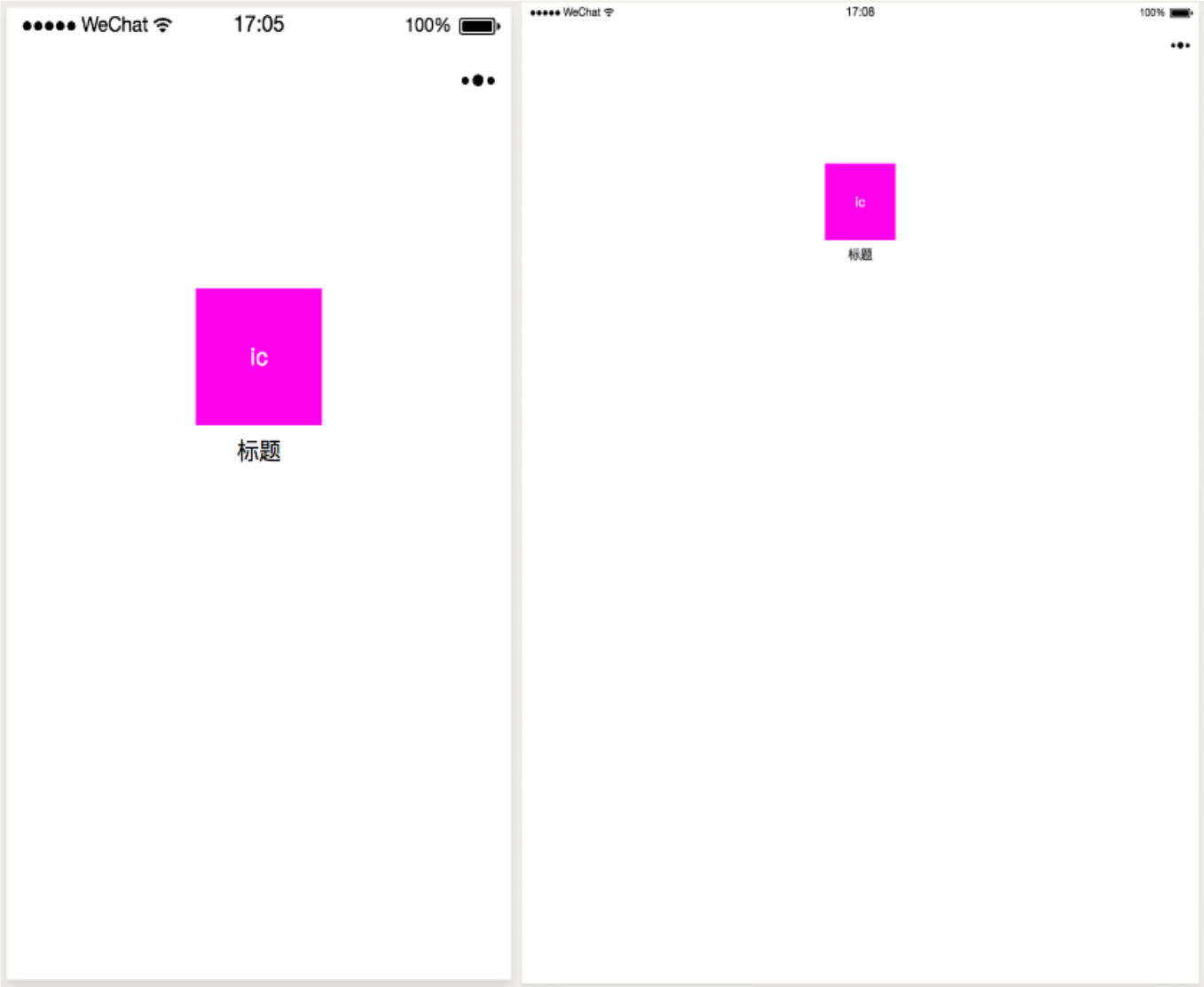
其它样式：其它样式可以被项目公共样式和页面样式引用，引用方法查看本章中的 WXSS 引用 。

在小程序开发中，开发者不需要像 Web 开发那样去优化样式文件的请求数量，只需要考虑代码的组织即可。样式文件最终会被编译优化，具体的编译原理我们留在后面章节再做介绍。

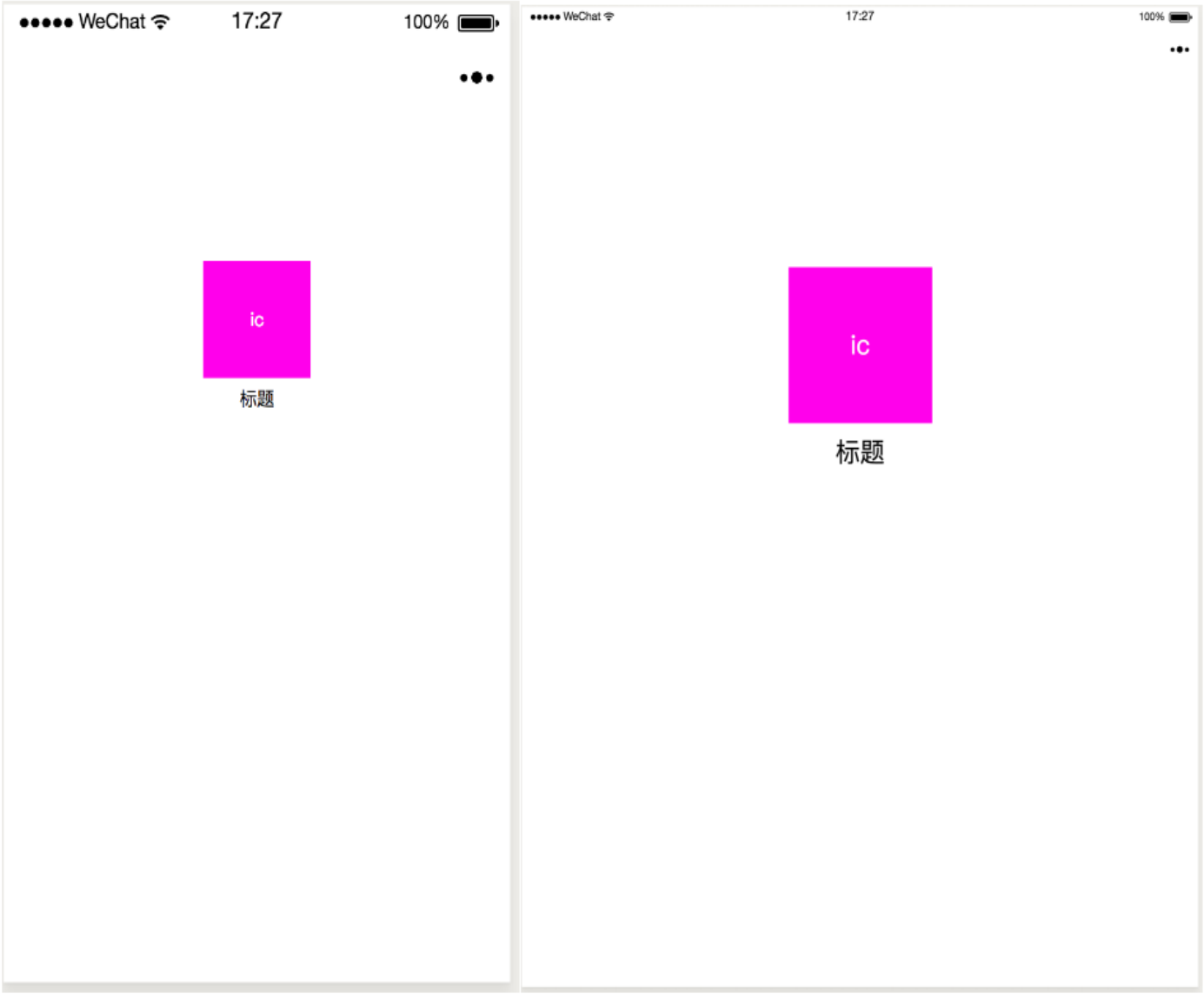
尺寸单位

在 WXSS 中，引入了 rpx（responsive pixel）尺寸单位。引用新尺寸单位的目的是，适配不同宽度的屏幕，开发起来更简单。

如下图所示，同一个元素，在不同宽度的屏幕下，如果使用 px 为尺寸单位，有可能造成页面留白过多。



修改为 rpx 尺寸单位，效果如下图所示。



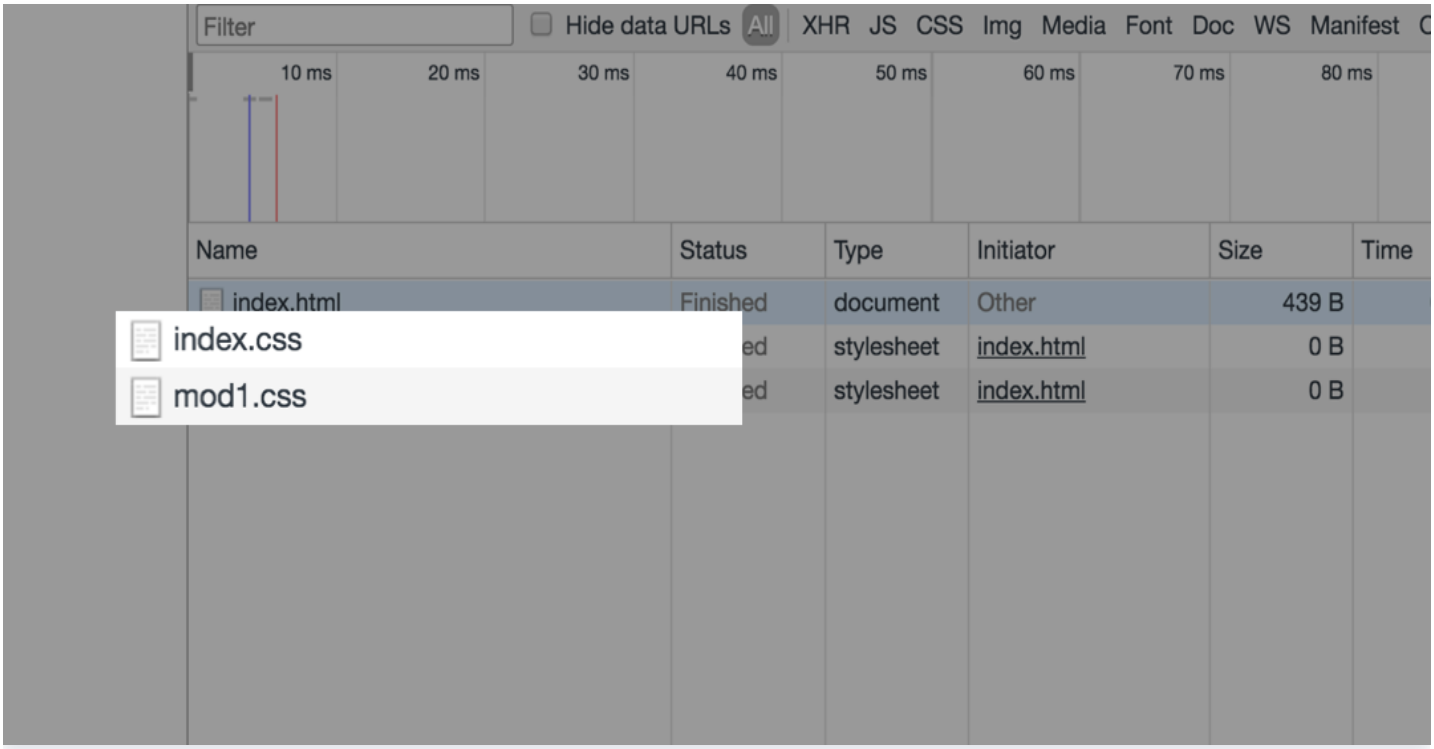
小程序编译后，rpx 会做一次 px 换算。换算是以375个物理像素为基准，也就是在一个宽度为375物理像素的屏幕下，1rpx = 1px。
举个例子：iPhone6屏幕宽度为375px，共750个物理像素，那么1rpx = 375 / 750 px = 0.5px。

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

WXSS 引用

在 CSS 中，开发者可以这样引用另一个样式文件：@import url('./test_0.css')

这种方法在请求上不会把 test_0.css 合并到 index.css 中，也就是请求 index.css 的时候，会多一个 test_0.css 的请求。



在小程序中，我们依然可以实现样式的引用，样式引用是这样写：

```
@import './test_0.wxss'
```

由于 WXSS 最终会被编译打包到目标文件中，用户只需要下载一次，在使用过程中不会因为样式的引用而产生多余的文件请求。

内联样式

WXSS 内联样式与 Web 开发一致：

```
<!--index.wxml-->

<!--内联样式-->
<view style="color: red; font-size: 48rpx"></view>
```

小程序支持动态更新内联样式：

```
<!--index.wxml-->

<!--可动态变化的内联样式-->
<!--
```

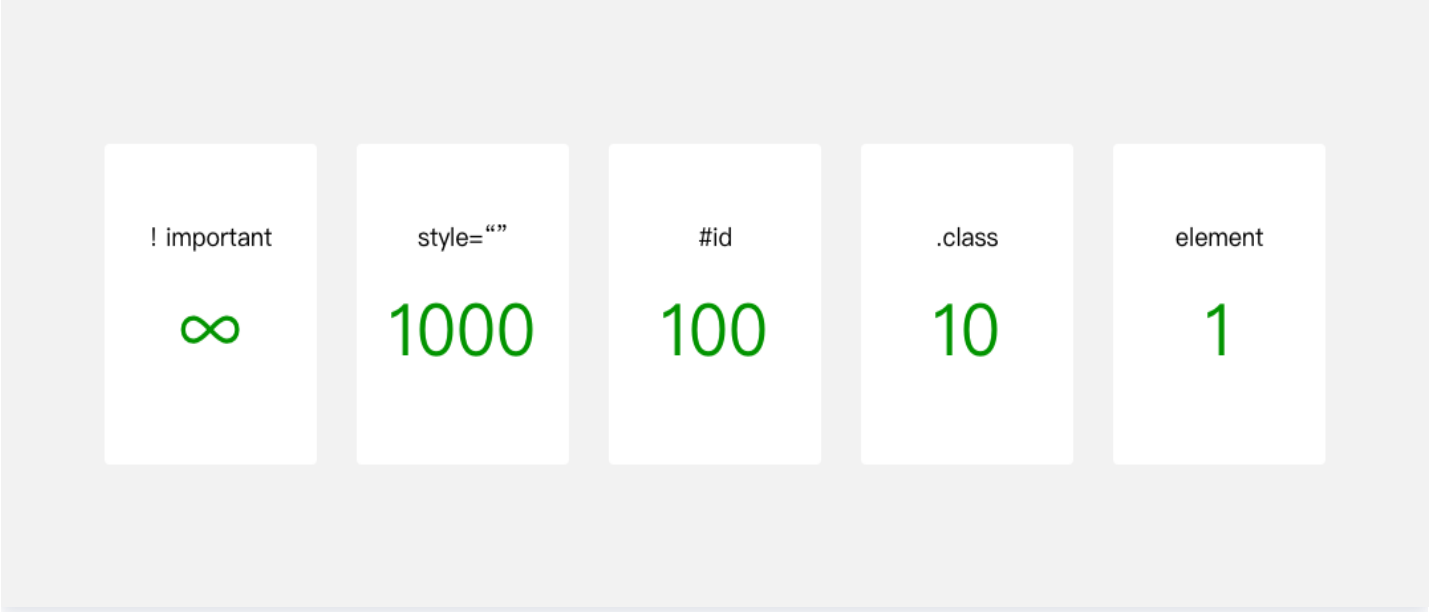
```
{
  eleColor: 'red',
  eleFontsize: '48rpx'
}
-->
<view style="color: {{eleColor}}; font-size: {{eleFontsize}}"></view>
```

选择器

目前支持的选择器如下表所示。

类型	选择器	样例	样例描述
类选择器	.class	.intro	选择所有拥有 class="intro" 的组件
id选择器	#id	#firstname	选择拥有 id="firstname" 的组件
元素选择器	element	view checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
伪元素选择器	::after	view::after	在 view 组件后边插入内容
伪元素选择器	::before	view::before	在 view 组件前边插入内容

WXSS 优先级与 CSS 类似，权重如下图所示。



权重越高越优先。在优先级相同的情况下，后设置的样式优先级高于先设置的样式。
WXSS 选择器优先级权重，代码如下所示：

```
view{ // 权重为 1
  color: blue
}

.ele{ // 权重为 10
  color: red
}

#ele{ // 权重为 100
  color: pink
}

view#ele{ // 权重为 1 + 100 = 101，优先级最高，元素颜色为orange
  color: orange
}

view.ele{ // 权重为 1 + 10 = 11
  color: green
}
```

官方样式库

为了减轻开发者样式开发的工作量，我们提供了 WeUI.wxss 基础样式库。
WeUI 是一套与微信原生视觉体验一致的基础样式库，由微信官方设计团队为微信内网页和微信小程序量身设计，令用户的使用感知更加统一。包含 button、cell、dialog、progress、toast、article、actionsheet、icon 等各式原生。


无 SIM 卡
下午5:26
WeUI for 小程序
WeUI
WeUI 是一套同微信原生视觉体验一致的基础样式库，由微信官方设计团队为微信内网页和微信小程序量身设计，令用户的使用感知更加统一。
表单
基础组件
操作反馈
导航相关
搜索相关

JavaScript 脚本

最近更新时间：2024-04-11 10:00:42

小程序的主要开发语言是 JavaScript ，开发者使用 JavaScript 来开发业务逻辑以及调用小程序的 API 来完成业务需求。

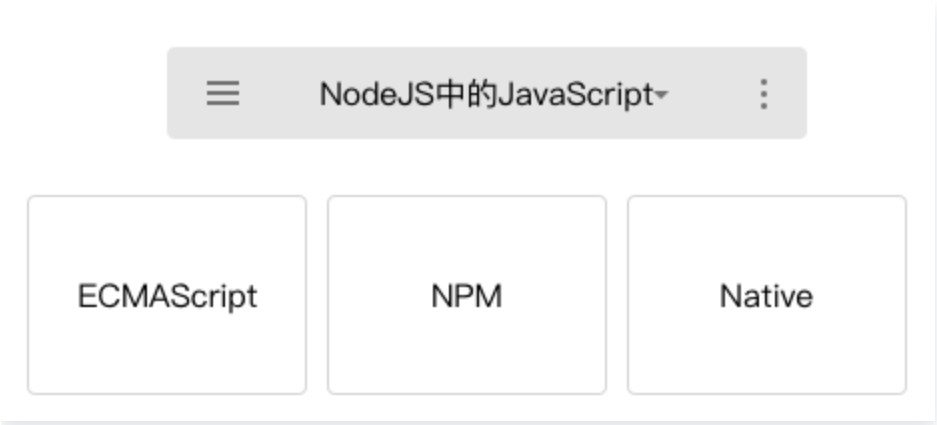
ECMAScript

在大部分开发者看来，ECMAScript 和 JavaScript 表达的是同一种含义，但是严格的说，两者的意义是不同的。ECMAScript 是一种由 Ecma 国际通过 ECMA-262标准化的脚本程序设计语言， JavaScript 是 ECMAScript 的一种实现。理解 JavaScript 是 ECMAScript 一种实现后，可以帮助开发者理解小程序中的 JavaScript 同浏览器中的 JavaScript 以及 NodeJS 中的 JavaScript 是不相同的。
ECMA-262 规定了 ECMAScript 语言的几个重要组成部分：

1. 语法
2. 类型
3. 语句
4. 关键字
5. 操作符
6. 对象

浏览器中JavaScript 构成如下图所示：

浏览器中的 JavaScript 是由 ECMAScript 和 BOM（浏览器对象模型）以及 DOM（文档对象模型）组成的，Web 前端开发者会很熟悉这两个对象模型，它使得开发者可以去操作浏览器的一些表现，例如修改 URL、修改页面呈现、记录数据等。NodeJS 中 JavaScript 构成如下图所示：



NodeJS 中的 JavaScript 是由 ECMAScript 和 NPM 以及 Native 模块组成，NodeJS 的开发者会非常熟悉 NPM 的包管理系统，通过各种拓展包来快速的实现一些功能，同时通过使用一些原生的模块例如 FS、HTTP、OS等来拥有一些语言本身所不具有的能力。
那么，同开发者所熟悉的这两个环境是不同的，小程序中 JavaScript 构成如下图所示。

小程序中的 JavaScript 是由ECMAScript 以及小程序框架和小程序 API 来实现的。同浏览器中的 JavaScript 相比没有 BOM 以及 DOM 对象，所以类似 JQuery、Zepto 这种浏览器类库是无法在小程序中运行起来的，同样的缺少 Native 模块和 NPM 包管理的机制，小程序中无法加载原生库，也无法直接使用大部分的 NPM 包。

小程序的执行环境

明白了小程序中的 JavaScript 同浏览器以及 NodeJS 有所不同后，开发者还需要注意到另外一个问题，不同的平台的小程序的脚本执行环境也是有所区别的。
小程序目前可以运行在三大平台：

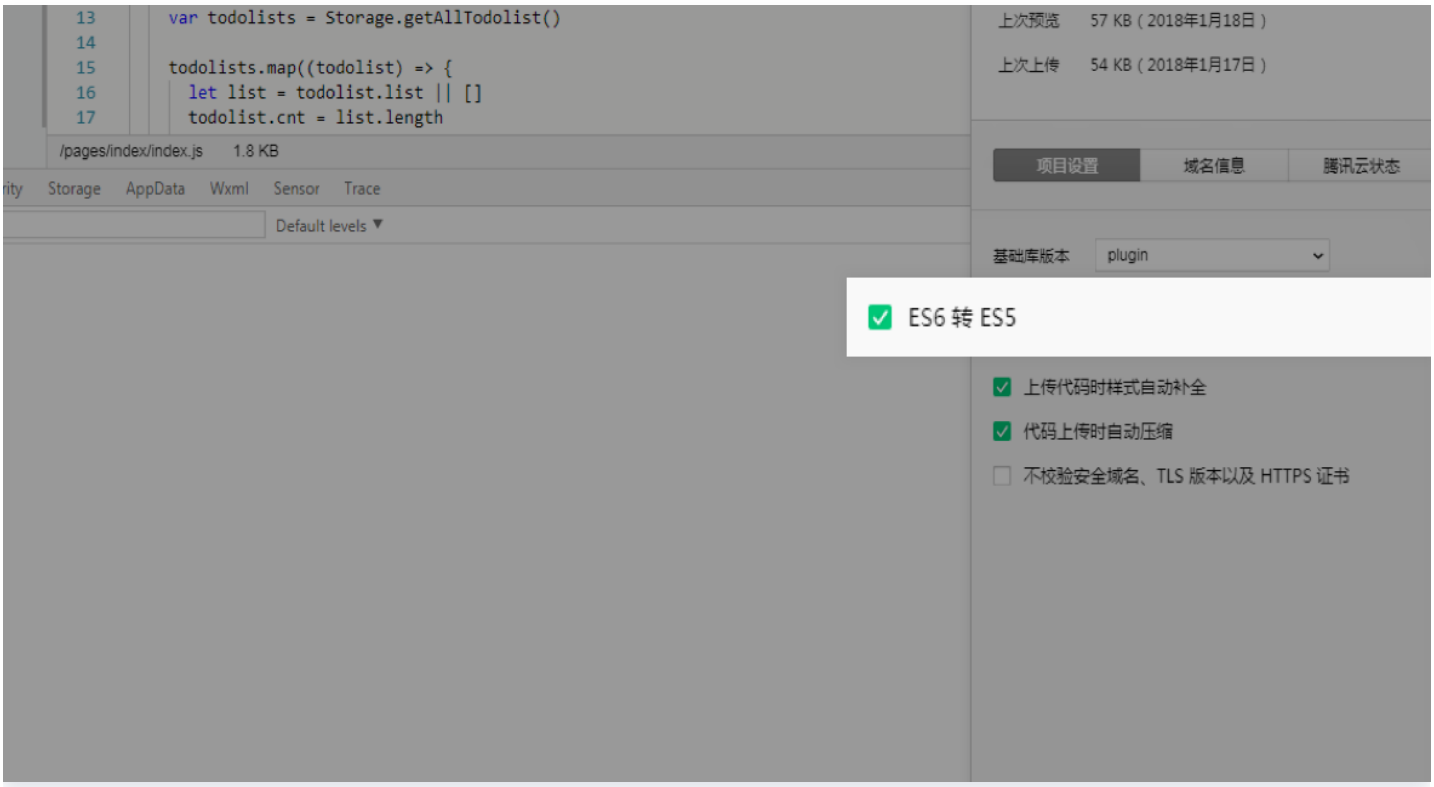
- iOS 平台，包括 iOS9、iOS10、iOS11。
- Android 平台
- 小程序 IDE

这种区别主要是体现三大平台实现的 ECMAScript 的标准有所不同。截止到当前一共有七个版本的 ECMAScript 标准，目前开发者大部分使用的是 ECMAScript 5 和 ECMAScript 6 的标准，但是在小程序中， iOS9 和 iOS10 所使用的运行环境并没有完全的兼容到 ECMAScript 6 标准，一些 ECMAScript 6 中规定的语法和关键字是没有的或者同标准是有所不同的，例如：

- 箭头函数

- let const
- 模板字符串
- ...

所以一些开发者会发现有些代码在旧的手机操作系统上出现一些语法错误。为了帮助开发者解决这类问题，小程序 IDE 提供语法转码工具帮助开发者，将 ECMAScript 6代码转为 ECMAScript 5代码，从而在所有的环境都能得到很好的执行。
开发者需要在项目设置中，勾选 ES6 转 ES5 开启此功能。



模块化

浏览器中，所有 JavaScript 是在运行在同一个作用域下的，定义的参数或者方法可以被后续加载的脚本访问或者改写。同浏览器不同，小程序中可以将任何一个 JavaScript 文件作为一个模块，通过 module.exports 或者 exports 对外暴露接口。
请看是一个简单模块示例，B.js 引用模块A，并使用A暴露的 multiplyBy2 方法完成一个变量乘以2的操作，如下代码所示：

```
// moduleA.js
module.exports = function( value ){
  return value * 2;
}
```

```
// B.js
```

```
// 在B.js中引用模块A
var multiplyBy2 = require('./moduleA')
var result = multiplyBy2(4)
```

下方代码在需要使用这些模块的文件中，使用 require(path) 将公共代码引入：

```
var common = require('common.js')
Page({
  helloMINA: function() {
    common.sayHello('MINA')
  },
  goodbyeMINA: function() {
    common.sayGoodbye('MINA')
  }
})
```

脚本的执行顺序

浏览器中，脚本严格按照加载的顺序执行，如下代码所示：

```
<html>
<head>
  <!-- a.js
  console.log('a.js')
   -->
  <script src ="a.js"></script>
  <script>
    console.log('inline script')
  </script>

  <!-- b.js
  console.log('b.js')
   -->
  <script src ="b.js"></script>
</head>
</html>
```

以上代码的输出是：

```
a.js
inline script
b.js
```

而在小程序中的脚本执行顺序有所不同。小程序的执行的入口文件是 app.js 。并且会根据其中 require 的模块顺序决定文件的运行顺序，下方代码是一个 app.js 示例。

```
/* a.js
console.log('a.js')
*/
var a = require('./a.js')
console.log('app.js')

/* b.js
console.log('b.js')
*/
var b = require('./b.js')
```

以上代码的输出顺序是：

```
a.js
app.js
b.js
```

当 app.js 执行结束后，小程序会按照开发者在 app.json 中定义的 pages 的顺序，逐一执行，如下代码所示：

```
{
  "pages": [
    "pages/index/index",
    "pages/log/log",
    "pages/result/result"
  ],
  "window": {}
}
```

```
// app.js
console.log('app.js')
```

```
// pages/index/index
console.log('pages/index/index')
```

```
// pages/log/log
console.log('pages/log/log')
```

```
// pages/result/result
console.log('pages/result/result')
```

以上文件执行后输出的结果如下：

```
app.js
pages/index/index
pages/log/log
pages/result/result
```

作用域

同浏览器中运行的脚本文件有所不同，小程序的脚本的作用域同 NodeJS 更为相似。
在文件中声明的变量和函数只在该文件中有效，不同的文件中可以声明相同名字的变量和函数，不会互相影响，如下代码所示：

```
// a.js
// 定义局部变量
var localValue = 'a'
```

```
// b.js
//在脚本 b.js 中无法访问 a.js 定义的变量：
// 定义局部变量console.log(localValue) // 触发一个错误 b.js中无法访问 a.js 中定义的变量
```

当需要使用全局变量的时，通过使用全局函数 getApp() 获取全局的实例，并设置相关属性值，来达到设置全局变量的目的，如下代码所示：

```
// a.js
// 获取全局变量
var global = getApp()
global.globalValue = 'globalValue'
```

```
// b.js
// 访问全局变量
var global = getApp()
console.log(global.globalValue)
// 输出 globalValue
```

需要注意的是，上述示例只有在 a.js 比 b.js 先执行才有效，当需要保证全局的数据可以在任何文件中安全的被使用到，那么可以在 App() 中进行设置，如下代码所示：

```
// app.js
App({
  globalData: 1
})
```

```
// a.js
// 局部变量
var localValue = 'a'

// 获取 global 变量
var app = getApp()

// 修改 global 变量
app.globalData++  // 执行后 globalData 数值为 2
```

```
// b.js
// 定义另外的局部变量，并不会影响 a.js 中文件变量
var localValue = 'b'

// 如果先执行了 a.js 这里的输出应该是 2
console.log(getApp().globalData)
```

框架层

框架概述

最近更新时间：2024-03-21 15:51:52

小程序开发框架的目标是为开发者提供一种简单、高效的方式，在宿主 App 中开发具有原生 App 体验的服务。为此，框架提供了以下工具和框架：

- 视图层描述语言 WXML 和 WXSS。
- 基于 JavaScript 的逻辑层框架。

此外，框架还提供了数据传输和事件系统，以便视图层和逻辑层之间进行通信和交互。这些系统可以帮助开发者更轻松地实现各种功能和交互效果，从而提高开发效率和用户体验。

响应的数据绑定

小程序框架的核心是一个响应式数据绑定系统，它可以简单地保持数据与视图同步。当数据发生变化时，只需在逻辑层进行修改，视图层会自动更新。小程序框架分为两部分：

- 视图层（View）
- 逻辑层（App Service）

视图层负责渲染页面，而逻辑层则负责处理业务逻辑和数据操作。两者之间通过数据绑定系统进行通信和交互，从而实现数据和视图的同步更新。
通过如下示例来看：

```
<!-- This is our View -->
<view>Hello {{name}}!</view>
<button bindtap="changeName">Click me!</button>
```

```
// This is our App Service.
// This is our data.
const helloData = {
  name: 'TMF'
}

// Register a Page.
Page({
  data: helloData,
  changeName(e) {
    // sent data change to view
    this.setData({
      name: 'MINA'
    })
```

```
  }
})
```

- 开发者通过框架将逻辑层数据中的 `name` 与视图层的 `name` 进行了绑定，所以在页面一打开的时候会显示 `Hello TCMPP!` 。
- 当单击按钮的时候，视图层会发送 `changeName` 的事件给逻辑层，逻辑层找到并执行对应的事件处理函数。
- 回调函数触发后，逻辑层执行 `setData` 的操作，将 `data` 中的 `name` 从 `TCMPP` 变为 `MINA` ，因为该数据和视图层已经绑定了，从而视图层会自动改变为 `Hello MINA!` 。

页面管理

小程序框架管理了整个小程序的页面路由，可以做到页面间的无缝切换，并给以页面完整的生命周期。开发者只需要将页面的数据、方法、生命周期函数注册到框架中，其他的一切复杂的操作都交由框架处理。
框架可以帮助开发者更轻松地实现页面的管理和切换，从而提高开发效率和用户体验。开发者只需要关注页面的业务逻辑和交互效果，而不必过多关注底层实现细节。

基础组件

框架提供了一套基础的组件，这些组件自带 TMF 风格的样式以及特殊的逻辑，开发者可以通过组合基础组件，创建出强大的 TMF 小程序 。

丰富的 API

框架提供丰富的宿主 App 原生 API，可以方便的调起宿主 App 提供的能力，如获取用户信息，本地存储，支付功能等。

目录结构

最近更新时间：2024-03-21 15:51:52

小程序包含一个描述整体程序的 `app` 和多个描述各自页面的 `page` 。
一个小程序主体部分由三个文件组成，必须放在项目的根目录，如下：

文件	必需	作用
app.js	是	小程序逻辑
app.json	是	小程序公共配置
app.wxss	否	小程序公共样式表

一个小程序页面由四个文件组成，分别是：

文件	必需	作用
js	是	页面逻辑
wxml	是	页面结构
json	否	页面配置
wxss	否	页面样式表

> ⚠ **注意：**
> 为了方便开发者减少配置项，描述页面的四个文件必须具有相同的路径与文件名。

允许上传的文件

在项目目录中，以下文件会经过编译，因此上传之后无法直接访问到：*.js、app.json、*.wxml、*.wxss（其中 wxml 和 wxss 文件仅针对在 app.json 中配置了的页面）。除此之外，只有后缀名在白名单内的文件可以被上传。具体白名单列表如下：

1. qs
2. png
3. jpg
4. jpeg
5. gif
6. svg
7. json

8. cer
9. mp3
10. aac
11. m4a
12. mp4
13. wav
14. ogg
15. silk

配置

最近更新时间：2024-04-10 17:06:21

全局配置

小程序根目录下的 app.json 文件用来对 TMF 小程序进行全局配置，可以通过设置来决定页面文件的路径、窗口表现、网络超时时间以及多 tab 等功能。

配置示例

以下是一个包含了部分常用配置选项的 app.json ：

```
{
 "pages": ["pages/index/index", "pages/logs/index"],
 "window": {
  "navigationBarTitleText": "Demo"
 },
 "tabBar": {
  "list": [
   {
    "pagePath": "pages/index/index",
    "text": "首页"
   },
   {
    "pagePath": "pages/logs/logs",
    "text": "日志"
   }
  ]
 },
 "networkTimeout": {
  "request": 10000,
  "downloadFile": 10000
 },
 "debug": true,
 "navigateToMiniProgramAppIdList": ["tmfq0nbl3hxxlv2w7u"],
 "groupIdList":["123456","34356576","457658769"]
}
```

app.json配置项列表

属性	类型	必填	描述
pages	String	是	页面路径列表

	Array		
window	Object	否	全局的默认窗口表现
tabBar	Object	否	底部 tab 栏表现
networkTimeout	Object	否	网络超时时间
requiredBackgroundModes	String Array	否	需要后台使用的能力，如**音乐播放**
navigateToMiniProgramAppIdList	String Array	否	需要跳转的小程序列表，详见 navigateToMiniProgram
permission	Object	否	小程序接口权限相关设置
darkmode	Boolean	否	获取当前手机是否为夜间模式

pages

用于指定小程序由哪些页面组成，每一项都对应一个页面的路径 + 文件名信息。文件名不需要写文件后缀，框架会自动去寻找对于位置的 `.json` **,** `.js` **，** `.wxml` **，** `.wxss` **四个文件进行处理。**
数组的第一项代表小程序的初始页面（首页）。小程序中新增/减少页面，都需要对 pages 数组进行修改。
如开发目录为：

```
├── app.js
├── app.json
├── app.wxss
├── pages
│   ├── index
│   │   ├── index.wxml
│   │   ├── index.js
│   │   ├── index.json
│   │   └── index.wxss
│   └── logs
│       ├── logs.wxml
│       └── logs.js
└── utils
```

则需要在 `app.json` **中写：**

```
{
  "pages": ["pages/index/index", "pages/logs/logs"]
}
```

window

用于设置小程序的状态栏、导航条、标题、窗口背景色。

属性	类型	默认值	描述
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，如 #000000
navigationBarTextStyle	String	white	导航栏标题颜色，仅支持 black / white
navigationBarTitleText	String	-	导航栏标题文字内容
navigationStyle	String	default	导航栏样式，仅支持以下值： • default：默认样式； • custom：自定义导航栏，只保留右上角胶囊按钮； • hide：自定义导航栏，可以支持隐私导航栏和胶囊按钮。
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	String	dark	下拉 loading 的样式，仅支持 dark / light
backgroundColorTop	String	#ffffff	顶部窗口的背景色，仅 iOS 支持
backgroundColorBottom	String	#ffffff	底部窗口的背景色，仅 iOS 支持
enablePullDownRefresh	Boolean	false	是否开启当前页面的下拉刷新。详见 Page.onPullDownRefresh
customNavigateBack	Boolean	false	是否需要拦截页面的默认返回（单击 tabBar 返回或侧滑或 back 键返回）结合Page.onCustomBack 使用
pageOrientation	String	portrait	屏幕旋转设置，支持 auto / portrait / landscape。详见 响应显示区域变化

- 注1：HexColor（十六进制颜色值），如"#ff00ff"。
- 注2：关于 `navigationStyle`

如 app.json：

```
{
 "window": {
  "navigationBarBackgroundColor": "#ffffff",
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "TMF接口功能演示",
  "backgroundColor": "#eeeeee",
  "backgroundTextStyle": "light"
 }
}
```

tabBar

如果小程序是一个多 tab 应用（客户端窗口的底部或顶部有 tab 栏可以切换页面），可以通过 tabBar 配置项指定 tab 栏的表现，以及 tab 切换时显示的对应页面。

属性	类型	必填	默认值	描述
color	HexColor	是	–	tab 上的文字默认颜色，仅支持十六进制颜色
selectedColor	HexColor	是	–	tab 上的文字选中时的颜色，仅支持十六进制颜色
backgroundColor	HexColor	是	–	tab 的背景色，仅支持十六进制颜色
borderStyle	String	否	black	tabbar上边框的颜色， 仅支持 black / white
list	Array	是	–	tab 的列表，最少2个 tab、最多5个 tab
position	String	否	bottom	tabBar的位置，仅支持 bottom / top
custom	Boolean	否	false	自定义 tabBar，详情请参见 自定义 tabBar

其中 list 接受一个数组，**只能配置最少2个、最多5个 tab**。tab 按数组的顺序排序，每个项都是一个对象，其属性值如下：

属性	类型	必填	说明
pagePath	String	是	页面路径，必须在 pages 中先定义
text	String	是	tab 上按钮文字

iconPath	String	否	图片路径，icon 大小限制为40kb，建议尺寸为 81px * 81px，不支持网络图片
selectedIconPath	String	否	选中时的图片路径，icon 大小限制为 40kb，建议尺寸为 81px * 81px，不支持网络图片。当 postion 为 top 时，不显示 icon。

disableSlideCloseGesture

是否禁用侧滑关闭小程序的手势。默认为 false；

networkTimeout

各类网络请求的超时时间，单位均为毫秒。

属性	类型	必填	默认值	描述
request	Number	否	60000	request 的超时时间，单位：毫秒
uploadFile	Number	否	60000	uploadFile 的超时时间，单位：毫秒

requiredBackgroundModes

申明需要后台运行的能力，类型为数组。目前支持以下项目：
`audio`：后台音乐播放。
示例代码：

```
{
  "pages": ["pages/index/index"],
  "requiredBackgroundModes": ["audio"]
}
```

> **说明：**
> 在此处申明了后台运行的接口，开发版和体验版上可以直接生效，正式版还需通过审核。

navigateToMiniProgramAppIdList

当小程序需要使用 navigateToMiniProgram 接口跳转到其他小程序时，需要先在配置文件中声明需要跳转的小程序 appId 列表，最多允许填写 10 个。

permission

小程序接口权限相关设置。字段类型为 Object，结构为：

属性	类型	必填	默认值	描述
scope.userLocation	PermissionObject	否	-	位置相关权限声明

PermissionObject 结构

属性	类型	必填	默认值	说明
desc	string	是	-	小程序获取权限时展示的接口中文用途说明。最长80个字符
desc-en	string	是	-	小程序获取权限时展示的接口英文用途说明。最长80个字符
desc-adesc-ar	string	是	-	小程序获取权限时展示的接口阿语用途说明。最长80个字符

示例代码：

```
{
 "pages": ["pages/index/index"],
 "permission": {
  "scope.userLocation": {
   "desc": "你的位置信息将用于小程序位置接口的效果展示",
   "desc-en": "where are you? do you know",
   "desc-ar":"انا من السعودية"
  }
 }
}
```

页面配置

每一个小程序页面也可以使用 .json 文件来对本页面的窗口表现进行配置。
页面的配置，只能设置 app.json 中部分 window 配置项的内容，页面中配置项会覆盖 app.json 的 window 中相同的配置项。

配置示例

```
{
 "navigationBarBackgroundColor": "#ffffff",
 "navigationBarTextStyle": "black",
```

```
  "navigationBarTitleText": "TMF接口功能演示",
  "backgroundColor": "#eeeeee",
  "backgroundTextStyle": "light"
}
```

页面配置项列表

属性	类型	默认值	描述
navigationBarBackgroundColor	HexColor	#000000	导航栏背景颜色，如 #000000
navigationBarTextStyle	String	white	导航栏标题颜色，仅支持 black / white
navigationBarTitleText	String	-	导航栏标题文字内容
navigationStyle	String	default	导航栏样式，仅支持以下值： • default：默认样式； • custom：自定义导航栏，只保留右上角胶囊按钮； • hide：自定义导航栏，可以支持隐私导航栏和胶囊按钮。
backgroundColor	HexColor	#ffffff	窗口的背景色
backgroundTextStyle	String	dark	下拉 loading 的样式，仅支持 dark / light
backgroundColorTop	String	#ffffff	顶部窗口的背景色，仅 iOS 支持
backgroundColorBottom	String	#ffffff	底部窗口的背景色，仅 iOS 支持
enablePullDownRefresh	Boolean	false	是否全局开启下拉刷新。详见 Page.onPullDownRefresh
pageOrientation	String	portrait	屏幕旋转设置，支持 auto / portrait / landscape 详见 响应显示区域变化

逻辑层
逻辑层说明

最近更新时间：2024-03-21 15:51:52

逻辑层 App Service

小程序开发框架的逻辑层使用 JavaScript 引擎为小程序提供开发者 JavaScript 代码的运行环境以及TMF小程序的特有功能。
逻辑层将数据进行处理后发送给视图层，同时接受视图层的事件反馈。
开发者写的所有代码最终将会打包成一份 JavaScript 文件，并在小程序启动的时候运行，直到小程序销毁。这一行为类似 ServiceWorker，所以逻辑层也称之为 App Service。
在 JavaScript 的基础上，我们增加了一些功能，以方便小程序的开发：

- 增加 App 和 Page 方法，进行程序和页面的注册。
- 增加 getApp 和 getCurrentPages 方法，分别用来获取 App 实例和当前页面栈。
- 提供丰富的 API，如TMF用户数据，扫一扫，支付等TMF特有能力。
- 每个页面有独立的 作用域，并提供 模块化 能力。

> ⚠ **注意：**
> 小程序框架的逻辑层并非运行在浏览器中，因此 JavaScript 在 web 中一些能力都无法使用，如 window，document 等。

注册小程序 App

App（Object）

App() 函数用来注册一个小程序。接受一个 Object 参数，其指定小程序的生命周期回调等。
App() 必须在 app.js 中调用，必须调用且只能调用一次。不然会出现无法预期的后果。
Object 参数说明：

属性	类型	描述	触发时机
onLaunch	function	生命周期回调—监听小程序初始化	小程序初始化完成时（全局只触发一次）
onShow	function	生命周期回调—监听小程序初显示	小程序启动，或从后台进入前台显示时
onHide	function	生命周期回调—监听小程序初隐藏	小程序从前台进入后台时

onError	function	错误监听函数	小程序发生脚本错误，或者 api 调用失败时触发，会带上错误信息
onPageNotFound	function	页面不存在监听函数	小程序要打开的页面不存在时触发，会带上页面信息回调该函数
其他	any	开发者可以添加任意的函数或数据到 `Object` 参数中，用 `this` 可以访问	–

前台、后台定义：当用户单击右上角关闭，或者按了设备 Home 键离开宿主 App，小程序并没有直接销毁，而是进入了后台；当再次进入宿主 App 或再次打开小程序，又会从后台进入前台。

> **说明：**
> 只有当小程序进入后台一定时间，或者系统资源占用过高，才会被真正的销毁。

关闭小程序： 当用户从扫一扫、转发等入口（ 场景值为1007, 1008, 1011, 1025 ）进入小程序，且没有置顶小程序的情况下退出，小程序会被销毁。

示例代码：

```
App({
  onLaunch(options) {
    // Do something initial when launch.
  },
  onShow(options) {
    // Do something when show.
  },
  onHide() {
    // Do something when hide.
  },
  onError(msg) {
    console.log(msg)
  },
  globalData: 'I am global data'
})
```

onLaunch(Object)

- 小程序初始化完成时触发，全局只触发一次。参数也可以使用 getLaunchOptionsSync 获取。
- **参数及说明：** 与 getLaunchOptionsSync 一致。

onShow(Object)

小程序启动，或从后台进入前台显示时触发。

onHide()

小程序从前台进入后台时触发。

onError(String error)

- 小程序发生脚本错误或 API 调用报错时触发。也可以使用 onError 绑定监听。
- **参数及说明：**与 onError 一致

onPageNotFound(Object)

- 小程序要打开的页面不存在时触发。
- **示例代码：**

```
App({
  onPageNotFound(res) {
    wx.redirectTo({
      url: 'pages/...'
    }) // 如果是 tabbar 页面，请使用 wx.switchTab
  }
})
```

getApp（Object）

- 全局的 `getApp()` 函数可以用来获取到小程序 `App` 实例。
- Object 参数说明：

字段	类型	说明	最低版本
allowDefault	boolean	在 `App` 未定义时返回默认实现。当 App 被调用时，默认实现中定义的属性会被覆盖合并到 App 中。	–

- **示例代码：**

```
// other.js
const appInstance = getApp()
console.log(appInstance.globalData) // I am global data
```

> ⚠ **注意：**

- 不要在定义于 App() 内的函数中调用 getApp() ，使用 this 就可以拿到 app 实例。
- 通过 getApp() 获取实例之后，不要私自调用生命周期函数。

场景值

获取场景值

当前支持的场景值有：
对于小程序，可以在 App 的 onLaunch 和 onShow ，或 getLaunchOptionsSync 中获取上述场景值。

返回来源信息的场景

部分场景值下还可以获取来源应用、公众号或小程序的 appId。

场景值	场景	appId含义
1020	公众号 profile 页相关小程序列表	来源公众号
1035	公众号自定义菜单	来源公众号
1036	App 分享消息卡片	来源 App
1037	小程序打开小程序	来源小程序
1038	从另一个小程序返回	来源小程序
1043	公众号模板消息	来源公众号

说明：
由于 Android 系统限制，目前还无法获取到按 Home 键退出到桌面，然后从桌面再次进小程序的场景值，对于这种情况，会保留上一次的场景值。

注册页面

注册页面相关操作，可参见 注册页面 。

页面路由

在小程序中所有页面的路由全部由框架进行管理。

页面栈

框架以栈的形式维护了当前的所有页面。 当发生路由切换的时候，页面栈的表现如下：

路由方式	页面栈表现

初始化	新页面入栈
打开新页面	新页面入栈
页面重定向	当前页面入栈，新页面入栈
页面返回	页面不断出栈，直到目标返回页
Tab切换	页面全部出栈，只留下新的Tab页面
重加载	页面全部出栈，只留下新的页面

getCurrentPages()

`getCurrentPages()` 函数用于获取当前页面栈的实例，以数组形式按栈的顺序给出，第一个元素为首页，最后一个元素为当前页面。

> ⚠ **注意：**
> - **不要尝试修改页面栈，会导致路由以及页面状态错误。**
> - **不要在 `App.onLaunch` 的时候调用 `getCurrentPages()`，此时 `page` 还没有生成。**

路由方式

对于路由的触发方式以及页面生命周期函数如下：

路由方式	触发时机	路由前页面	路由后页面
初始化	小程序打开的第一个页面	-	onLoad, onShow
打开新页面	调用 API wx.navigateTo 或使用组件 <navigator open-type="navigateTo"/>	onHide	onLoad, onShow
页面重定向	调用 API wx.redirectTo 或使用组件 <navigator open-type="redirectTo"/>	onUnload	onLoad, onShow
页面返回	调用 API wx.navigateBack 或使用组件<navigator open-type="navigateBack">或用户按左上角返回按钮	onUnload	onShow
Tab 切换	调用 API wx.switchTab 或使用组件 <navigator open-type="switchTab"/> 或用户切换 Tab	-	各种情况请参考下表
重启动	调用 API wx.reLaunch 或使用组件 <navigator open-type="reLaunch"/>	onUnload	onLoad, onShow

Tab 切换对应的生命周期（以 A、B 页面为 Tabbar 页面，C 是从 A 页面打开的页面，D 页面是从 C 页面打开的页面为例）：

当前页面	路由后页面	出发的生命周期（按顺序）
A	A	Nothing happend
A	B	A.onHide(), B.onLoad(), B.onShow()
A	B（再次打开）	A.onHide(), B.onShow()
C	A	C.onUnload(), A.onShow()
C	B	C.onUnload(), B.onLoad(), B.onShow()
D	B	D.onUnload(), C.onUnload(), B.onLoad(), B.onShow()
D（从转发进入）	A	D.onUnload(), A.onLoad(), A.onShow()
D（从转发进入）	B	D.onUnload(), B.onLoad(), B.onShow()

说明：

- navigateTo, redirectTo 只能打开非 tabBar 页面。
- switchTab 只能打开 tabBar 页面。
- reLaunch 可以打开任意页面。
- 页面底部的 tabBar 由页面决定，即只要是定义为 tabBar 的页面，底部都有 tabBar。
- 调用页面路由带的参数可以在目标页面的 onLoad 中获取。

文件作用域

在 JavaScript 文件中声明的变量和函数只在该文件中有效；不同的文件中可以声明相同名字的变量和函数，不会互相影响。
通过全局函数 getApp() 可以获取全局的应用实例，如果需要全局的数据可以在 App() 中设置，如：

```
// app.js
App({
  globalData: 1
})
```

```
// a.js
// The localValue can only be used in file a.js.
```

```
const localValue = 'a'
// Get the app instance.
const app = getApp()
// Get the global data and change it.
app.globalData++
```

```
// b.js
// You can redefine localValue in file b.js, without interference with the localValue in
a.js.
const localValue = 'b'
// If a.js it run before b.js, now the globalData shoule be 2.
console.log(getApp().globalData)
```

模块化

可以将一些公共的代码抽离成为一个单独的 js 文件，作为一个模块。模块只有通过 `module.exports` 或者 `exports` 才能对外暴露接口。

需要注意的是：

- `exports` 是 `module.exports` 的一个引用，因此在模块里边随意更改 `exports` 的指向会造成未知的错误。所以更推荐开发者采用 `module.exports` 来暴露模块接口，除非您已经清晰知道这两者的关系。
- 小程序目前不支持直接引入 `node_modules`，开发者需要使用到 `node_modules` 时候建议拷贝出相关的代码到小程序的目录中或者使用小程序支持的 npm 功能。

```
// common.js
function sayHello(name) {
  console.log(`Hello ${name} !`)
}
function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}

module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye
```

在需要使用这些模块的文件中，使用 `require(path)` 将公共代码引入。

```
const common = require('common.js')
Page({
  helloMINA() {
    common.sayHello('MINA')
  },
```

```
  goodbyeMINA() {
    common.sayGoodbye('MINA')
  }
})
```

> **说明：**
> `require` 暂时不支持绝对路径。

API

事件监听

约定使用以“on”开头的 API 来监听事件是否触发，例如：wx.onCompassChange 等。
这类 API 接受一个回调函数作为参数，当事件触发时会调用这个回调函数，并将相关数据以参数形式传入。
代码示例：

```
wx.onCompassChange(function (res) {
  console.log(res.direction)
})
```

同步

我们约定，以 `Sync` 结尾的 API 都是同步 API， 如 wx.setStorageSync，wx.getSystemInfoSync 等。
此外，也有一些其他的同步 API，详情参见 API 文档中的相关说明。
同步 API 的执行结果可以通过函数返回值直接获取，如果执行出错会抛出异常。
代码示例：

```
try {
  wx.setStorageSync('key', 'value')
} catch (e) {
  console.error(e)
}
```

异步

大多数 API 都是异步 API，如 wx.request，wx.login 等。这类 API 接口通常都接受一个 Object 类型的参数，这个参数都支持按需指定以下字段来接收接口调用结果：
Object 参数说明

参数名	类型	必填	说明

success	function	否	接口调用成功的回调函数
fail	function	否	接口调用成功的回调函数
complete	function	否	接口调用结束的回调函数（调用成功、失败都会执行）
其他	any	-	接口定义的其他参数

回调函数的参数

`success`，`fail`，`complete` **函数调用时会传入一个** `Object` **类型参数，包含以下字段：**

属性	类型	说明
errMsg	string	错误信息，如果调用成功返回 ${apiName}:ok
errCode	number	错误码，仅部分 API 支持，具体含义请参考对应 API 文档，成功时为 0。
其他	any	接口返回的其他数据

代码示例：

```
wx.login({
  success(res) {
    console.log(res.code)
  }
})
```

WXS

WXS 相关操作可参见：WXS 详情 。

注册页面

最近更新时间：2024-04-10 15:12:32

页面Page

Page（Object）构造器

`Page(Object)` 函数用来注册一个页面。接受一个 `Object` 类型参数，其指定页面的初始数据、生命周期回调、事件处理函数等。

- **Object参数说明：**

属性	类型	描述
data	object	页面的初始数据
onLoad	fuction	生命周期回调—监听页面加载
onShow	fuction	生命周期回调—监听页面显示
onReady	fuction	生命周期回调—监听页面初次渲染完成
onHide	fuction	生命周期回调—监听页面隐藏
onUnload	fuction	生命周期回调—监听页面卸载
onPullDownRefresh	fuction	监听用户下拉动作
onReachBottom	fuction	页面上拉触底事件的处理函数
onPageScroll	fuction	页面滚动触发事件的处理函数
onAddToFavorites	fuction	用户点击右上角菜单的收藏按钮（基础库支持1.4.0及以上版本）
onShareAppMessage	fuction	用户点击右上角菜单的转发按钮
onResize	fuction	页面尺寸改变时触发，详见 响应显示区域变化
onCustomBack	function	开发者拦截页面的默认返回时回调
onTabItemTap	function	当前是 tab 页时，点击 tab 时触发

其他	any	开发者可以添加任意的函数或数据到 Object 参数中，在页面的函数中用 this 可以访问

- **示例代码：**

```
// index.js
Page({
  data: {
    text: 'This is page data.'
  },
  onLoad(options) {
    // Do some initialize when page load.
  },
  onReady() {
    // Do something when page ready.
  },
  onShow() {
    // Do something when page show.
  },
  onHide() {
    // Do something when page hide.
  },
  onUnload() {
    // Do something when page close.
  },
  onPullDownRefresh() {
    // Do something when pull down.
  },
  onReachBottom() {
    // Do something when page reach bottom.
  },
  onShareAppMessage() {
    // return custom share data when user share.
  },
  onAddToFavorites: function(res) {
    // return custom favorite data.
  },
  onPageScroll() {
    // Do something when page scroll
  },
  onResize() {
    // Do something when page resize
  },
  onTabItemTap(item) {
```

```
    console.log(item.index)
    console.log(item.pagePath)
    console.log(item.text)
  },
  // Event handler.
  viewTap() {
    this.setData({
      text: 'Set some data for updating view.'
    }, function () {
      // this is setData callback
    })
  },
  customData: {
    hi: 'MINA'
  }
})
```

除了 Page ，作为高级用法，页面可以像自定义组件一样使用 component 来创建，这样就可以使用自定义组件的特性，如 behaviors 等。具体细节，请参见 Component 构造器 章节。

初始数据

data

data 是页面第一次渲染使用的初始数据。

页面加载时， data 将会以 JSON 字符串的形式由逻辑层传至渲染层，因此 data 中的数据必须是可以转成 JSON 的类型：字符串，数字，布尔值，对象，数组。

渲染层可以通过 WXML 对数据进行绑定。

示例代码：

```
<view>{{text}}</view>
<view>{{array[0].msg}}</view>
```

```
Page({
  data: {
    text: 'init data',
    array: [{msg: '1'}, {msg: '2'}]
  }
})
```

生命周期回调函数

生命周期的触发以及 页面路由 方式。

onLoad(Object query)

页面加载时触发。一个页面只会调用一次，可以在 onLoad 的参数中获取打开当前页面路径中的参数。

参数说明：

名称	类型	说明
query	object	打开当前页面路径中的参数

onShow()

页面显示/切入前台时触发。

onReady()

页面初次渲染完成时触发。一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互。

> ⚠ **注意：**
> 对界面内容进行设置的 API，如 wx.setNavigationBarTitle，请在 onReady 之后进行。详见 生命周期。

onHide()

页面隐藏/切入后台时触发。如 `navigateTo` 或底部 `tab` 切换到其他页面，小程序切入后台等。

onUnload()

页面卸载时触发。如 `redirectTo` 或 `navigateBack` 到其他页面时。

页面事件处理函数

onPullDownRefresh()

监听用户下拉刷新事件。

- 需要在 `app.json` 的 window 选项中或 页面配置 中开启 `enablePullDownRefresh`。
- 可以通过 wx.startPullDownRefresh 触发下拉刷新，调用后触发下拉刷新动画，效果与用户手动下拉刷新一致。
- 当处理完数据刷新后，wx.stopPullDownRefresh 可以停止当前页面的下拉刷新。

onReachBottom()

监听用户上拉触底事件。

- 可以在 `app.json` 的 window 选项中或 页面配置 中设置触发距离 `onReachBottomDistance`。

- 在触发距离内滑动期间，本事件只会被触发一次。

onPageScroll(Object)

监听用户滑动页面事件。
Object 参数说明：

属性	类型	说明
scrollTop	number	页面在垂直方向已滚动的举例（单位：px）

> **注意：**
> - 请只在需要的时候才在 page 中定义此方法，不要定义空方法。以减少不必要的事件派发对渲染层-逻辑层通信的影响。
> - 请避免在 onPageScroll 中过于频繁的执行 setData 等引起逻辑层-渲染层通信的操作。尤其是每次传输大量数据，会影响通信耗时。

onAddToFavorites(Object)

监听用户单击页面内收藏按钮（button 组件 open-type="addToFavorites"）或右上角菜单收藏按钮的行为，并自定义收藏内容。基础库支持1.4.0及以上版本。

- **Object 参数说明：**

参数	类型	说明
fromw	string	转发事件来源。 • button：页面内转发按钮 • menu：右上角转发菜单
webviewUrl	string	页面中包含 <web-view> 组件时，返回当前 <web-view> 的url

需要在事件处理函数中返回一个 Object 对象，以便自定义收藏内容，需返回内容如下：

- **自定义参数内容**

字段	类型	说明	默认值
title	string	参数标题	当前小程序名称
imageUrl	string	自定义图片路径，可以是本地文件路径、代码包文件路径或者网络图片路径。支持 PNG 及 JPG	当前小程序 logo
query	stri	页面查询参数，如 a=1&b=2	当前页面的查询参

	ng		数

- **示例代码：**

```
Page({
  onAddToFavorites: function(res) {
    console.log('webviewUrl', res.webviewUrl);
    console.log('from', res.from);
    return {
      title: '添加收藏标题',
      imageUrl: '',
      query: 'a=1&b=2',
    }
  },
})
```

onShareAppMessage(Object)

监听用户单击右上角菜单"分享给好友"、"分享到空间"按钮的行为，并自定义转发内容。

> **说明：**
> **定义了此事件处理函数时单击右上角菜单会显示"分享给好友"、"分享到空间"按钮，无需调用 showShareMenu ，hideShareMenu 无效。**

- **Object 参数说明：**

参数	类型	说明
from	string	转发事件来源。 button：页面内转发按钮； menu：右上角转发菜单
target	object	如果 from 值是 button，则 target 是触发这次转发事件的 button，否则为 undefined
webViewUrl	string	页面中包含 <web-view> 组件时，返回当前 <web-view> 的 url
shareTarget	string	用户单击分享到哪里，详见下面的 shareTarget 参数说明

需要在事件处理函数中返回一个 Object 对象，以便自定义转发内容，返回内容如下：

- shareTarget 参数说明：

取值	说明

0	分享到 QQ 好友
1	分享到 QQ 空间
2	从当前聊天窗口打开，快速分享到当前聊天窗口
3	分享到微信好友
4	分享到微信朋友圈
5	分享面板分享到最近联系人
6	分享到快捷分享好友列表

- **自定义转发内容**

字段	类型	说明	默认值	最低版本
title	string	转发标题	当前小程序名称	–
path	string	转发路径	当前页面 path ，必须是以 / 开头的完整路径	–
imageUrl	string	自定义图片路径，可以是本地文件路径、代码包文件路径或者网络图片路径。支持 PNG 及 JPG。默认分享模版显示图片长宽比是 5:4。	使用默认截图	–
shareType	string	指定分享的类型，具体查看下面 shareType 说明	"miniapp"	1.4.0

- shareType

  用来指定分享的类型，目前分享有两种，一种是分享小程序，另外一种是分享一张图片。

值	说明
"miniapp"	以小程序的形式分享，title、path、imageUrl、generalWebpageUrl、entryDataHash、shareTemplateId、shareTemplateData 参数会生效
"picture"	以图片的形式分享，imageUrl、entryDataHash 参数会生效

- **示例代码：**

```
Page({
  onShareAppMessage(res) {
    if (res.from === 'button') {
      // 来自页面内转发按钮
      console.log(res.target)
    }
    return {
      title: '自定义转发标题',
      path: '/page/user?id=123'
    }
  }
})
```

onResize(object)

小程序屏幕旋转时触发。详情请参见 响应显示区域变化 。

onTabItemTap(Object)

单击 tab 时触发

- **Object 参数说明：**

参数	类型	说明
index	string	被单击的 tabItem 的序号，从0开始
pagePath	string	被单击 tabItem 的页面路径
text	string	被单击 tabItem 的按钮文字

- **示例代码：**

```
Page({
  onTabItemTap(item) {
    console.log(item.index)
    console.log(item.pagePath)
    console.log(item.text)
  }
})
```

组件事件处理函数

`Page` 中还可以定义组件事件处理函数。在渲染层的组件中加入 事件绑定 ，当事件被触发时，就会执行 Page 中定义的事件处理函数。

示例代码：

```
<view bindtap="viewTap">click me</view>
```

```
Page({
  viewTap() {
    console.log('view tap')
  }
})
```

route

Page.route

到当前页面的路径，类型为 String 。

```
Page({
  onShow() {
    console.log(this.route)
  }
})
```

setData

Page.prototype.setData(Object data, Function callback)

setData 函数用于将数据从逻辑层发送到视图层（异步），同时改变对应的 this.data 的值（同步）。

参数说明

字段	类型	必填	描述
data	object	是	这次要改变的数据
callback	function	否	setData 引起的界面更新渲染完毕后的回调函数

Object 以 key: value 的形式表示，将 this.data 中的 key 对应的值改变成 value 。
其中 key 可以以数据路径的形式给出，支持改变数组中的某一项或对象的某个属性，如 array[2].message ，a.b.c.d ，并且不需要在 this.data 中预先定义。

> **说明：**

- 直接修改 this.data 而不调用 this.setData 是无法改变页面的状态的，还会造成数据不一致。
- 仅支持设置可 JSON 化的数据。
- 单次设置的数据不能超过1024kB，请尽量避免一次设置过多的数据。
- 请不要把 data 中任何一项的 value 设为 undefined ，否则这一项将不被设置并可能遗留一些潜在问题。

- **示例代码：**

```
<!--index.wxml-->
<view>{{text}}</view>
<button bindtap="changeText">Change normal data</button>
<view>{{num}}</view>
<button bindtap="changeNum">Change normal num</button>
<view>{{array[0].text}}</view>
<button bindtap="changeItemInArray">Change Array data</button>
<view>{{object.text}}</view>
<button bindtap="changeItemInObject">Change Object data</button>
<view>{{newField.text}}</view>
<button bindtap="addNewField">Add new data</button>
```

```
// index.js
Page({
  data: {
    text: 'init data',
    num: 0,
    array: [{text: 'init data'}],
    object: {
      text: 'init data'
    }
  },
  changeText() {
    // this.data.text = 'changed data' // 不要直接修改 this.data
    // 应该使用 setData
    this.setData({
      text: 'changed data'
    })
  },
  changeNum() {
    // 或者，可以修改 this.data 之后马上用 setData 设置一下修改了的字段
    this.data.num = 1
    this.setData({
      num: this.data.num
```

```
    })
  },
  changeItemInArray() {
    // 对于对象或数组字段，可以直接修改一个其下的子字段，这样做通常比修改整个对象或数
组更好
    this.setData({
      'array[0].text': 'changed data'
    })
  },
  changeItemInObject() {
    this.setData({
      'object.text': 'changed data'
    })
  },
  addNewField() {
    this.setData({
      'newField.text': 'new data'
    })
  }
})
```

setData 使用注意事项

1. **setData 的流程**

   setData 的过程，大致可以分成如下几个阶段：

   - 逻辑层虚拟 DOM 树的遍历和更新，触发组件生命周期和 observer 等。
   - 将 data 从逻辑层传输到视图层。
   - 视图层虚拟 DOM 树的更新、真实 DOM 元素的更新并触发页面渲染更新。

2. **使用建议**

   2.1 **data 应只包括渲染相关的数据**

   setData 应只用来进行渲染相关的数据更新。用 setData 的方式更新渲染无关的字段，会触发额外的渲染流程，或者增加传输的数据量，影响渲染耗时。

   - 页面或组件的 data 字段，应用来存放和页面或组件渲染相关的数据（即直接在 wxml 中出现的字段）。
   - 页面或组件渲染无关的数据，应挂在非 data 的字段下，如 this.userData = {userId: 'xxx'}。
   - 避免在 data 中包含渲染无关的业务数据。
   - 避免使用 data 在页面或组件方法间进行数据共享。

   2.2 **控制 setData 的频率**

   每次 setData 都会触发逻辑层虚拟 DOM 树的遍历和更新，也可能会导致触发一次完整的页面渲染流程。过于频繁（毫秒级）的调用 setData，会导致以下后果：

- 逻辑层 JS 线程持续繁忙，无法正常响应用户操作的事件，也无法正常完成页面切换。
- 视图层 JS 线程持续处于忙碌状态，逻辑层 -> 视图层通信耗时上升，视图层收到消息的延时较高，渲染出现明显延迟。
- 视图层无法及时响应用户操作，用户滑动页面时感到明显卡顿，操作反馈延迟，用户操作事件无法及时传递到逻辑层，逻辑层亦无法及时将操作处理结果及时传递到视图层。

因此，开发者在调用 setData 时要注意：

- 仅在需要进行页面内容更新时调用 setData。
- 对连续的 setData 调用尽可能的进行合并。
- 避免不必要的 setData。
- 避免以过高的频率持续调用 setData，例如毫秒级的倒计时。
- 避免在 onPageScroll 回调中每次都调用 setData。

2.3 setData 应只传发生变化的数据

setData 的数据量会影响数据拷贝和数据通讯的耗时，增加页面更新的开销，造成页面更新延迟。

- setData 应只传入发生变化的字段。
- 建议以数据路径形式改变数组中的某一项或对象的某个属性，如 this.setData({'array[2].message': 'newVal', 'a.b.c.d': 'newVal'})，而不是每次都更新整个对象或数组。
- 不要在 setData 中偷懒一次性传所有data：this.setData(this.data)。

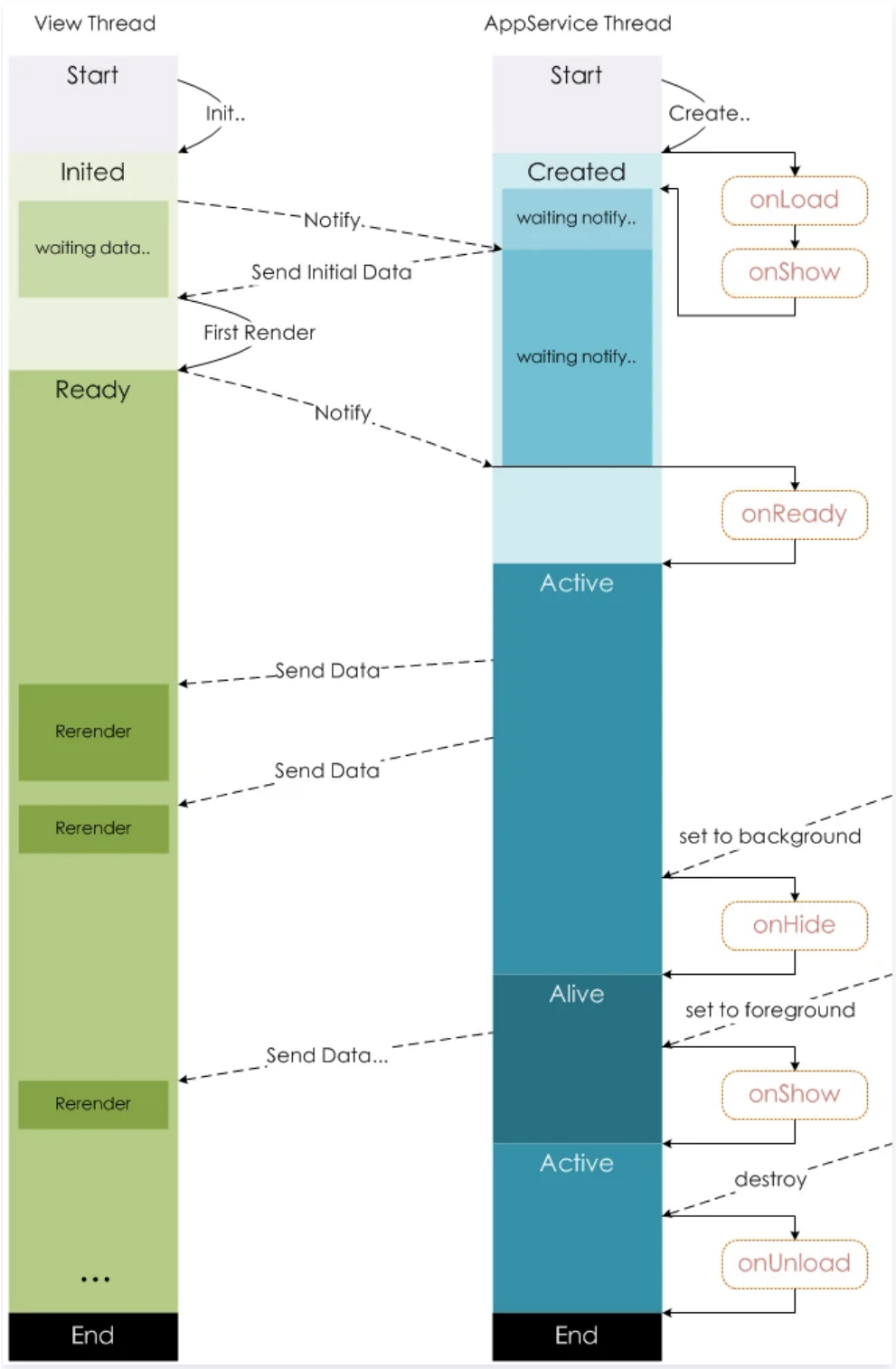
2.4 控制后台态页面的 setData

由于小程序逻辑层是单线程运行的，后台态页面去 setData 也会抢占前台页面的运行资源，且后台态页面的渲染用户是无法感知的，会产生浪费。在某些平台上，小程序渲染层各 WebView 也是共享同一个线程，后台页面的渲染和逻辑执行也会导致前台页面的卡顿。

- 页面切后台后的更新操作，应尽量避免，或延迟到页面 onShow 后延迟进行。
- 避免在切后台后仍进行高频的 setData，例如倒计时更新。

生命周期

下图说明了 Page 实例的生命周期。


View Thread
AppService Thread
Start
Init..
Inited
waiting data..
Notify
Send Initial Data
First Render
Ready
Notify
Rerender
Send Data
Rerender
Send Data
Rerender
Send Data...
...
End
Start
Create..
Created
waiting notify..
onLoad
onShow
waiting notify..
onReady
Active
set to background
onHide
Alive
set to foreground
onShow
Active
destroy
onUnload
End

WXS

最近更新时间：2024-03-21 15:51:52

WXS（WeiXin Script）是小程序的一套脚本语言，结合 WXML，可以构建出页面的结构。

注意事项

- WXS 不依赖于运行时的基础库版本，可以在所有版本的小程序中运行。
- WXS 与 javascript 是不同的语言，有自己的语法，并不和 javascript 一致。
- WXS 的运行环境和其他 javascript 代码是隔离的，WXS 中不能调用其他 javascript 文件中定义的函数，也不能调用小程序提供的 API。
- WXS 函数不能作为组件的事件回调。
- 由于运行环境的差异，在 iOS 设备上小程序内的 WXS 会比 javascript 代码快2 ~ 20倍。在 Android 设备上二者运行效率无差异。

以下是一些使用 WXS 的简单示例：

页面渲染

```
<!--wxml-->
<wxs module="m1">
 var msg = "hello world";
 module.exports.message = msg;
</wxs>

<view>{{m1.message}}</view>
```

页面输出：

```
hello world
```

数据处理

```
// page.js
Page({
 data: {
  array: [1, 2, 3, 4, 5, 1, 2, 3, 4],
 },
})
```

```
<!--wxml-->
<!-- 下面的 getMax 函数，接受一个数组，且返回数组中最大的元素的值 -->
<wxs module="m1">
 var getMax = function(array) {
  var max = undefined;
  for (var i = 0; i < array.length; ++i) {
    max = max === undefined ? array[i] : (max >= array[i] ? max : array[i]);
  }
  return max;
 }
 module.exports.getMax = getMax;
</wxs>

<!-- 调用 wxs 里面的 getMax 函数，参数为 page.js 里面的 array -->
<view>{{m1.getMax(array)}}</view>
```

页面输出：

```
5
```

模块

WXS 模块

WXS 代码可以编写在 wxml 文件中的 `<wxs>` 标签内，或以 `.wxs` 为后缀名的文件内。

模块

每一个 `.wxs` 文件和 `<wxs>` 标签都是一个单独的模块。
每个模块都有自己独立的作用域。即在一个模块里面定义的变量与函数，默认为私有的，对其他模块不可见。
一个模块要想对外暴露其内部的私有变量与函数，只能通过 `module.exports` 实现。

.WXS 文件

在开发者工具里面，右键可以直接创建 `.wxs` 文件，在其中直接编写 WXS 脚本。
示例代码：

```
// /pages/comm.wxs

var foo = "'hello world' from comm.wxs"
var bar = function(d) {
 return d
```

```
}
module.exports = {
  foo: foo,
  bar: bar,
}
```

上述例子在 `/pages/comm.wxs` 的文件里面编写了 WXS 代码。该 `.wxs` 文件可以被其他的 `.wxs` 文件或 WXML 中的 `<wxs>` 标签引用。

module 对象

每个 `wxs` 模块均有一个内置的 `module` 对象。

属性

`exports`：通过该属性，可以对外共享本模块的私有变量与函数。

示例代码：

```
// /pages/tools.wxs

var foo = "'hello world' from tools.wxs"
var bar = function(d) {
  return d
}
module.exports = {
  FOO: foo,
  bar: bar,
}
module.exports.msg = 'some msg'
```

```
<!-- page/index/index.wxml -->

<wxs src="./../tools.wxs" module="tools" />
<view>{{tools.msg}}</view>
<view>{{tools.bar(tools.FOO)}}</view>
```

页面输出：

```
some msg
'hello world' from tools.wxs
```

require 函数

在 `.wxs` 模块中引用其他 `wxs` 文件模块，可以使用 `require` 函数。

引用的时候，要注意如下几点：

- 只能引用 `.wxs` 文件模块，且必须使用相对路径。
- `wxs` 模块为单例模式，即在第一次引用时自动初始化为单例对象。不论在多个页面、多个地方、多次引用，都使用同一个 `wxs` 模块对象。
- 如果一个 `wxs` 模块在定义之后，一直没有被引用，则该模块不会被解析与运行。

示例代码：

```
// /pages/tools.wxs

var foo = "'hello world' from tools.wxs"
var bar = function(d) {
  return d
}
module.exports = {
  FOO: foo,
  bar: bar,
}
module.exports.msg = 'some msg'
```

```
// /pages/logic.wxs

var tools = require('./tools.wxs')

console.log(tools.FOO)
console.log(tools.bar('logic.wxs'))
console.log(tools.msg)
```

```
<!-- /page/index/index.wxml -->

<wxs src="./../logic.wxs" module="logic" />
```

控制台输出：

```
'hello world' from tools.wxs
logic.wxs
some msg
```

<WXS> 标签

属性	类型	默认值

名		
module	string	当前 <wxs> 标签的模块名。必填字段
src	string	引用 .wxs 文件的相对路径。仅当本标签为**单闭合标签**或**标签的内容为空**时有效

- **module 属性**

  module 属性是当前 `<wxs>` 标签的模块名。在单个 wxml 文件内，建议其值唯一。有重复模块名则按照先后顺序覆盖（后者覆盖前者）。不同文件之间的 WXS 模块名不会相互覆盖。

  module 属性值的命名必须符合下面两个规则：

  - 首字符必须是：字母（a-zA-Z），下划线（_）
  - 剩余字符可以是：字母（a-zA-Z），下划线（_），数字（0-9）

  示例代码：

```
<!--wxml-->

<wxs module="foo">
  var some_msg = "hello world";
  module.exports = { msg : some_msg, }
</wxs>
<view>{{foo.msg}}</view>
```

  页面输出：

```
hello world
```

- **src 属性**

  src 属性可以用来引用其他的 `wxs` 文件模块。

  引用的时候，要注意如下几点：

  - 只能引用 `.wxs` 文件模块，且必须使用相对路径。
  - `wxs` 模块为单例模式，即在第一次引用时自动初始化为单例对象。不论在多个页面、多个地方、多次引用，都使用同一个 `wxs` 模块对象。
  - 如果一个 `wxs` 模块在定义之后，一直没有被引用，则该模块不会被解析与运行。

  示例代码：

```
// /pages/index/index.js

Page({
```

```
  data: {
    msg: "'hello wrold' from js",
  },
})
```

```
<!-- /pages/index/index.wxml -->

<wxs src="./../comm.wxs" module="some_comms"></wxs>
<!-- 也可以直接使用单标签闭合的写法 <wxs src="./../comm.wxs"
module="some_comms" /> -->

<!-- 调用 some_comms 模块里面的 bar 函数，且参数为 some_comms 模块里面的 foo --
>
<view>{{some_comms.bar(some_comms.foo)}}</view>
<!-- 调用 some_comms 模块里面的 bar 函数，且参数为 page/index/index.js 里面的 msg
-->
<view>{{some_comms.bar(msg)}}</view>
```

页面输出：

```
'hello world' from comm.wxs
'hello wrold' from js
```

上述例子在文件 `/page/index/index.wxml` 中通过 `<wxs>` 标签引用了 `/page/comm.wxs` 模块。

> ⚠ **注意：**
>
> - <wxs> 模块只能在定义模块的 WXML 文件中被访问到。使用 <include> 或 <import> 时，<wxs> 模块不会被引入到对应的 WXML 文件中。
> - <template> 标签中，只能使用定义该 <template> 的 WXML 文件中定义的 <wxs> 模块。

变量

概念

- WXS 中的变量均为值的引用。
- 没有声明的变量直接赋值使用，会被定义为全局变量。
- 如果只声明变量而不赋值，则默认值为 `undefined` 。
- var 表现与 javascript 一致，会有变量提升。

```
var foo = 1
```

```
var bar = 'hello world'
var i // i === undefined
```

上面代码，分别声明了 `foo` 、 `bar` 、 `i` 三个变量。然后， `foo` 赋值为数值 `1` ， `bar` 赋值为字符串 `"hello world"` 。

变量名

变量命名必须符合下面两个规则：

- 首字符必须是：字母（a-zA-Z），下划线（_）。
- 剩余字符可以是：字母（a-zA-Z），下划线（_）， 数字（0-9）。

保留标识符

以下标识符不能作为变量名：

```
delete
void
typeof
null
undefined
NaN
Infinity
var
if
else
true
false
require
this
function
arguments
return
for
while
do
break
continue
switch
case
default
```

注释

WXS 主要有 3 种注释的方法。

示例代码：

```
<!-- wxml -->
<wxs module="sample">
  // 方法一：单行注释
  /* 方法二：多行注释 */
  /* 方法三：结尾注释。即从 /* 开始往后的所有 WXS 代码均被注释 var a = 1; var b = 2; var
c = "fake";
</wxs>
```

上述例子中，所有 WXS 代码均被注释掉了。

> **说明：**
> 方法三和方法二的唯一区别是：没有 `*/` 结束符。

运算符

基本运算符

示例代码：

```
var a = 10, b = 20

// 加法运算
console.log(30 === a + b)
// 减法运算
console.log(-10 === a - b)
// 乘法运算
console.log(200 === a * b)
// 除法运算
console.log(0.5 === a / b)
// 取余运算
console.log(10 === a % b)
```

此外：加法运算（+）也可以用作字符串的拼接。

```
var a = '.q', b = 's'

// 字符串拼接
console.log('.wxs' === a + b)
```

一元运算符

示例代码：

```
var a = 10, b = 20

// 自增运算
console.log(10 === a++)
console.log(12 === ++a)
// 自减运算
console.log(12 === a--)
console.log(10 === --a)
// 正值运算
console.log(10 === +a)
// 负值运算
console.log(0 - 10 === -a)
// 否运算
console.log(-11 === ~a)
// 取反运算
console.log(false === !a)
// delete 运算
console.log(true === delete a.fake)
// void 运算
console.log(undefined === void a)
// typeof 运算
console.log('number' === typeof a)
```

位运算符

示例代码：

```
var a = 10, b = 20

// 左移运算
console.log(80 === a << 3)
// 无符号右移运算
console.log(2 === a >> 2)
// 带符号右移运算
console.log(2 === a >>> 2)
// 与运算
console.log(2 === (a & 3))
// 异或运算
console.log(9 === (a ^ 3))
// 或运算
```

```
console.log(11 === (a | 3))
```

比较运算符

示例代码：

```
var a = 10, b = 20

// 小于
console.log(true === a < b)
// 大于
console.log(false === a > b)
// 小于等于
console.log(true === a <= b)
// 大于等于
console.log(false === a >= b)
```

等值运算符

示例代码：

```
var a = 10, b = 20

// 等号
console.log(false === (a == b))
// 非等号
console.log(true === (a != b))
// 全等号
console.log(false === (a === b))
// 非全等号
console.log(true === (a !== b))
```

赋值运算符

示例代码：

```
var a = 10

a = 10
a *= 10
console.log(100 === a)
a = 10
a /= 5
```

```
console.log(2 === a)
a = 10
a %= 7
console.log(3 === a)
a = 10
a += 5
console.log(15 === a)
a = 10
a -= 11
console.log(-1 === a)
a = 10
a <<= 10
console.log(10240 === a)
a = 10
a >>= 2
console.log(2 === a)
a = 10
a >>>= 2
console.log(2 === a)
a = 10
a &= 3
console.log(2 === a)
a = 10
a ^= 3
console.log(9 === a)
a = 10
a |= 3
console.log(11 === a)
```

二元逻辑运算符

示例代码：

```
var a = 10, b = 20

// 逻辑与
console.log(20 === (a && b))
// 逻辑或
console.log(10 === (a || b))
```

其他运算符

示例代码：

```
var a = 10, b = 20

//条件运算符
console.log(20 === (a >= 10 ? a + 10 : b + 10))
//逗号运算符
console.log(20 === (a, b))
```

运算符优先级

优先级	运算符	说明	结合性
20	(...)	括号	n/a
19	... .	成员访问	从左到右
	... [...]	成员访问	从左到右
	... (...)	函数调用	从左到右
17	... ++	后置递增	n/a
	... --	后置递减	n/a
16	! ...	逻辑非	从右到左
	~ ...	按位非	从右到左
	+ ...	一元加法	从右到左
	- ...	一元减法	从右到左
	++ ...	前置递增	从右到左
	-- ...	前置递减	从右到左
	typeof ...	typeof	从右到左
	void ...	void	从右到左
	delete ...	delete	从右到左
14	... * ...	乘法	从左到右
	... / ...	除法	从左到右

	... % ...	取模	从左到右
13	... + ...	加法	从左到右
	... - ...	减法	从左到右
12	... << ...	按位左移	从左到右
	... >> ...	按位右移	从左到右
	... >>> ...	无符号右移	从左到右
11	... < ...	小于	从左到右
	... <= ...	小于等于	从左到右
	... > ...	大于	从左到右
10	... == ...	大于等于	从左到右
	... != ...	等号	从左到右
	... === ...	非等号	从左到右
	... !== ...	全等号	从左到右
9	... & ...	按位与	从左到右
8	... ^ ...	按位异或	从左到右
7	... | ...	按位或	从左到右
6	... && ...	逻辑或	从左到右
5	... || ...	逻辑与	从左到右
4	... ? ... : ...	条件运算符	从右到左
3	... = ...	赋值	从右到左
	... += ...	赋值	从右到左
	... -= ...	赋值	从右到左
	... *= ...	赋值	从右到左

	... /= ...	赋值	从右到左
	... %= ...	赋值	从右到左
	... <<= ...	赋值	从右到左
	... >>= ...	赋值	从右到左
	... >>>= ...	赋值	从右到左
	... &= ...	赋值	从右到左
	... ^= ...	赋值	从右到左
	... |= ...	赋值	从右到左
0	... , ...	逗号	从左到右

语句

if 语句

在 WXS 中，可以使用以下格式的 if 语句：

- `if (expression) statement` **：当** `expression` **为 truthy 时，执行** `statement` **。**
- `if (expression) statement1 else statement2:` **当** `expression` **为 truthy 时，执行** `statement1` **。否则，执行** `statement2` **。**
- `if ... else if ... else statementN` **通过该句型，可以在** `statement1 ~ statementN` **之间选其中一个执行。**

示例语法：

```
// if ...

if (表达式) 语句

if (表达式) 语句

if (表达式) {
  代码块
}

// if ... else

if (表达式) 语句
else 语句
```

```
if (表达式) 语句
else 语句

if (表达式) {
  代码块
} else {
  代码块
}

// if ... else if ... else ...

if (表达式) {
  代码块
} else if (表达式) {
  代码块
} else if (表达式) {
  代码块
} else {
  代码块
}
```

switch 语句

示例语法：

```
switch (表达式) {
  case 变量:
    语句
  case 数字:
    语句
    break
  case 字符串:
    语句
  default:
    语句
}
```

示例代码：

```
var exp = 10

switch (exp) {
  case '10':
```

```
    console.log('string 10')
    break
  case 10:
    console.log('number 10')
    break
  case exp:
    console.log('var exp')
    break
  default:
    console.log('default')
}
```

输出：

```
number 10
```

for 语句

示例语法：

```
for (语句; 语句; 语句) 语句

for (语句; 语句; 语句) {
  代码块
}
```

支持使用 `break` ， `continue` 关键词。
示例代码：

```
for (语句; 语句; 语句) 语句

for (语句; 语句; 语句) {
  代码块
}
```

输出：

```
0
1
```

while 语句

示例语法：

```
while (表达式) 语句

while (表达式) {
  代码块
}

do {
  代码块
} while (表达式)
```

数据类型

WXS 语言目前共有以下几种数据类型：

类型	说明
number	数值
string	字符串
boolean	布尔值
object	对象
function	函数
array	数组
date	日期
regexp	正则

number

- **语法**

  number 包括两种数值：整数，小数。

  ```
  var a = 10
  var PI = 3.141592653589793
  ```

- **属性**

  `constructor`：返回字符串 "Number"。

- **方法**
  - toString
  - toLocaleString
  - valueOf
  - toFixed
  - toExponential
  - toPrecision

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

string

- **语法**

  string 有两种写法：

```
'hello world'
```

- **属性**
  - constructor：返回字符串 "String"。
  - length

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

- **方法**
  - toString
  - valueOf
  - charAt
  - charCodeAt
  - concat
  - indexOf
  - lastIndexOf
  - localeCompare
  - match
  - replace

- search
- slice
- split
- substring
- toLowerCase
- toLocaleLowerCase
- toUpperCase
- toLocaleUpperCase
- trim

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

boolean

- **语法**

  布尔值只有两个特定的值：`true` 和 `false` 。
- **属性**

  `constructor`：返回字符串 `"Boolean"` 。
- **方法**
  - toString
  - valueOf

> **说明：**
> 以上方法的具体使用请参考 ES5 标准。

object

- **语法**

  object 是一种无序的键值对。使用方法如下所示：

```
var o = {} //生成一个新的空对象

//生成一个新的非空对象
o = {
  string: 1, //object 的 key 可以是字符串
  const_var: 2, //object 的 key 也可以是符合变量定义规则的标识符
  func: {}, //object 的 value 可以是任何类型
}
```

```
//对象属性的读操作
console.log(1 === o['string'])
console.log(2 === o.const_var)

//对象属性的写操作
o['string']++
o['string'] += 10
o.const_var++
o.const_var += 10

//对象属性的读操作
console.log(12 === o['string'])
console.log(13 === o.const_var)
```

- **属性**

  `constructor`：返回字符串 `"Object"`

```
console.log('Object' === { k: '1', v: '2' }.constructor)
```

- **方法**

  `toString`：返回字符串 `"[object Object]"`

function

语法

function 支持以下的定义方式：

```
//方法 1
function a(x) {
  return x
}

//方法 2
var b = function(x) {
  return x
}
```

function 同时也支持以下的语法（匿名函数，闭包等）：

```
var a = function(x) {
  return function() {
```

```
    return x
  }
}

var b = a(100)
console.log(100 === b())
```

arguments

function 里面可以使用 `arguments` 关键词。该关键词目前只支持以下的属性：

- `length` ：传递给函数的参数个数。
- `[index]`：通过 `index` 下标可以遍历传递给函数的每个参数。

示例代码：

```
var a = function() {
  console.log(3 === arguments.length)
  console.log(100 === arguments[0])
  console.log(200 === arguments[1])
  console.log(300 === arguments[2])
}
a(100, 200, 300)
```

- **属性**
  - `constructor`：返回字符串 `"Function"`
  - `length`：返回函数的形参个数
- **方法**

  `toString`：返回字符串 `"[function Function]"`

  示例代码：

```
var func = function(a, b, c) {}

console.log('Function' === func.constructor)
console.log(3 === func.length)
console.log('[function Function]' === func.toString())
```

array

- **语法**

  array 支持以下的定义方式：

```
var a = [] //生成一个新的空数组

a = [1, '2', {}, function() {}] //生成一个新的非空数组，数组元素可以是任何类型
```

- **属性**
  - `constructor`：返回字符串 `"Array"`
  - `length`
- **方法**
  - toString
  - concat
  - join
  - pop
  - push
  - reverse
  - shift
  - slice
  - sort
  - splice
  - unshift
  - indexOf
  - lastIndexOf
  - every
  - some
  - forEach
  - map
  - filter
  - reduce
  - reduceRight

> **说明：**
> 以上方法的具体使用请参考 ES5 标准。

date

- **语法**

- 生成 date 对象需要使用 getDate函数, 返回一个当前时间的对象。

```
getDate()
getDate(milliseconds)
getDate(datestring)
getDate(year, month[, date[, hours[, minutes[, seconds[, milliseconds]]]]])
```

- **参数**
  - `milliseconds`：从 1970 年 1 月 1 日 00:00:00 UTC 开始计算的毫秒数。
  - `datestring`：日期字符串，其格式为："month day, year hours:minutes:seconds"。

- **属性**

  `constructor`：返回字符串 “Date”。

- **方法**
  - toString
  - toDateString
  - toTimeString
  - toLocaleString
  - toLocaleDateString
  - toLocaleTimeString
  - valueOf
  - getTime
  - getFullYear
  - getUTCFullYear
  - getMonth
  - getUTCMonth
  - getDate
  - getUTCDate
  - getDay
  - getUTCDay
  - getHours
  - getUTCHours
  - getMinutes
  - getUTCMinutes
  - getSeconds
  - getUTCSeconds

- getMilliseconds
- getUTCMilliseconds
- getTimezoneOffset
- setTime
- setMilliseconds
- setUTCMilliseconds
- setSeconds
- setUTCSeconds
- setMinutes
- setUTCMinutes
- setHours
- setUTCHours
- setDate
- setUTCDate
- setMonth
- setUTCMonth
- setFullYear
- setUTCFullYear
- toUTCString
- toISOString
- toJSON

regexp

- **语法**

  生成 `regexp` 对象需要使用 getRegExp 函数。

  ```
  getRegExp(pattern[, flags])
  ```

- **参数：**
  - pattern: 正则表达式的内容
  - flags：修饰符
  - flagg: global
  - flagi: ignoreCase
  - flagm: multiline

  示例代码：

```
var a = getRegExp('x', 'img')
console.log('x' === a.source)
console.log(true === a.global)
console.log(true === a.ignoreCase)
console.log(true === a.multiline)
```

- **属性**
  - constructor：返回字符串 "RegExp"
  - source
  - global
  - ignoreCase
  - multiline
  - lastIndex

> **说明：**
> 除 constructor 外属性的具体含义请参考 ES5 标准。

- **方法**
  - exec
  - test
  - toString

> **说明：**
> 以上方法的具体使用请参考 ES5 标准。

数据类型判断

construct 属性

数据类型的判断可以使用 `constructor` 属性
示例代码：

```
var number = 10
console.log('Number' === number.constructor)

var string = 'str'
console.log('String' === string.constructor)

var boolean = true
```

```
console.log('Boolean' === boolean.constructor)

var object = {}
console.log('Object' === object.constructor)

var func = function() {}
console.log('Function' === func.constructor)

var array = []
console.log('Array' === array.constructor)

var date = getDate()
console.log('Date' === date.constructor)

var regexp = getRegExp()
console.log('RegExp' === regexp.constructor)
```

typeof

使用 `typeof` 也可以区分部分数据类型。
示例代码：

```
var number = 10
var boolean = true
var object = {}
var func = function() {}
var array = []
var date = getDate()
var regexp = getRegExp()

console.log('number' === typeof number)
console.log('boolean' === typeof boolean)
console.log('object' === typeof object)
console.log('function' === typeof func)
console.log('object' === typeof array)
console.log('object' === typeof date)
console.log('object' === typeof regexp)

console.log('undefined' === typeof undefined)
console.log('object' === typeof null)
```

基础类库

console

console.log 方法用于在 console 窗口输出信息。它可以接受多个参数，将它们的结果连接起来输出。

Math

- **属性**
  - E
  - LN10
  - LN2
  - LOG2E
  - LOG10E
  - PI
  - SQRT1_2
  - SQRT2

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

- **方法**
  - abs
  - acos
  - asin
  - atan
  - atan2
  - ceil
  - cos
  - exp
  - floor
  - log
  - max
  - min
  - pow
  - random
  - round
  - sin
  - sqrt
  - tan

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

JSON

- **方法**
  - `stringify(object)`：将 `object` 对象转换为 `JSON` 字符串，并返回该字符串。
  - `parse(string)`：将 `JSON` 字符串转化成对象，并返回该对象。
- **示例代码：**

```
console.log(undefined === JSON.stringify())
console.log(undefined === JSON.stringify(undefined))
console.log('null' === JSON.stringify(null))

console.log('111' === JSON.stringify(111))
console.log('"111"' === JSON.stringify('111'))
console.log('true' === JSON.stringify(true))
console.log(undefined === JSON.stringify(function() {}))

console.log(undefined === JSON.parse(JSON.stringify()))
console.log(undefined === JSON.parse(JSON.stringify(undefined)))
console.log(null === JSON.parse(JSON.stringify(null)))

console.log(111 === JSON.parse(JSON.stringify(111)))
console.log('111' === JSON.parse(JSON.stringify('111')))
console.log(true === JSON.parse(JSON.stringify(true)))

console.log(undefined === JSON.parse(JSON.stringify(function() {})))
```

Number

- **属性**
  - MAX_VALUE
  - MIN_VALUE
  - NEGATIVE_INFINITY
  - POSITIVE_INFINITY

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

Date

- **属性**
  - parse
  - UTC
  - now

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

Global

- **属性**
  - NaN
  - Infinity
  - undefined

> **说明：**
> 以上属性的具体使用请参考 ES5 标准。

- **方法**
  - parseInt
  - parseFloat
  - isNaN
  - isFinite
  - decodeURI
  - decodeURIComponent
  - encodeURI
  - encodeURIComponent

视图层

视图层说明

最近更新时间：2024-03-28 15:39:01

概述

小程序框架的视图层由 WXML 和 WXSS 编写，由组件来进行展示。WXML 用于描述页面的结构，而 WXSS 用于描述页面的样式。WXS 是小程序的一套脚本语言，结合 WXML，可以构建出页面的结构。
框架的逻辑层负责将数据转化为视图，同时将视图层的事件发送给逻辑层。这样，开发者可以专注于数据和逻辑的处理，而不必过多关注视图层的实现细节。
组件是视图的基本组成单元，可以帮助开发者更轻松地实现各种功能和交互效果。开发者可以自定义组件，也可以使用框架提供的组件库，从而快速地构建出具有原生 App 体验的服务。

WXML

WXML 相关操作，可参见 WXML 详情 。

WXSS

WXSS 相关操作，可参见 WXSS 详情 。

基础组件

说明

框架为开发者提供了一系列基础组件，开发者可以通过组合这些基础组件进行快速开发。
什么是组件：

- 组件是视图层的基本组成单元。
- 组件自带一些功能与 TMF 风格一致的样式。
- 一个组件通常包括 `开始标签` 和 `结束标签` ， `属性` 用来修饰这个组件， `内容` 在两个标签之内。

```
<tagname property="value">
  Content goes here ...
</tagname>
```

> **说明：**
> 所有组件与属性都是小写，以连字符-连接。

属性类型

类型	描述	注解
Boolean	布尔值	组件写上该属性，不管是什么值都被当作 True；只有组件上没有该属性时，属性值才为 False。如果属性值为变量，变量的值会被转换为 Boolean 类型
Number	数字	1, 2.5
String	字符串	"string"
Array	数组	[1, "string"]
Object	对象	{ key: value }
EventHandler	事件处理函数名	"handlerName" 是 Page 中定义的事件处理函数名
Any	任意属性	-

公共属性

所有组件都有以下属性：

属性名	类型	描述	注解
id	string	组件的唯一标识	保持整个页面唯一
class	string	组件的唯一标识	在对应的 WXSS 中定义的样式类
style	string	组件的内联样式	可以动态设置的内联样式
hidden	boolean	组件是否显示	所有组件默认显示
date-*	any	自定义属性	组件上触发的事件时，会发送给事件处理函数
bind/catch	eventhandler	组件的事件	详见 WXML

特殊属性

几乎所有组件都有各自定义的属性，可以对该组件的功能或样式进行修饰，请参考各个 组件概览 的定义。

获取界面上的节点信息

WXML 节点信息

节点信息查询 API 可以用于获取节点属性、样式、在界面上的位置等信息。
最常见的用法是使用这个接口来查询某个节点的当前位置，以及界面的滚动位置。
示例代码：

```
const query = wx.createSelectorQuery()
query.select('#the-id').boundingClientRect(function (res) {
  res.top // #the-id 节点的上边界坐标（相对于显示区域）
})
query.selectViewport().scrollOffset(function (res) {
  res.scrollTop // 显示区域的竖直滚动位置
})
query.exec()
```

上述示例中，`#the-id` 是一个节点选择器，与 CSS 的选择器相近但略有区别，请参见 SelectorQuery 的相关说明。
在自定义组件或包含自定义组件的页面中，推荐使用 `this.createSelectorQuery` 来代替 wx.createSelectorQuery，这样可以确保在正确的范围内选择节点。

WXML 节点布局相交状态

节点布局相交状态 API 可用于监听两个或多个组件节点在布局位置上的相交状态。这一组 API 常常可以用于，推断某些节点是否可以被用户看见、有多大比例可以被用户看见。
这一组 API 涉及的主要概念如下。

- 参照节点：监听的参照节点，取它的布局区域作为参照区域。如果有多个参照节点，则会取它们布局区域的交集作为参照区域。页面显示区域也可作为参照区域之一。
- 目标节点：监听的目标，默认只能是一个节点（使用 `selectAll` 选项时，可以同时监听多个节点）。
- 相交区域：目标节点的布局区域与参照区域的相交区域。
- 相交比例：相交区域占参照区域的比例。
- 阈值：相交比例如果达到阈值，则会触发监听器的回调函数。阈值可以有多个。

以下示例代码可以在目标节点（用选择器 `.target-class` 指定）每次进入或离开页面显示区域时，触发回调函数。
示例代码：

```
Page({
  onLoad() {
    wx.createIntersectionObserver().relativeToViewport().observe('.target-class', (res)
=> {
      res.id // 目标节点 id
      res.dataset // 目标节点 dataset
      res.intersectionRatio // 相交区域占目标节点的布局区域的比例
      res.intersectionRect // 相交区域
      res.intersectionRect.left // 相交区域的左边界坐标
```

```
    res.intersectionRect.top // 相交区域的上边界坐标
    res.intersectionRect.width // 相交区域的宽度
    res.intersectionRect.height // 相交区域的高度
   })
  }
})
```

以下示例代码可以在目标节点（用选择器 .target-class 指定）与参照节点（用选择器 .relative-class 指定）在页面显示区域内相交或相离，且相交或相离程度达到目标节点布局区域的20%和50%时，触发回调函数。

示例代码：

```
Page({
  onLoad() {
    wx.createIntersectionObserver(this, {
      thresholds: [0.2, 0.5]
    }).relativeTo('.relative-class').relativeToViewport().observe('.target-class', (res) => {
      res.intersectionRatio // 相交区域占目标节点的布局区域的比例
      res.intersectionRect // 相交区域
      res.intersectionRect.left // 相交区域的左边界坐标
      res.intersectionRect.top // 相交区域的上边界坐标
      res.intersectionRect.width // 相交区域的宽度
      res.intersectionRect.height // 相交区域的高度
    })
  }
})
```

说明：

- 与页面显示区域的相交区域并不准确代表用户可见的区域，因为参与计算的区域是“布局区域”，布局区域可能会在绘制时被其他节点裁剪隐藏（如遇祖先节点中 overflow 样式为 hidden 的节点）或遮盖（如遇 fixed 定位的节点）。
- 在自定义组件或包含自定义组件的页面中，推荐使用 `this.createIntersectionObserver` 来代替 wx.createIntersectionObserver，这样可以确保在正确的范围内选择节点。

响应显示区域变化

显示区域尺寸

显示区域指小程序界面中可以自由布局展示的区域。在默认情况下，小程序显示区域的尺寸自页面初始化起就不会发生变化。但以下两种方式都可以改变这一默认行为。

在手机上启用屏幕旋转支持

小程序在手机上支持屏幕旋转。使小程序中的页面支持屏幕旋转的方法是：在 app.json 的 window 段中设置 "pageOrientation": "auto" ，或在页面 json 文件中配置 "pageOrientation": "auto" 。
以下是在单个页面 json 文件中启用屏幕旋转的示例。
代码示例：

```
{
  "pageOrientation": "auto"
}
```

在iPad上启用屏幕选择支持

在 iPad 上运行的小程序可以支持屏幕旋转。使小程序支持 iPad 屏幕旋转的方法是：在 app.json 中添加 "resizable": true 。
代码示例：

```
{
  "resizable": true
}
```

Media Query

有时，对于不同尺寸的显示区域，页面的布局会有所差异。此时可以使用 media query 来解决大多数问题。
代码示例：

```
.my-class {
  width: 40px;
}

@media (min-width: 480px) {
  /* 仅在 480px 或更宽的屏幕上生效的样式规则 */
  .my-class {
    width: 200px;
  }
}
```

屏幕旋转事件

有时，仅仅使用 media query 无法控制一些精细的布局变化。此时可以使用 js 作为辅助。
在 js 中读取页面的显示区域尺寸，可以使用 selectorQuery.selectViewport 。
页面尺寸发生改变的事件，可以使用页面的 onResize 来监听。对于自定义组件，可以使用 resize 生命周期来监听。回调函数中将返回显示区域的尺寸信息。

代码示例：

```
Page({
  onResize(res) {
    res.size.windowWidth // 新的显示区域宽度
    res.size.windowHeight // 新的显示区域高度
  }
})
```

```
Component({
  pageLifetimes: {
    resize(res) {
      res.size.windowWidth // 新的显示区域宽度
      res.size.windowHeight // 新的显示区域高度
    }
  }
})
```

此外，还可以使用 `wx.onWindowResize` 来监听（但这不是推荐的方式）。

动画

界面动画的常见方式

在小程序中，通常可以使用 CSS 渐变 和 CSS 动画 来创建简易的界面动画。
同时，还可以使用 wx.canvasContext 接口来动态创建简易的动画效果。
动画过程中，可以使用 `bindtransitionend` `bindanimationstart` `bindanimationiteration` `bindanimationend` 来监听动画事件。

- transitionend：CSS 渐变结束或 wx.createAnimation 结束一个阶段。
- animationstart：CSS 动画开始。
- animationiteration：CSS 动画结束一个阶段。
- animationend：CSS 动画结束。

> **说明：**
> 这几个事件都不是冒泡事件，需要绑定在真正发生了动画的节点上才会生效。

高级的动画方式

在一些复杂场景下，上述的动画方法可能并不适用。
WXML 相应事件 的方式可以通过使用 QS 来响应事件的方法来动态调整节点的 style 属性。通过不断改变 style 属性的值可以做到动画效果。同时，这种方式也可以根据用户的触摸事件来动态地生成动画。

使用连续使用 setData 来改变界面的方法也可以达到动画的效果。这样可以任意地改变界面，但通常会产生较大的延迟或卡顿，甚至导致小程序僵死。此时可以通过将页面的 setData 改为 自定义模块组件 中的 setData 来提升性能。

WXML

最近更新时间：2024-03-21 15:51:52

WXML（WeiXin Markup Language）是框架设计的一套标签语言，结合基础组件、事件系统，可以构建出页面的结构。

> **说明：**
> WXML 的数据绑定，不能使用 ECMAScript 规范已经规定的关键字和保留字，使用了会提示 `Unexpected token` 或者 `Unexpected token: "tokenName", may be reserved words.` 的报错。

能力概要

用以下一些简单的例子来介绍 WXML 有什么能力，具体的能力以及使用方式在以下章节查看：

- 数据绑定

```
<!--WXML-->
<view>{{message}}</view>
```

```
// page.js
Page({
  data: {
    message: 'Hello MINA!',
  },
})
```

- 列表渲染

```
<!--WXML-->
<view wx:for="{{array}}">{{item}}</view>
```

```
// page.js
Page({
  data: {
    array: [1, 2, 3, 4, 5],
  },
})
```

- 条件渲染

```
<!--WXML-->
<view wx:if="{{view == 'WEBVIEW'}}">WEBVIEW</view>
<view wx:elif="{{view == 'APP'}}">APP</view>
<view wx:else="{{view == 'MINA'}}">MINA</view>
```

```
// page.js
Page({
  data: {
    view: 'MINA',
  },
})
```

- **模板**

```
<!--WXML-->
<template name="staffName">
  <view>
    FirstName: {{firstName}}, LastName: {{lastName}}
  </view>
</template>

<template is="staffName" data="{{...staffA}}"></template>
<template is="staffName" data="{{...staffB}}"></template>
<template is="staffName" data="{{...staffC}}"></template>
```

```
// page.js
Page({
  data: {
    staffA: { firstName: 'Hulk', lastName: 'Hu' },
    staffB: { firstName: 'Shang', lastName: 'You' },
    staffC: { firstName: 'Gideon', lastName: 'Lin' },
  },
})
```

- **事件**

```
<view bindtap="add">{{count}}</view>
```

```
Page({
  data: {
```

```
    count: 1,
  },
  add(e) {
    this.setData({
      count: this.data.count + 1,
    })
  },
})
```

数据绑定

WXML 中的动态数据均来自对应 Page 的 data。

简单绑定

需要使用 Mustache 语法（双大括号）将变量包起来才能实现引用，可以作用于：

内容

```
<view>{{ message }}</view>
```

```
Page({
  data: {
    message: 'Hello MINA!',
  },
})
```

组件属性（需要在双引号之内）

```
<view id="item-{{id}}"></view>
```

```
Page({
  data: {
    id: 0,
  },
})
```

控制属性（需要在双引号之内）

```
<view wx:if="{{condition}}"></view>
```

```
Page({
  data: {
    condition: true,
  },
})
```

关键字（需要在双引号之内）

`true`：boolean 类型的 true，代表真值。
`false`： boolean 类型的 false，代表假值。

> ⚠ **注意：**
> 不要直接写 `checked="false"`，其计算结果是一个字符串，转成 boolean 类型后代表真值。

运算

可以在 {{}} 内进行简单的运算，支持的有如下几种方式：

三元运算

```
<view hidden="{{flag ? true : false}}">Hidden</view>
```

算数运算

```
<view>{{a + b}} + {{c}} + d</view>
```

```
Page({
  data: {
    a: 1,
    b: 2,
    c: 3,
  },
})
```

view 中的内容为 3 + 3 + d。

逻辑判断

```
<view wx:if="{{length > 5}}"></view>
```

字符串运算

```
<view>{{"hello" + name}}</view>
```

```
Page({
  data: {
    name: 'MINA',
  },
})
```

数据路径运算

```
<view>{{object.key}} {{array[0]}}</view>
```

```
Page({
  data: {
    object: {
      key: 'Hello ',
    },
    array: ['MINA'],
  },
})
```

组合

也可以在 Mustache 内直接进行组合，构成新的对象或者数组。

数组

```
<view wx:for="{{[zero, 1, 2, 3, 4]}}">{{item}}</view>
```

```
Page({
  data: {
    zero: 0,
  },
})
```

最终组合成数组[0, 1, 2, 3, 4]。

对象

```
<template is="objectCombine" data="{{for: a, bar: b}}"></template>
```

```
Page({
  data: {
    a: 1,
    b: 2,
  },
})
```

最终组合成的对象是 `{for: 1, bar: 2}`
也可以用扩展运算符 `...` 来将一个对象展开。

```
<template is="objectCombine" data="{{...obj1, ...obj2, e: 5}}"></template>
```

```
Page({
  data: {
    obj1: {
      a: 1,
      b: 2,
    },
    obj2: {
      c: 3,
      d: 4,
    },
  },
})
```

最终组合成的对象是 `{a: 1, b: 2, c: 3, d: 4, e: 5}` 。
如果对象的 key 和 value 相同，也可以间接地表达。

```
<template is="objectCombine" data="{{foo, bar}}"></template>
```

```
Page({
  data: {
    foo: 'my-foo',
    bar: 'my-bar',
  },
```

```
})
```

最终组合成的对象是 `{foo: 'my-foo', bar:'my-bar'}` 。
上述方式可以随意组合，但是如有存在变量名相同的情况，后边的会覆盖前面，如：

```
<template is="objectCombine" data="{{...obj1, ...obj2, a, c: 6}}"></template>
```

```
Page({
 data: {
  obj1: {
   a: 1,
   b: 2,
  },
  obj2: {
   b: 3,
   c: 4,
  },
  a: 5,
 },
})
```

最终组合成的对象是 `{a: 5, b: 3, c: 6}` 。
花括号和引号之间如果有空格，将最终被解析成为字符串。

```
<view wx:for="{{[1,2,3]}} ">
 {{item}}
</view>
```

等同于

```
<view wx:for="{{[1,2,3] + ' '}}">
 {{item}}
</view>
```

列表渲染

wx：for

在组件上使用 `wx:for` 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该组件。
默认数组的当前项的下标变量名默认为 `index` ，数组当前项的变量名默认为 `item` 。

```
```

```
<view wx:for="{{array}}">
  {{index}}: {{item.message}}
</view>
```

```
Page({
  data: {
    array: [
      {
        message: 'foo',
      },
      {
        message: 'bar',
      },
    ],
  },
})
```

使用 `wx:for-item` 可以指定数组当前元素的变量名，
使用 `wx:for-index` 可以指定数组当前下标的变量名：

```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
  {{idx}}: {{itemName.message}}
</view>
```

`wx:for` 也可以嵌套，下边是一个九九乘法表：

```
<view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="i">
  <view wx:for="{{[1, 2, 3, 4, 5, 6, 7, 8, 9]}}" wx:for-item="j">
    <view wx:if="{{i <= j}}">
      {{i}} * {{j}} = {{i * j}}
    </view>
  </view>
</view>block wx：for
```

类似 `block wx:if`，也可以将 `wx:for` 用在 `<block/>` 标签上，以渲染一个包含多节点的结构块。例如：

```
<block wx:for="{{[1, 2, 3]}}">
  <view>{{index}}:</view>
  <view>{{item}}</view>
</block>
```

wx：key

如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（如 <input> 中的输入内容，<switch> 的选中状态），需要使用 wx:key 来指定列表中项目的唯一的标识符。

wx:key 的值以两种形式提供：

- 字符串，代表在 for 循环的 array 中 item 的某个 property，该 property 的值需要是列表中唯一的字符串或数字，且不能动态改变。
- 保留关键字 *this 代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数字，如：

当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率。

如不提供 wx:key，会报一个 warning，如果明确知道该列表是静态，或者不必关注其顺序，可以选择忽略。

示例代码：

```
<switch wx:for="{{objectArray}}" wx:key="unique" style="display: block;">
 {{item.id}}
</switch>
<button bindtap="switch">Switch</button>
<button bindtap="addToFront">Add to the front</button>

<switch wx:for="{{numberArray}}" wx:key="*this" style="display: block;">
 {{item}}
</switch>
<button bindtap="addNumberToFront">Add to the front</button>
```

```
Page({
 data: {
  objectArray: [
   { id: 5, unique: 'unique_5' },
   { id: 4, unique: 'unique_4' },
   { id: 3, unique: 'unique_3' },
   { id: 2, unique: 'unique_2' },
   { id: 1, unique: 'unique_1' },
   { id: 0, unique: 'unique_0' },
  ],
  numberArray: [1, 2, 3, 4],
 },
 switch(e) {
  const length = this.data.objectArray.length
  for (let i = 0; i < length; ++i) {
```

```
      const x = Math.floor(Math.random() * length)
      const y = Math.floor(Math.random() * length)
      const temp = this.data.objectArray[x]
      this.data.objectArray[x] = this.data.objectArray[y]
      this.data.objectArray[y] = temp
    }
    this.setData({
      objectArray: this.data.objectArray,
    })
  },
  addToFront(e) {
    const length = this.data.objectArray.length
    this.data.objectArray = [{ id: length, unique: 'unique_' + length }].concat(
      this.data.objectArray
    )
    this.setData({
      objectArray: this.data.objectArray,
    })
  },
  addNumberToFront(e) {
    this.data.numberArray = [this.data.numberArray.length + 1].concat(
      this.data.numberArray
    )
    this.setData({
      numberArray: this.data.numberArray,
    })
  },
})
```

花括号和引号之间如果有空格，将最终被解析成为字符串。

```
<view wx:for="{{[1,2,3]}} ">
  {{item}}
</view>
```

等同于：

```
<view wx:for="{{[1,2,3] + ' '}}">
  {{item}}
</view>
```

条件渲染

wx：if

在框架中，使用 `wx:if="{{condition}}"` 来判断是否需要渲染该代码块：

```
<view wx:if="{{condition}}">True</view>
```

也可以用 `wx:elif` 和 `wx:else` 来添加一个 else 块：

```
<view wx:if="{{length > 5}}">1</view>
<view wx:elif="{{length > 2}}">2</view>
<view wx:else>3</view>
```

block wx：if

因为 `wx:if` 是一个控制属性，需要将它添加到一个标签上。如果要一次性判断多个组件标签，可以使用一个 `<block/>` 标签将多个组件包装起来，并在上边使用 `wx:if` 控制属性。

```
<block wx:if="{{true}}">
  <view>view1</view>
  <view>view2</view>
</block>
```

`<block/>` 并不是一个组件，它只是一个包装元素，不会在页面中做任何渲染，只接受控制属性。

wx:if vs hidden

因为 wx:if 之中的模板也可能包含数据绑定，所以当 `wx:if` 的条件值切换时，框架有一个局部渲染的过程，因此它会确保条件块在切换时，销毁或重新渲染。

同时 `wx:if` 也是**惰性的**，如果在初始渲染条件为 `false` ，框架什么也不做，在条件第一次变成真的时候才开始局部渲染。

相比之下， `hidden` 就简单的多，组件始终会被渲染，只是简单的控制显示与隐藏。

一般来说， `wx:if` 有更高的切换消耗而 `hidden` 有更高的初始渲染消耗。因此，如果需要频繁切换的情景下，用 `hidden` 更好，如果在运行时条件不大可能改变则 `wx:if` 较好。

模板

WXML 提供模板（template），可以在模板中定义代码片段，然后在不同的地方调用。

定义模板

使用 name 属性，作为模板的名字。然后在 `<template/>` 内定义代码片段，如：

```
<!--
```

```
  index: int
  msg: string
  time: string
-->
<template name="msgItem">
  <view>
    <text>{{index}}: {{msg}}</text>
    <text>Time: {{time}}</text>
  </view>
</template>
```

使用模板

使用 is 属性，声明需要的使用的模板，然后将模板所需要的 data 传入，如：

```
<template is="msgItem" data="{{...item}}" />
```

```
Page({
  data: {
    item: {
      index: 0,
      msg: 'this is a template',
      time: '2016-09-15',
    },
  },
})
```

is 属性可以使用 Mustache 语法，来动态决定具体需要渲染哪个模板：

```
<template name="odd">
  <view>odd</view>
</template>
<template name="even">
  <view>even</view>
</template>

<block wx:for="{{[1, 2, 3, 4, 5]}}">
  <template is="{{item % 2 == 0 ? 'even' : 'odd'}}" />
</block>
```

模板作用域

模板拥有自己的作用域，只能使用 data 传入的数据以及模板定义文件中定义的 `<wxs />` 模块。

事件

什么是事件

- 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。
- 事件可以绑定在组件上，当达到触发事件，就会执行逻辑层中对应的事件处理函数。
- 事件对象可以携带额外信息，如 id，dataset，touches。

事件的使用方式

- 在组件中绑定一个事件处理函数。

  如 `bindtap` ，当用户单击该组件的时候会在该页面对应的 Page 中找到相应的事件处理函数。

```
<view id="tapTest" data-hi="WeChat" bindtap="tapName">Click me!</view>
```

- 在相应的 Page 定义中写上相应的事件处理函数，参数是 event。

```
Page({
 tapName(event) {
  console.log(event)
 },
})
```

- 可以看到 log 出来的信息大致如下：

```
{
 "type": "tap",
 "timeStamp": 895,
 "target": {
  "id": "tapTest",
  "dataset": {
   "hi": "WeChat"
  }
 },
 "currentTarget": {
  "id": "tapTest",
  "dataset": {
   "hi": "WeChat"
  }
 },
```

```
  "detail": {
    "x": 53,
    "y": 14
  },
  "touches": [
    {
      "identifier": 0,
      "pageX": 53,
      "pageY": 14,
      "clientX": 53,
      "clientY": 14
    }
  ],
  "changedTouches": [
    {
      "identifier": 0,
      "pageX": 53,
      "pageY": 14,
      "clientX": 53,
      "clientY": 14
    }
  ]
}
```

使用 WXS 函数响应事件

WXS 函数接受 2 个参数，第一个是 event，在原有的 event 的基础上加了 `event.instance` 对象，第二个参数是 `ownerInstance` ，和 `event.instance` 一样是一个 `ComponentDescriptor` 对象。具体使用如下：

- 在组件中绑定和注册事件处理的 WXS 函数。

```
<wxs module="wxs" src="./test.wxs"></wxs>
<view id="tapTest" data-hi="TMF" bindtap="{{wxs.tapName}}">Click me!
</view>
**注：绑定的WXS函数必须用{{}}括起来**
```

- test.wxs 文件实现 tapName 函数：

```
function tapName(event, ownerInstance) {
  console.log('tap wechat', JSON.stringify(event))
}
module.exports = {
  tapName,
}
```

`ownerInstance` 包含了一些方法，可以设置组件的样式和 class，具体包含的方法以及为什么要用 WXS 函数响应事件，请单击查看详情。

事件详解

事件的分类

事件分为冒泡事件和非冒泡事件：

- 冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递。
- 非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。

WXML 的冒泡事件列表：

类型	触发条件
touchstart	手指触摸动作开始
touchmove	手指触摸后移动
touchcancel	手指触摸动作被打断，如来电提醒，弹窗
touchend	手指触摸动作结束
tap	手指触摸后马上离开
longpress	手指触摸后，超过 350ms 再离开，如果指定了事件回调函数并触发了这个事件，tap 事件将不被触发
longtap	手指触摸后，超过 350ms 再离开（推荐使用 longpress 事件代替）
transitionend	会在 WXSS transition 或 wx.createAnimation 动画结束后触发
animationstart	会在一个 WXSS animation 动画开始时触发
animationiteration	会在一个 WXSS animation 一次迭代结束时触发
animationend	会在一个 WXSS animation 动画完成时触发
touchforcechange	在支持 3D Touch 的 iPhone 设备，重按时会触发

> **说明：**
> 除上表之外的其他组件自定义事件如无特殊声明都是非冒泡事件，如<form/>的 submit 事件，<input/>的 input 事件，<scroll-view/>的 scroll 事件。

事件绑定和冒泡

事件绑定的写法同组件的属性，以 key、value 的形式。

- key 以 bind 或 catch 开头，然后跟上事件的类型，如 `bindtap` 、 `catchtouchstart` 。在非原生组件中，`bind` 和 `catch` 后可以紧跟一个冒号，其含义不变，如 `bind:tap` 、 `catch:touchstart` 。
- value 是一个字符串，需要在对应的 Page 中定义同名的函数。不然当触发事件的时候会报错。

`bind` 事件绑定不会阻止冒泡事件向上冒泡， `catch` 事件绑定可以阻止冒泡事件向上冒泡。
在下面的代码中，单击 inner view 会先后调用 `handleTap3` 和 `handleTap2` (因为 tap 事件会冒泡到 middle view，而 middle view 阻止了 tap 事件冒泡，不再向父节点传递)，单击 middle view 会触发 `handleTap2` ，单击 outer view 会触发 `handleTap1` 。

```
<view id="outer" bindtap="handleTap1">
  outer view
  <view id="middle" catchtap="handleTap2">
    middle view
    <view id="inner" bindtap="handleTap3">
      inner view
    </view>
  </view>
</view>
```

事件的捕获阶段

触摸类事件支持捕获阶段。捕获阶段位于冒泡阶段之前，且在捕获阶段中，事件到达节点的顺序与冒泡阶段恰好相反。需要在捕获阶段监听事件时，可以采用 capture-bind、capture-catch 关键字，后者将中断捕获阶段和取消冒泡阶段。
在下面的代码中，单击 inner view 会先后调用 `handleTap2` 、 `handleTap4` 、 `handleTap3` 、 `handleTap1` 。

```
<view id="outer" bind:touchstart="handleTap1" capture-
bind:touchstart="handleTap2">
  outer view
  <view id="inner" bind:touchstart="handleTap3" capture-
bind:touchstart="handleTap4">
    inner view
  </view>
</view>
```

如果将上述代码中的第一个 `capture-bind` 改为 `capture-catch` ，则只会触发 `handleTap2` 。

```
<view id="outer" bind:touchstart="handleTap1" capture-
catch:touchstart="handleTap2">
  outer view
```

```
  <view id="inner" bind:touchstart="handleTap3" capture-
bind:touchstart="handleTap4">
    inner view
  </view>
</view>
```

事件对象

如无特殊说明，当组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。

- BaseEvent基础事件对象属性列表：

属性	类型	说明
type	string	事件类型
timeStamp	interger	事件生成时的时间戳
target	object	触发事件的组件的一些属性值集合
currentTarget	object	当前组件的一些属性值集合

- CustomEvent 自定义事件对象属性列表（继承 BaseEvent）：

属性	类型	说明
detail	object	额外的信息

- TouchEvent 触摸事件对象属性列表（继承 BaseEvent）：

属性	类型	说明
touches	array	触摸事件，当前停留在屏幕中的触摸点信息的数组
changedTouches	array	触摸事件，当前变化的触摸点信息的数组

特殊事件： `<canvas>` **中的触摸事件不可冒泡，所以没有 currentTarget。**

- type

  代表事件的类型。
- timeStamp

  页面打开到触发事件所经过的毫秒数。
- target

属性	类型	说明

id	string	事件源组件的 id
tagName	string	当前组件的类型
dataset	object	事件源组件上由 data- 开头的自定义属性组成的集合

- currenTarget

属性	类型	说明
id	string	当前组件的 id
tagName	string	当前组件的类型
dataset	object	当前组件上由 data- 开头的自定义属性组成的集合

- dataset

在组件中可以定义数据，这些数据将会通过事件传递给 SERVICE。书写方式：以 `data-` 开头，多个单词由连字符 `-` 链接，不支持大写(大写会自动转成小写)如 `data-element-type`，最终在 `event.currentTarget.dataset` 中会将连字符转成驼峰 `elementType`。

代码示例：

```
<view data-alpha-beta="1" data-alphaBeta="2" bindtap="bindViewTap">
 DataSet Test
</view>
```

```
Page({
 bindViewTap(event) {
  event.currentTarget.dataset.alphaBeta === 1 // - 会转为驼峰写法
  event.currentTarget.dataset.alphabeta === 2 // 大写会转为小写
 },
})
```

- touches

touches 是一个数组，每个元素为一个 Touch 对象（canvas 触摸事件中携带的 touches 是 CanvasTouch 数组）。表示当前停留在屏幕上的触摸点。

Touch 对象

属性	类型	说明

identifier	number	触摸点的标识符
pageX，pageY	number	距离文档左上角的距离，文档的左上角为原点，横向为 X 轴，纵向为 Y 轴
clientX，clientY	number	距离页面可显示区域（屏幕除去导航条）左上角距离，横向为 X 轴，纵向为 Y 轴

CanvasTouch 对象

属性	类型	说明
identifier	number	触摸点的标识符
x, y	number	距离 Canvas 左上角的距离，Canvas 的左上角为原点 ，横向为 X 轴，纵向为 Y 轴

- changedTouches

  changedTouches 数据格式同 touches。 表示有变化的触摸点，如从无变有（touchstart），位置变化（touchmove），从有变无（touchend、touchcancel）。
- detail

自定义事件所携带的数据，如表单组件的提交事件会携带用户的输入，媒体的错误事件会携带错误信息。

单击事件的 `detail` 带有的 x，y 同 pageX，pageY 代表距离文档左上角的距离。

WXS响应事件

背景

有频繁用户交互的效果在小程序上表现是比较卡顿的，例如页面有 2 个元素 A 和 B，用户在 A 上做 touchmove 手势，要求 B 也跟随移动，<movable-view> 就是一个典型的例子。一次 touchmove 事件的响应过程为：

1. touchmove 事件从视图层（Webview）抛到逻辑层（App Service）。
2. 逻辑层（App Service）处理 touchmove 事件，再通过 setData 来改变 B 的位置。

一次 touchmove 的响应需要经过 2 次的逻辑层和渲染层的通信以及一次渲染，通信的耗时比较大。此外 setData 渲染也会阻塞其它脚本执行，导致了整个用户交互的动画过程会有延迟。

实现方案

本方案基本的思路是减少通信的次数，让事件在视图层（Webview）响应。小程序的框架分为视图层（Webview）和逻辑层（App Service），这样分层的目的是管控，开发者的代码只能运行在逻辑层（App Service），而这个思路就必须要让开发者的代码运行在视图层（Webview）。

使用 WXS 函数用来响应小程序事件，目前只能响应内置组件的事件，不支持自定义组件事件。WXS 函数的除了纯逻辑的运算，还可以通过封装好的 ComponentDescriptor 实例来访问以及设置组件的 class 和样式，对于交互动画，设置 style 和 class 足够了。WXS 函数的例子如下：

```
const wxsFunction = function(event, ownerInstance) {
  const instance = ownerInstance.selectComponent('.classSelector') // 返回组件的实例
  instance.setStyle({
    'font-size': '14px', // 支持rpx
  })
  instance.getDataset()
  instance.setClass(className)
  // ...
  return false // 不往上冒泡，相当于调用了同时调用了stopPropagation和preventDefault
}
```

其中入参 event 是小程序事件对象基础上多了 event.instance 来表示触发事件的组件的 ComponentDescriptor 实例。 ownerInstance 表示的是触发事件的组件所在的组件的 ComponentDescriptor 实例，如果触发事件的组件是在页面内的， ownerInstance 表示的是页面实例。 ComponentDescriptor 的定义如下：

方法	参数	描述
selectComponent	selector 对象	返回组件的 ComponentDescriptor 实例
selectAllComponents	selector 对象数组	返回组件的 ComponentDescriptor 实例数组
setStyle	Object/string	设置组件样式，支持 rpx。设置的样式优先级比组件 WXML 里面定义的样式高。不能设置最顶层页面的样式
addClass/removeClass/ hasClass	String	设置组件的 class。设置的 class 优先级比组件 WXML 里面定义的 class 高。不能设置最顶层页面的 class
getDataset	无	返回当前组件/页面的 dataset 对象
callMethod	(funcName:string, args:object)	调用当前组件/页面在逻辑层（App Service）定义的函数。funcName 表示函数名称，args 表示函数的参数
requestAnimationFrame	Function	和原生 requestAnimationFrame 一样。用于设置动画
getState	无	返回一个 object 对象，当有局部变量需要存储起来

		后续使用的时候用这个方法
triggerEvent	(eventName, detail)	和组件的 triggerEvent 一致

WXS 运行在视图层（Webview），里面的逻辑毕竟能做的事件比较少，需要有一个机制和逻辑层（App Service）开发者的代码通信，上面的 `callMethod` 是 WXS 里面调用逻辑层（App Service）开发者的代码的方法，而 `WxsPropObserver` 是逻辑层（App Service）开发者的代码调用 WXS 逻辑的机制。

使用方法

- WXML 定义事件：

```
<wxs module="test" src="./test.wxs"></wxs>
<view
  change:prop="{{test.propObserver}}"
  prop="{{propValue}}"
  bindtouchmove="{{test.touchmove}}"
  class="movable"
></view>
```

上面的 `change:prop`（属性前面带 change:前缀）是在 prop 属性被设置的时候触发 WXS 函数，值必须用 `{{}}` 括起来。类似 Component 定义的 properties 里面的 observer 属性，在 `setData({propValue: newValue})` 调用之后会触发。

> **说明：**
> WXS 函数必须用 `{{}}` 括起来。当 prop 的值被设置 WXS 函数就会触发，而不只是值发生改变，所以在页面初始化的时候会调用一次 `WxsPropObserver` 的函数。

- WXS 文件 test.wxs 里面定义并导出事件处理函数和属性改变触发的函数：

```
module.exports = {
  touchmove(event, instance) {
    console.log('log event', JSON.stringify(event))
  },
  propObserver(newValue, oldValue, ownerInstance, instance) {
    console.log('prop observer', newValue, oldValue)
  },
}
```

- WXS 文件 test.wxs 里面定义并导出事件处理函数和属性改变触发的函数：

> **说明：**
> - 目前还不支持原生组件的事件、<input> 和 <textarea> 组件的 bindinput 事件。
> - 目前在 WXS 函数里面仅支持 console.log 方式打日志定位问题，注意连续的重复日志会被过滤掉。

引用

WXML 提供两种文件引用方式 `import` 和 `include` 。

import

`import` 可以在该文件中使用目标文件定义的 `template` ，如：
在 item.wxml 中定义了一个叫 `item` 的 `template` ：

```
<!-- item.wxml -->
<template name="item">
  <text>{{text}}</text>
</template>
```

在 index.wxml 中引用了 item.wxml，就可以使用 `item` 模板：

```
<import src="item.wxml" />
<template is="item" data="{{text: 'forbar'}}" />
```

import 的作用域

import 有作用域的概念，即只会 import 目标文件中定义的 template，而不会 import 目标文件 import 的 template。
例如：C import B，B import A，在 C 中可以使用 B 定义的 `template` ，在 B 中可以使用 A 定义的 `template` ，但是 C 不能使用 A 定义的 `template` 。

```
<!-- A.wxml -->
<template name="A">
  <text>A template</text>
</template>
```

```
<!-- B.wxml -->
<import src="a.wxml" />
<template name="B">
  <text>B template</text>
```

```
</template>
```

```
<!-- C.wxml -->
<import src="b.wxml" />
<template is="A" />
<!-- Error! Can not use tempalte when not import A. -->
<template is="B" />
```

include

include 可以将目标文件除了 <template/> <wxs/> 外的整个代码引入，相当于是拷贝到 include 位置，如：

```
<!-- index.wxml -->
<include src="header.wxml" />
<view>body</view>
<include src="footer.wxml" />
```

```
<!-- header.wxml -->
<view>header</view>
```

```
<!-- footer.wxml -->
<view>footer</view>
```

WXSS

最近更新时间：2024-03-21 15:51:52

WXSS（WeiXin Style Sheets）是一套样式语言，用于描述 WXML 组件的样式。它决定了 WXML 组件的显示效果。
为了方便广大前端开发者使用，WXSS 借鉴了 CSS 的大部分特性。同时，为了更好地适应开发 TMF 小程序的需求，WXSS 进行了一些扩充和修改。
与 CSS 相比，WXSS 扩展了以下特性：

> ⚠ **注意：**
> iOS 8 及以下的 iOS 版本，如使用 flexbox 布局，需添加 `display: -webkit-flex` 属性。

尺寸单位

rpx（responsive pixel）：可以根据屏幕宽度进行自适应。规定屏幕宽为750rpx。例如在 iPhone 6 上，屏幕宽度为375px，共有750个物理像素，则750rpx = 375px = 750物理像素，1rpx = 0.5px = 1物理像素。
在腾讯云小程序平台开发中，我们使用 rpx 作为尺寸单位，以适应不同屏幕尺寸的需求。下面是 rpx 和 px 之间的换算关系：

- 设备屏幕宽度为 750px 时，1rpx = 0.5px。
- 设备屏幕宽度为 750px 时，1px = 2rpx。

根据不同的设备屏幕宽度，rpx 和 px 的换算关系也会有所不同。例如：

- 在 iPhone 5 上，设备屏幕宽度为 320px，1rpx = 0.42px，1px = 2.34rpx。
- 在 iPhone 6 上，设备屏幕宽度为 375px，1rpx = 0.5px，1px = 2rpx。
- 在 iPhone 6 Plus 上，设备屏幕宽度为 414px，1rpx = 0.552px，1px = 1.81rpx。

建议在设计应用或小程序时，设计师可以用 iPhone 6作为视觉稿的标准。但需要注意的是，在较小的屏幕上可能会出现一些毛刺，因此在开发时需要尽量避免这种情况。

样式导入

使用 `@import` 语句可以导入外联样式表， `@import` 后跟需要导入的外联样式表的相对路径，用 `;` 表示语句结束。
示例代码：

```
/** common.wxss **/
.small-p {
  padding:5px;
}
```

```
/** app.wxss **/
@import "common.wxss";
.middle-p {
  padding:15px;
}
```

内联样式

框架组件上支持使用 style、class 属性来控制组件的样式。

- style：静态的样式统一写到 class 中。style 接收动态的样式，在运行时会进行解析，请尽量避免将静态的样式写进 style 中，以免影响渲染速度。

```
<view style="color:{{color}};" />
```

- class：用于指定样式规则，其属性值是样式规则中类选择器名(样式类名)的集合，样式类名不需要带上 `.`，样式类名之间用空格分隔。

```
<view class="normal_view" />
```

选择器

目前支持的选择器有：

选择器	样例	样例描述
.class	.intro	选择所有拥有 class="intro" 的组件
#id	#firstname	选择拥有 id="firstname" 的组件
element	view	选择所有 view 组件
element, element	view, checkbox	选择所有文档的 view 组件和所有的 checkbox 组件
::aftter	view::after	在 view 组件后边插入内容
::before	view::before	在 view 组件前边插入内容

全局样式与局部样式

定义在 app.wxss 中的样式为全局样式，作用于每一个页面。在 page 的 wxss 文件中定义的样式为局部样式，只作用在对应的页面，并会覆盖 app.wxss 中相同的选择器。

小程序运行环境与机制

最近更新时间：2024-04-10 15:12:32

运行环境

小程序的运行环境

TCMPP 小程序可在多种平台上运行：iOS/iPadOS 微信客户端、Android 微信客户端、Mac 微信客户端和用于调试的微信开发者工具等。
不同运行环境下，脚本执行环境和用于组件渲染的环境是不同的，性能表现也存在差异：

- 在 iOS、iPadOS 和 Mac OS 上，小程序逻辑层的 JavaScript 代码运行在 JavaScriptCore 中，视图层是由 WKWebView 来渲染的，环境有 iOS 14、iPad OS 14、Mac OS 11.4 等。
- 在 Android 上，小程序逻辑层的 JavaScript 代码运行在 V8 中，视图层是由基于 Mobile Chromium 内核的微信自研 XWeb 引擎来渲染的。
- 在开发工具上，小程序逻辑层的 JavaScript 代码是运行在 NW.js 中，视图层是由 Chromium Webview 来渲染的。JavaScriptCore 无法开启 JIT 编译 (Just-In-Time Compiler)，同等条件下的运行性能要明显低于其他平台。

平台差异

各运行环境区别如下：

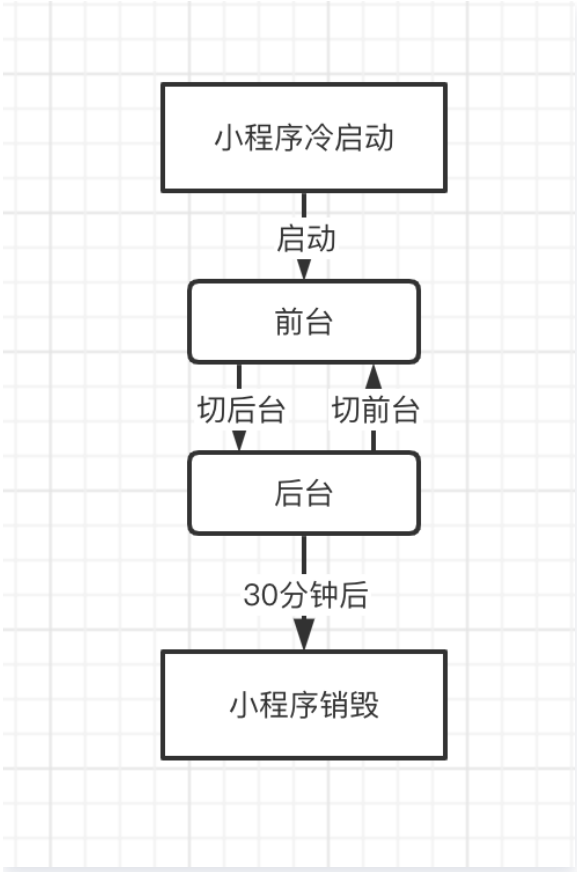
- JavaScript 语法和 API 支持不一致：语法上开发者可以通过开启 ES6 转 ES5 的功能来规避；此外，小程序基础库内置了必要的 Polyfill，来弥补 API 的差异。
- WXSS 渲染表现不一致：尽管可以通过开启来规避大部分的问题，还是建议开发者需要在各端分别检查小程序的真实表现。

开发者工具仅供调试使用，最终的表现以客户端为准。

运行机制

小程生命周期

小程序从启动到最终被销毁，会经历多种状态，不同状态下表现不同。



小程序启动

从用户的角度来看，小程序的启动可以分为两种情况：冷启动和热启动。

- **冷启动**：如果用户首次打开，或小程序销毁后被用户再次打开，此时小程序需要重新加载启动，即冷启动。
- **热启动**：如果用户已经打开过某小程序，然后在一定时间内再次打开该小程序，此时小程序并未被销毁，只是从后台状态进入前台状态，这个过程就是热启动。

从小程序生命周期的角度来看，一般将冷启动称为**启动**，而将热启动称为**后台切前台**。

前台与后台

小程序启动后，界面被展示给用户，此时小程序处于**前台**状态。
当用户**关闭**小程序时，小程序并没有真正被关闭，而是进入了**后台**状态，此时小程序还可以短暂运行一小段时间，但部分 API 的使用会受到限制。切后台的方式包括但不限于以下几种：

- 单击右上角胶囊按钮离开小程序；
- iOS 系统从屏幕左侧右滑离开小程序；
- Android 系统单击返回键离开小程序；
- 小程序前台运行时，通过手势或 Home 键直接把微信切至后台；
- 小程序前台运行时直接锁屏；
- 当用户再次进入微信并打开小程序，小程序又会重新进入**前台**状态。

小程序销毁

如果用户很久没有使用小程序，或者系统资源紧张，小程序会被**销毁**，即完全终止运行。具体包括以下几种情形：

- 当小程序进入后台并被**挂起**后，如果很长时间（目前是30分钟）都未再次进入前台，小程序会被销毁。
- 当小程序占用系统资源过高，可能会被系统销毁或被微信客户端主动回收。
- 在 iOS 系统上，当微信客户端在一定时间间隔内连续收到系统内存告警时，会根据一定的策略，主动销毁小程序，并提示用户**运行内存不足，请重新打开该小程序**。

具体策略会持续进行调整优化。

调试

开发者可以通过以下工具来调试小程序：

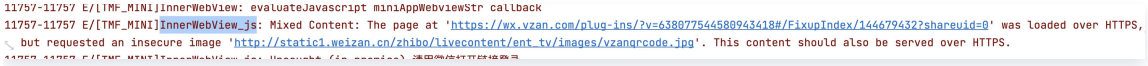
- **vConsole**：vConsole 可以在手机上查看 console API 输出的日志以及额外的调试信息。使用它可以更轻松地找到并解决问题。
- **Source Map**：Source Map 可以还原 JS 错误堆栈，通过它可以追踪到代码哪里出错，从而更加快速地调试代码。
- **日志**：利用小程序自带的日志功能，可以快捷地排查漏洞并定位问题。日志记录了一些重要的操作流程以及错误日志信息，开发者可以通过分析这些日志信息定位并解决问题。

以上提到的工具都是开发小程序时必备的辅助工具，它们可以方便开发者更加快速地调试小程序代码，提高开发效率。

问题总结

小程序内 webview 使用

当小程序页面，直接使用 webview 加载 H5 页面时，如果页面中有错误或页面中资源有 HTTP 请求，如下：



加载过程中可能会出现如下错误页面：

解决办法：

将 http 资源替换为 https 资源。

QQ 分享错误

```
Caused by: java.lang.ClassNotFoundException: Didn't find class
 "org.apache.http.conn.scheme.SchemeRegistry"
```

QQ 分享错误时如果出现上面的错误，请添加如下配置：

```
<application
<uses-library android:name="org.apache.http.legacy" android:required="false"/>
</application>
```

自定义组件
自定义模块组件

最近更新时间：2024-03-21 15:51:52

在腾讯云小程序平台开发中，开发者可以将页面内的功能模块抽象成自定义组件，以便在不同的页面中重复使用。同时，也可以将复杂的页面拆分成多个低耦合的模块，有助于代码维护和复用。

创建自定义组件

一个自定义组件由 json、wxml、wxss、js 四个文件组成。要编写一个自定义组件，首先需要在 json 文件中进行自定义组件声明，将 component 字段设为 true，以便让开发者知道这一组文件是一个自定义组件。

```
{
  "component": true
}
```

同时，还要在 `wxml` 文件中编写组件模板，在 `wxss` 文件中加入组件样式，它们的写法与页面的写法类似。具体细节和注意事项参见 组件模板和样式 。
代码示例：

```
<!-- 这是自定义组件的内部WXML结构 -->
<view class="inner">
  {{innerText}}
</view>
<slot></slot>
```

```
/* 这里的样式只应用于这个自定义组件 */
.inner {
  color: red;
}
```

> **说明：**
> 在组件 wxss 中不应使用 ID 选择器、属性选择器和标签名选择器。

在自定义组件的 `js` 文件中，需要使用 `Component()` 来注册组件，并提供组件的属性定义、内部数据和自定义方法。
组件的属性值和内部数据将被用于组件 `wxml` 的渲染，其中，属性值是可由组件外部传入的。更多细节参见 Component 构造器 。

代码示例：

```
Component({
  properties: {
    // 这里定义了innerText属性，属性值可以在组件使用时指定
    innerText: {
      type: String,
      value: 'default value',
    }
  },
  data: {
    // 这里是一些组件内部数据
    someData: {}
  },
  methods: {
    // 这里是一个自定义方法
    customMethod() {}
  }
})
```

使用自定义组件

使用已注册的自定义组件前，首先要在页面的 `json` 文件中进行引用声明。此时需要提供每个自定义组件的标签名和对应的自定义组件文件路径：

```
{
  "usingComponents": {
    "component-tag-name": "path/to/the/custom/component"
  }
}
```

这样，在页面的 `wxml` 中就可以像使用基础组件一样使用自定义组件。节点名即自定义组件的标签名，节点属性即传递给组件的属性值。

> **说明：**
> 在 `app.json` 中声明 `usingComponents` 字段，在此处声明的自定义组件视为全局自定义组件，在小程序内的页面或自定义组件中可以直接使用而无需再声明。

```
<view>
  <!-- 以下是对一个自定义组件的引用 -->
  <component-tag-name inner-text="Some text"></component-tag-name>
</view>
```

自定义组件的 wxml 节点结构在与数据结合之后，将被插入到引用位置内。

细节注意事项

需要注意的细节：

- 自定义组件的标签名和 WXML 节点标签名仅支持小写字母、中划线和下划线的组合。
- 自定义组件也是可以引用自定义组件的，引用方法类似于页面引用自定义组件的方式（使用 usingComponents 字段）。
- 自定义组件和页面所在项目根目录名不能以“wx-”为前缀，否则会报错。

是否在页面文件中使用 usingComponents 会使得页面的 this 对象的原型稍有差异，包括：

- 使用 usingComponents 页面的原型与不使用时不一致，即 Object.getPrototypeOf(this) 结果不同。
- 使用 usingComponents 时会多一些方法，如 selectComponent 。
- 出于性能考虑，使用 usingComponents 时， setData 内容不会被直接深复制，即 this.setData({ field: obj }) 后 this.data.field === obj 。（深复制会在这个值被组件间传递时发生。）

> **说明：**
> 如果页面比较复杂，新增或删除 usingComponents 定义段时建议重新测试一下。

组件模板和样式

最近更新时间：2024-03-21 15:51:52

在腾讯云小程序平台开发中，自定义组件拥有自己的 wxml 模板和 wxss 样式，可以通过定义自定义组件的 wxml 和 wxss 文件，来实现组件的结构和样式。

组件模板

组件模板的写法与页面模板相同。组件模板与组件数据结合后生成的节点树，将被插入到组件的引用位置上。
在组件模板中可以提供一个 `<slot>` 节点，用于承载组件引用时提供的子节点。
代码示例：

```
<!-- 组件模板 -->
<view class="wrapper">
  <view>这里是组件的内部节点</view>
  <slot></slot>
</view>
```

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot> 的位置上 -->
    <view>这里是插入到组件slot中的内容</view>
  </component-tag-name>
</view>
```

> **说明：**
> 在模板中引用到的自定义组件及其对应的节点名需要在 `json` 文件中显式定义，否则会被当作一个无意义的节点。除此以外，节点名也可以被声明为 抽象节点 。

模板数据绑定

与普通的 WXML 模板类似，可以使用数据绑定，这样就可以向子组件的属性传递动态数据。
代码示例：

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name prop-a="{{dataFieldA}}" prop-b="{{dataFieldB}}">
    <!-- 这部分内容将被放置在组件 <slot> 的位置上 -->
    <view>这里是插入到组件slot中的内容</view>
```

```
  </component-tag-name>
</view>
```

在上述，组件的属性 propA 和 propB 将收到页面传递的数据。页面可以通过 setData 来改变绑定的数据字段。

> **说明：**
> 这样的数据绑定只能传递 JSON 兼容数据。

组件 wxml 的 slot

在组件的 wxml 中可以包含 slot 节点，用于承载组件使用者提供的 wxml 结构。
默认情况下，一个组件的 wxml 中只能有一个 slot。需要使用多 slot 时，可以在组件 js 中声明启用。

```
Component({
  options: {
    multipleSlots: true // 在组件定义时的选项中启用多slot支持
  },
  properties: { /* ... */ },
  methods: { /* ... */ }
})
```

此时，可以在这个组件的 wxml 中使用多个 slot，以不同的 name 来区分。

```
<!-- 组件模板 -->
<view class="wrapper">
  <slot name="before"></slot>
  <view>这里是组件的内部细节</view>
  <slot name="after"></slot>
</view>
```

使用时，用 slot 属性来将节点插入到不同的 slot 上。

```
<!-- 引用组件的页面模板 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot name="before"> 的位置上 -->
    <view slot="before">这里是插入到组件slot name="before"中的内容</view>
    <!-- 这部分内容将被放置在组件 <slot name="after"> 的位置上 -->
    <view slot="after">这里是插入到组件slot name="after"中的内容</view>
  </component-tag-name>
</view>
```

组件样式

组件对应 wxss 文件的样式，只对组件 wxml 内的节点生效。编写组件样式时，需要注意以下几点：

- 组件和引用组件的页面不能使用id选择器（ #a ）、属性选择器（ [a] ）和标签名选择器，请改用 class 选择器。
- 组件和引用组件的页面中使用后代选择器（ .a .b ）在一些极端情况下会有非预期的表现，如遇，请避免使用。
- 子元素选择器（ .a>.b ）只能用于 view 组件与其子节点之间，用于其他组件可能导致非预期的情况。
- 继承样式，如 font 、 color ，会从组件外继承到组件内。
- 除继承样式外， app.wxss 中的样式、组件所在页面的样式对自定义组件无效。

```
#a {
} /* 在组件中不能使用 */
[a] {
} /* 在组件中不能使用 */
button {
} /* 在组件中不能使用 */
.a > .b {
} /* 除非 .a 是 view 组件节点，否则不一定会生效 */
```

除此以外，组件可以指定它所在节点的默认样式，使用 : host 选择器。
代码示例：

```
/* 组件 custom-component.wxss */
:host {
  color: yellow;
}
```

```
<!-- 页面的 WXML -->
<custom-component>这段文本是黄色的</custom-component>
```

外部样式类

有时，组件希望接受外部传入的样式类。此时可以在 Component 中用 externalClasses 定义段定义若干个外部样式类。
这个特性可以用于实现类似于 view 组件的 hover-class 属性：页面可以提供一个样式类，赋予 view 的 hover-class ，这个样式类本身写在页面中而非 view 组件的实现中。

> ⚠ **注意：**
> 在同一个节点上使用普通样式类和外部样式类时，两个类的优先级是未定义的，因此最好避免这种情况。

代码示例：

```
/* 组件 custom-component.js */
Component({
  externalClasses: ['my-class']
})
```

```
<!-- 组件 custom-component.wxml -->
<custom-component class="my-class">
  这段文本的颜色由组件外的 class 决定
</custom-component>
```

这样，组件的使用者可以指定这个样式类对应的 class，就像使用普通属性一样。
代码示例：

```
<!-- 页面的 WXML -->
<custom-component my-class="red-text" />
```

```
.red-text {
  color: red;
}
```

使组件接受全局样式

默认情况下，自定义组件的样式只受到自定义组件 wxss 的影响。除非以下两种情况：

- app.wxss 或页面的 wxss 中使用了标签名选择器（或一些其他特殊选择器）来直接指定样式，这些选择器会影响到页面和全部组件。通常情况下这是不推荐的做法。
- 在特定的自定义组件激活了 addGlobalClass 选项，这使得这个自定义组件能被 app.wxss 或页面的 wxss 中的所有的样式定义影响。

要激活 addGlobalClass 选项，只需要在 Component 构造器中将 options.addGlobalClass 字段置为 true 。

> ⚠ **注意：**
> 当激活了 **addGlobalClass** 选项后，存在外部样式污染组件样式的风险，请谨慎选择。

代码示例：

```
/* 组件 custom-component.js */
Component({
  options: {
```

```
    addGlobalClass: true,
  }
})
```

```
<!-- 组件 custom-component.wxml -->
<text class="red-text">
  这段文本的颜色由 `app.wxss` 和页面 `wxss` 中的样式定义来决定
</text>
```

```
/* app.wxss */
.red-text {
  color: red;
}
```

Component 构造器

最近更新时间：2024-03-21 15:51:52

定义段与示例方法

Component 构造器可用于定义组件，调用 Component 构造器时可以指定组件的属性、数据、方法等。

定义段	类型	是否必填	描述
properties	Object Map	否	组件的对外属性，是属性名到属性设置的映射表，属性设置中可包含三个字段，`type` 表示属性类型、`value` 表示属性初始值、`observer` 表示属性值被更改时的响应函数
observer	Object	否	组件的内部数据，和 properties 一同用于组件的模板渲染
data	Object	否	组件数据字段监听器，用于监听 properties 和 data 的变化，详情请参见 数据监听器
observers	Object	否	组件的方法，包括事件响应函数和任意的自定义方法，关于事件响应函数的使用，详情请参见 组件事件
methods	Object	否	类似于 mixins 和 traits 的组件间代码复用机制，详情请参见 behaviors
behaviors	String Array	否	组件生命周期函数，在组件实例刚刚被创建时执行，注意此时不能调用 setData ，详情请参见 组件生命周期
created	Function	否	组件生命周期函数，在组件实例刚刚被创建时执行，注意此时不能调用 setData ，详情请参见 组件生命周期
attached	Function	否	组件生命周期函数，在组件布局完成后执行，详情请参见 组件生命周期
ready	Function	否	组件生命周期函数，在组件实例进入页面节点树时执行，详情请参见 组件生命周期
moved	Function	否	组件生命周期函数，在组件实例被从页面节点树移除时执行，详情请参见 组件生命周期
detached	Function	否	组件生命周期函数，在组件实例被从页面节点树移除时执行，详情请参见 组件生命周期
relations	Object	否	组件间关系定义，详情请参见 组件间关系
externalCla	String	否	组件接受的外部样式类，详情请参见 组件模板和样式

sses	Array		
options	Object Map	否	一些选项（文档中介绍相关特性时会涉及具体的选项设置，这里暂不列举）
lifetimes	Object	否	组件生命周期声明对象，详情请参见 组件生命周期
pageLifetimes	Object	否	组件所在页面的生命周期声明对象，支持页面的 show 、 hide 等生命周期，详情请参见 组件生命周期
definitionFilter	Function	否	定义段过滤器，用于自定义组件扩展，详情请参见 自定义组件扩展

生成的组件实例可以在组件的方法、生命周期函数和属性 observer 中通过 this 访问。组件的通用属性如下表：

属性名	类性	描述
is	string	组件的文件路径
id	string	节点 id
dataset	string	节点 dataset
data	object	组件数据，包括内部数据和属性值
properties	object	组件数据，包括内部数据和属性值（与 `data` 一致）

组件的通用方法如下表：

方法名	参数	描述
setData	Object newData	设置 data 并执行视图层渲染
hasBehavior	Object behavior	检查组件是否具有 behavior （检查时会递归检查被直接或间接引入的所有 behavior）
triggerEvent	String name, Object detail, Object options	触发事件，详情请参见 组件事件
createSelectorQuery	–	创建一个 SelectorQuery 对象，选择器选取范围为这个组件实例内
createInter	–	创建一个 IntersectionObserver 对象，选择器选取范围为这个组

sectionObserver		件实例内
selectComponent	String selector	使用选择器选择组件实例节点，返回匹配到的第一个组件实例对象（会被 wx://component-export 影响）
selectAllComponents	String selector	使用选择器选择组件实例节点，返回匹配到的全部组件实例对象组成的数组
getRelationNodes	String relationKey	获取这个关系所对应的所有关联节点，详情请参见 组件间关系
groupSetData	Function callback	立刻执行 callback ，其中的多个 setData 之间不会触发界面绘制（只有某些特殊场景中需要，如用于在不同组件同时 setData 时进行界面绘制同步）

示例代码：

```
Component({

  behaviors: [],

  properties: {
    myProperty: { // 属性名
      type: String, // 类型（必填），目前接受的类型包括：String, Number, Boolean, Object,
Array, null（表示任意类型）
      value: '', // 属性初始值（可选），如果未指定则会根据类型选择一个
      observer(newVal, oldVal, changedPath) {
        // 属性被改变时执行的函数（可选），也可以写成在methods段中定义的方法名字符串,
如：'_propertyChange'
        // 通常 newVal 就是新设置的数据， oldVal 是旧数据
      }
    },
    myProperty2: String // 简化的定义方式
  },
  data: {}, // 私有数据，可用于模板渲染

  lifetimes: {
    // 生命周期函数，可以为函数，或一个在methods段中定义的方法名
    attached() { },
    moved() { },
    detached() { },
  },

  // 生命周期函数，可以为函数，或一个在methods段中定义的方法名
  attached() { }, // 此处attached的声明会被lifetimes字段中的声明覆盖
```

```
  ready() { },

  pageLifetimes: {
    // 组件所在页面的生命周期函数
    show() { },
    hide() { },
    resize() { },
  },

  methods: {
    onMyButtonTap() {
      this.setData({
        // 更新属性和数据的方法与更新页面数据的方法类似
      })
    },
    // 内部方法建议以下划线开头
    _myPrivateMethod() {
      // 这里将 data.A[0].B 设为 'myPrivateData'
      this.setData({
        'A[0].B': 'myPrivateData'
      })
    },
    _propertyChange(newVal, oldVal) {

    }
  }

})
```

在 properties 定义段中，属性名采用驼峰写法（ propertyName ）；在 wxml 中，指定属性值时则对应使用连字符写法（ component-tag-name property-name="attr value" ），应用于数据绑定时采用驼峰写法（ attr="{{propertyName}}" ）。

使用 Component 构造器构造页面

事实上，小程序的页面也可以视为自定义组件。因而，页面也可以使用 Component 构造器构造，拥有与普通组件一样的定义段与实例方法。但此时要求对应 json 文件中包含 usingComponents 定义段。
此时，组件的属性可以用于接收页面的参数，如访问页面 /pages/index/index?paramA=123&paramB=xyz ，如果声明有属性 paramA 或 paramB ，则它们会被赋值为123或 xyz。
页面的生命周期方法（即 on 开头的方法），应写在 methods 定义段中。
示例代码：

```
{
  "usingComponents": {}
```

```
}
```

```
Component({

  properties: {
    paramA: Number,
    paramB: String,
  },

  methods: {
    onLoad() {
      this.data.paramA // 页面参数 paramA 的值
      this.data.paramB // 页面参数 paramB 的值
    }
  }

})
```

说明：

- 使用 this.data 可以获取内部数据和属性值，但不要直接修改它们，应使用 setData 修改。
- 生命周期函数无法在组件方法中通过this访问到。
- 属性名应避免以 data 开头，即不要命名成 dataXyz 这样的形式，因为在 WXML 中，data-xyz="" 会被作为节点 dataset 来处理，而不是组件属性。
- 在一个组件的定义和使用时，组件的属性名和 data 字段相互间都不能冲突（尽管它们位于不同的定义段中）。
- 对象类型的属性和 data 字段中可以包含函数类型的子字段，即可以通过对象类型的属性字段来传递函数。低于这一版本的基础库不支持这一特性。
- 对于 type 为 Object 或 Array 的属性，如果通过该组件自身的 this.setData 来改变属性值的一个子字段，则依旧会触发属性 observer ，且 observer 接收到的 newVal 是变化的那个子字段的值，oldVal 为空，changedPath 包含子字段的字段名相关信息。
- 位于 slot 中的自定义组件没有触发 pageLifetimes 中声明的页面生命周期。

组件事件

最近更新时间：2024-03-21 15:51:52

组件间通信

组件间的基本通信方式有以下几种。

- **WXML 数据绑定**：用于父组件向子组件的指定属性设置数据，仅能设置 JSON 兼容数据开始，还可以在数据中包含函数。详情请参见 组件模板和样式 。
- **事件**：用于子组件向父组件传递数据，可以传递任意数据。

如果以上两种方式不足以满足需要，父组件还可以通过 this.selectComponent 方法获取子组件实例对象，即可直接访问组件的任意数据和方法。

监听事件

事件系统是组件间通信的主要方式之一。自定义组件可以触发任意的事件，引用组件的页面可以监听这些事件。关于事件的基本概念和用法，参见 WXML - 事件 。
监听自定义组件事件的方法与监听基础组件事件的方法完全一致。
代码示例：

```
<!-- 当自定义组件触发“myevent”事件时，调用“onMyEvent”方法 -->
<component-tag-name bindmyevent="onMyEvent" />
<!-- 或者可以写成 -->
<component-tag-name bind:myevent="onMyEvent" />
```

```
Page({
  onMyEvent(e) {
    e.detail // 自定义组件触发事件时提供的detail对象
  }
})
```

触发事件

自定义组件触发事件时，需要使用 triggerEvent 方法，指定事件名、detail 对象和事件选项：

```
<!-- 在自定义组件中 -->
<button bindtap="onTap">点击这个按钮将触发“myevent”事件</button>
```

```
Component({
  properties: {},
```

```
  methods: {
    onTap() {
      const myEventDetail = {} // detail对象，提供给事件监听函数
      const myEventOption = {} // 触发事件的选项
      this.triggerEvent('myevent', myEventDetail, myEventOption)
    }
  }
})
```

触发事件的选项包括：

选项名	类型	是否必填	默认值	描述
bubbles	Boolean	否	false	事件是否冒泡
composed	Boolean	否	false	事件是否可以穿越组件边界，为 false 时，事件将只能在引用组件的节点树上触发，不进入其他任何组件内部
capturePhase	Boolean	否	false	事件是否拥有捕获阶段

关于冒泡和捕获阶段的概念，请阅读参见 WXML – 事件 章节中的相关说明。

```
// 页面 page.wxml
<another-component bindcustomevent="pageEventListener1">
  <my-component bindcustomevent="pageEventListener2"></my-component>
</another-component>
```

```
// 组件 another-component.wxml
<view bindcustomevent="anotherEventListener">
  <slot />
</view>
```

```
// 组件 my-component.wxml
<view bindcustomevent="myEventListener">
  <slot />
</view>
```

```
// 组件 my-component.js
Component({
```

```
  methods: {
    onTap() {
      this.triggerEvent('customevent', {}) // 只会触发 pageEventListener2
      this.triggerEvent('customevent', {}, {bubbles: true}) // 会依次触发
pageEventListener2 、 pageEventListener1
      this.triggerEvent('customevent', {}, {bubbles: true, composed: true}) // 会依次触发
pageEventListener2 、 anotherEventListener 、 pageEventListener1
    }
  }
})
```

组件生命周期

最近更新时间：2024-03-21 15:51:52

组件主要生命周期

组件的生命周期，指的是组件自身的一些函数，这些函数在特殊的时间点或遇到一些特殊的框架事件时被自动触发。其中，最重要的生命周期是 `created` 、 `attached` 、 `detached` ，包含一个组件实例生命流程的最主要时间点。

- 组件实例刚刚被创建好时， `created` 生命周期被触发。此时，组件数据 `this.data` 就是在 `Component` 构造器中定义的数据 `data` 。**此时还不能调用** `setData` 。通常情况下，这个生命周期只应该用于给组件 this 添加一些自定义属性字段。
- 在组件完全初始化完毕、进入页面节点树后， `attached` 生命周期被触发。此时， `this.data` 已被初始化为组件的当前值。这个生命周期很有用，绝大多数初始化工作可以在这个时机进行。
- 在组件离开页面节点树后， `detached` 生命周期被触发。退出一个页面时，如果组件还在页面节点树中，则 `detached` 会被触发。

定义生命周期方法

生命周期方法可以直接定义在 `Component` 构造器的第一级参数中。
组件的生命周期也可以在 `lifetimes` 字段内进行声明（这是推荐的方式，其优先级最高）。
代码示例：

```
Component({
  lifetimes: {
    attached() {
      // 在组件实例进入页面节点树时执行
    },
    detached() {
      // 在组件实例被从页面节点树移除时执行
    },
  },
  // 以下是旧式的定义方式
  attached() {
    // 在组件实例进入页面节点树时执行
  },
  detached() {
    // 在组件实例被从页面节点树移除时执行
  },
  // ...
})
```

在 `behaviors` 中也可以编写生命周期方法，同时不会与其他 `behaviors` 中的同名生命周期相互覆盖。

> **注意：**
> 如果一个组件多次直接或间接引用同一个 `behavior`，这个 `behavior` 中的生命周期函数在一个引用时机内只会执行一次。

可用的全部生命周期如下表所示：

生命周期	参数	描述
created	无	在组件实例刚刚被创建时执行
attached	无	在组件实例进入页面节点树时执行
ready	无	在组件在视图层布局完成后执行
moved	无	在组件实例被移动到节点树另一个位置时执行
detached	无	在组件实例被从页面节点树移除时执行
error	Object Error	每当组件方法抛出错误时执行

组件所在页面的生命周期

还有一些特殊的生命周期，它们并非与组件有很强的关联，但有时组件需要获知，以便组件内部处理。这样的生命周期称为“组件所在页面的生命周期”，在 `pageLifetimes` 定义段中定义。其中可用的生命周期包括：

生命周期	参数	描述
show	无	组件所在的页面被展示时执行
hide	无	组件所在的页面被隐藏时执行
resize	Object Size	组件所在的页面尺寸变化时执行

代码示例：

```
Component({
  pageLifetimes: {
    show() {
      // 页面被展示
    },
    hide() {
      // 页面被隐藏
    },
    resize(size) {
      // 页面尺寸变化
```

```
    }
  }
})
```

behaviors

最近更新时间：2024-03-21 15:51:52

定义和使用 behaviors

在腾讯云小程序平台开发中，behaviors 是用于组件间代码共享的特性，类似于一些编程语言中的“mixins”或“traits”。每个 behavior 可以包含一组属性、数据、生命周期函数和方法，组件引用它时，它的属性、数据和方法会被合并到组件中，生命周期函数也会在对应时机被调用。每个组件可以引用多个 behavior，而 behavior 也可以引用其他 behavior。

`behavior` 需要使用 `Behavior()` 构造器定义。

代码示例：

```
// my-behavior.js
module.exports = Behavior({
  behaviors: [],
  properties: {
    myBehaviorProperty: {
      type: String
    }
  },
  data: {
    myBehaviorData: {}
  },
  attached() {},
  methods: {
    myBehaviorMethod() {}
  }
})
```

组件引用时，在 `behaviors` 定义段中将它们逐个列出即可。

代码示例：

```
// my-component.js
const myBehavior = require('my-behavior')
Component({
  behaviors: [myBehavior],
  properties: {
    myProperty: {
      type: String
    }
  },
  data: {
```

```
    myData: {}
  },
  attached() {},
  methods: {
    myMethod() {}
  }
})
```

在上述示例中， my-component 组件定义中加入了 my-behavior ，而 my-behavior 中包含有 myBehaviorProperty 属性、 myBehaviorData 数据字段、 myBehaviorMethod 方法和一个 attached 生命周期函数。这将使得 my-component 中最终包含 myBehaviorProperty 、 myProperty 两个属性， myBehaviorData 、 myData 两个数据字段，和 myBehaviorMethod 、 myMethod 两个方法。当组件触发 attached 生命周期时，会依次触发 my-behavior 中的 attached 生命周期函数和 my-component 中的 attached 生命周期函数。

字段的覆盖和组合规则

组件和它引用的 behavior 中可以包含同名的字段，对这些字段的处理方法如下：

- 如果有同名的属性或方法，组件本身的属性或方法会覆盖 behavior 中的属性或方法，如果引用了多个 behavior ，在定义段中靠后 behavior 中的属性或方法会覆盖靠前的属性或方法；
- 如果有同名的数据字段，且数据是对象类型，会进行对象合并，如果是非对象类型则会进行相互覆盖；
- 生命周期函数不会相互覆盖，而是在对应触发时机被逐个调用。如果同一个 behavior 被一个组件多次引用，它定义的生命周期函数只会被执行一次。

内置 behaviors

自定义组件可以通过引用内置的 behavior 来获得内置组件的一些行为。

```
Component({
  behaviors: ['wx://form-field']
})
```

在上例中， wx://form-field 代表一个内置 behavior ，它使得这个自定义组件有类似于表单控件的行为。
内置 behavior 往往会为组件添加一些属性。在没有特殊说明时，组件可以覆盖这些属性来改变它的 type 或添加 observer 。

wx://form-field

使自定义组件有类似于表单控件的行为。 form 组件可以识别这些自定义组件，并在 submit 事件中返回组件的字段名及其对应字段值。这将为它添加以下两个属性。

属性名	类型	描述

name	string	在表单中的字段名
value	任意	在表单中的字段值

wx://component-export

使自定义组件支持 `export` 定义段。这个定义段可以用于指定组件被 `selectComponent` 调用时的返回值。未使用这个定义段时， `selectComponent` 将返回自定义组件的 `this` 插件的自定义组件将返回 `null` 。使用这个定义段时，将以这个定义段的函数返回值代替。

代码示例：

```
// 自定义组件 my-component 内部
Component({
  behaviors: ['wx://component-export'],
  export() {
    return {myField: 'myValue'}
  }
})
```

```
<!-- 使用自定义组件时 -->
<my-component id="the-id" />
```

```
this.selectComponent('#the-id') // 等于 { myField: 'myValue' }
```

组件间关系

最近更新时间：2024-03-21 15:51:52

定义和使用组件间关系

有时需要实现这样的组件：

```
<custom-ul>
  <custom-li>item 1</custom-li>
  <custom-li>item 2</custom-li>
</custom-ul>
```

这个示例中，`custom-ul` 和 `custom-li` 都是自定义组件，它们有相互间的关系，相互间的通信往往比较复杂。此时在组件定义时加入 `relations` 定义段，可以解决这样的问题。
代码示例：

```
// path/to/custom-ul.js
Component({
  relations: {
    './custom-li': {
      type: 'child', // 关联的目标节点应为子节点
      linked(target) {
        // 每次有custom-li被插入时执行，target是该节点实例对象，触发在该节点attached生命周期之后
      },
      linkChanged(target) {
        // 每次有custom-li被移动后执行，target是该节点实例对象，触发在该节点moved生命周期之后
      },
      unlinked(target) {
        // 每次有custom-li被移除时执行，target是该节点实例对象，触发在该节点detached生命周期之后
      }
    }
  },
  methods: {
    _getAllLi() {
      // 使用getRelationNodes可以获得nodes数组，包含所有已关联的custom-li，且是有序的
      const nodes = this.getRelationNodes('path/to/custom-li')
    }
  },
  ready() {
```

```
    this._getAllLi()
  }
})
```

```
// path/to/custom-li.js
Component({
  relations: {
    './custom-ul': {
      type: 'parent', // 关联的目标节点应为父节点
      linked(target) {
        // 每次被插入到custom-ul时执行，target是custom-ul节点实例对象，触发在attached生
命周期之后
      },
      linkChanged(target) {
        // 每次被移动后执行，target是custom-ul节点实例对象，触发在moved生命周期之后
      },
      unlinked(target) {
        // 每次被移除时执行，target是custom-ul节点实例对象，触发在detached生命周期之后
      }
    }
  }
})
```

> **注意：**
> 必须在两个组件定义中都加入 relations 定义，否则不会生效。

关联一类组件

如需将一类组件进行关联处理，请参考以下示例代码：

```
<custom-form>
  <view>
    input
    <custom-input></custom-input>
  </view>
  <custom-submit>submit</custom-submit>
</custom-form>
```

`custom-form` 组件想要关联 `custom-input` 和 `custom-submit` 两个组件。此时，如果这两个组件都有同一个 behavior：

```
// path/to/custom-form-controls.js
module.exports = Behavior({
  // ...
})
```

```
// path/to/custom-input.js
const customFormControls = require('./custom-form-controls')
Component({
  behaviors: [customFormControls],
  relations: {
    './custom-form': {
      type: 'ancestor', // 关联的目标节点应为祖先节点
    }
  }
})
```

```
// path/to/custom-submit.js
const customFormControls = require('./custom-form-controls')
Component({
  behaviors: [customFormControls],
  relations: {
    './custom-form': {
      type: 'ancestor', // 关联的目标节点应为祖先节点
    }
  }
})
```

则在 `relations` 关系定义中，可使用这个 behavior 来代替组件路径作为关联的目标节点：

```
// path/to/custom-form.js
const customFormControls = require('./custom-form-controls')
Component({
  relations: {
    customFormControls: {
      type: 'descendant', // 关联的目标节点应为子孙节点
      target: customFormControls
    }
  }
})
```

relations 定义段

relations 定义段包含目标组件路径及其对应选项，可包含的选项见下表。

选项	类型	是否必填	描述
type	string	是	目标组件的相对关系，可选的值为 parent 、 child 、 ancestor 、 descendant
linked	function	否	关系生命周期函数，当关系被建立在页面节点树中时触发，触发时机在组件 attached 生命周期之后
linkChanged	function	否	关系生命周期函数，当关系在页面节点树中发生改变时触发，触发时机在组件 moved 生命周期之后
unlinked	function	否	关系生命周期函数，当关系脱离页面节点树时触发，触发时机在组件 detached 生命周期之后
target	string	否	如果这一项被设置，则它表示关联的目标节点所应具有的behavior，所有拥有这一 behavior 的组件节点都会被关联

抽象节点

最近更新时间：2024-03-21 15:51:52

在组件中使用抽象节点

在腾讯云小程序平台开发中，有时自定义组件模板中的一些节点，其对应的自定义组件不是由自定义组件本身确定的，而是自定义组件的调用者确定的。这时可以把这个节点声明为“抽象节点”。
例如，我们现在来实现一个“选框组”（selectable-group）组件，它其中可以放置单选框（custom-radio）或者复选框（custom-checkbox）。这个组件的 wxml 可以这样编写：

```
<!-- selectable-group.wxml -->
<view qq:for="{{labels}}">
  <label>
    <selectable disabled="{{false}}"></selectable>
    {{item}}
  </label>
</view>
```

其中，“selectable”不是任何在 json 文件的 `usingComponents` 字段中声明的组件，而是一个抽象节点。它需要在 `componentGenerics` 字段中声明：

```
{
  "componentGenerics": {
    "selectable": true
  }
}
```

使用包含抽象节点的组件

在使用 selectable-group 组件时，必须指定“selectable”具体是哪个组件：

```
<selectable-group generic:selectable="custom-radio" />
```

在使用 selectable-group 组件时，必须指定“selectable”具体是哪个组件：

```
<selectable-group generic:selectable="custom-checkbox" />
```

“selectable”节点则会生成“custom-checkbox”组件实例。

> **说明：**
> 上述的 `custom-radio` 和 `custom-checkbox` 需要包含在这个 wxml 对应 json 文件的 `usingComponents` 定义段中。

```
{
 "usingComponents": {
  "custom-radio": "path/to/custom/radio",
  "custom-checkbox": "path/to/custom/checkbox"
 }
}
```

抽象节点的默认组件

抽象节点可以指定一个默认组件，当具体组件未被指定时，将创建默认组件的实例。默认组件可以在 `componentGenerics` 字段中指定：

```
{
 "componentGenerics": {
  "selectable": {
   "default": "path/to/default/component"
  }
 }
}
```

> **说明：**
> 节点的 generic 引用 `generic:xxx="yyy"` 中，值 `yyy` 只能是静态值，不能包含数据绑定。因而抽象节点特性并不适用于动态决定节点名的场景。

自定义组件扩展

最近更新时间：2024-05-11 17:21:31

扩展后效果

```
// behavior.js
module.exports = Behavior({
  definitionFilter(defFields) {
    defFields.data.from = 'behavior'
  },
})

// component.js
Component({
  data: {
    from: 'component'
  },
  behaviors: [require('behavior.js')],
  ready() {
    console.log(this.data.from) // 此处会发现输出 behavior 而不是 component
  }
})
```

通过例子可以发现，自定义组件的扩展其实就是提供了修改自定义组件定义段的能力，上述示例就是修改了自定义组件中的 `data` 定义段里的内容。

使用扩展

`Behavior()` 构造器提供了新的定义段 `definitionFilter` ，用于支持自定义组件扩展。 `definitionFilter` 是一个函数，在被调用时会注入两个参数，第一个参数是使用该 behavior 的 component/behavior 的定义对象，第二个参数是该 behavior 所使用的 behavior 的 `definitionFilter` 函数列表。
以下举个例子来说明：

```
// behavior3.js
module.exports = Behavior({
  definitionFilter(defFields, definitionFilterArr) {},
})

// behavior2.js
module.exports = Behavior({
  behaviors: [require('behavior3.js')],
  definitionFilter(defFields, definitionFilterArr) {
```

```
    // definitionFilterArr[0](defFields)
  },
})

// behavior1.js
module.exports = Behavior({
  behaviors: [require('behavior2.js')],
  definitionFilter(defFields, definitionFilterArr) {},
})

// component.js
Component({
  behaviors: [require('behavior1.js')],
})
```

上述代码中声明了1个自定义组件和3个 behavior，每个 behavior 都使用了 definitionFilter 定义段。那么按照声明的顺序会有如下事情发生：

1. 当进行 behavior2 的声明时就会调用 behavior3 的 definitionFilter 函数，其中 defFields 参数是 behavior2 的定义段， definitionFilterArr 参数即为空数组，因为 behavior3 没有使用其他的 behavior。
2. 当进行 behavior1 的声明时就会调用 behavior2 的 definitionFilter 函数，其中 defFields 参数是 behavior1 的定义段， definitionFilterArr 参数是一个长度为1的数组， definitionFilterArr[0] 即为 behavior3 的 definitionFilter 函数，因为 behavior2 使用了 behavior3。用户在此处可以自行决定在进行 behavior1 的声明时要不要调用 behavior3 的 definitionFilter 函数，如果需要调用，在此处补充代码 definitionFilterArr[0](defFields) 即可， definitionFilterArr 参数会由基础库补充传入。

同理，在进行 component 的声明时就会调用 behavior1 的 definitionFilter 函数。
简单概括， definitionFilter 函数可以理解为当 A 使用了 B 时，A 声明就会调用 B 的 definitionFilter 函数并传入 A 的定义对象让 B 去过滤。此时如果 B 还使用了 C 和 D ，那么 B 可以自行决定要不要调用 C 和 D 的 definitionFilter 函数去过滤 A 的定义对象。

真实案例

下面利用扩展简单实现自定义组件的计算属性功能：

```
// behavior.js
module.exports = Behavior({
  lifetimes: {
    created() {
      this._originalSetData = this.setData // 原始 setData
      this.setData = this._setData // 封装后的 setData
    }
  },
  definitionFilter(defFields) {
```

```
    const computed = defFields.computed || {}
    const computedKeys = Object.keys(computed)
    const computedCache = {}

    // 计算 computed
    const calcComputed = (scope, insertToData) => {
      const needUpdate = {}
      const data = defFields.data = defFields.data || {}

      for (const key of computedKeys) {
        const value = computed[key].call(scope) // 计算新值
        if (computedCache[key] !== value) needUpdate[key] = computedCache[key] =
value
        if (insertToData) data[key] = needUpdate[key] // 直接插入到 data 中，初始化时才需
要的操作
      }

      return needUpdate
    }

    // 重写 setData 方法
    defFields.methods = defFields.methods || {}
    defFields.methods._setData = function (data, callback) {
      const originalSetData = this._originalSetData // 原始 setData
      originalSetData.call(this, data, callback) // 做 data 的 setData
      const needUpdate = calcComputed(this) // 计算 computed
      originalSetData.call(this, needUpdate) // 做 computed 的 setData
    }

    // 初始化 computed
    calcComputed(defFields, true) // 计算 computed
  }
})
```

在组件中使用：

```
const beh = require('./behavior.js')
Component({
  behaviors: [beh],
  data: {
    a: 0,
  },
  computed: {
    b() {
      return this.data.a + 100
```

```
    },
  },
  methods: {
    onTap() {
      this.setData({
        a: ++this.data.a,
      })
    }
  }
})
```

```
<view>data: {{a}}</view>
<view>computed: {{b}}</view>
<button bindtap="onTap">click</button>
```

实现原理很简单，对已有的 setData 进行二次封装，在每次 setData 的时候计算出 computed 里各字段的值，然后设到 data 中，已达到计算属性的效果。

> **说明：**
> 此实现示例仅作为一个简单案例来展示，请勿直接在生产环境中使用。

官方扩展包

详情请参考 computed 。

数据监听器

最近更新时间：2024-03-21 15:51:52

定义和使用数据监听器

数据监听器可以用于监听和响应任何属性和数据字段的变化。有时，在一些数据字段被 setData 设置时，需要执行一些操作。例如，this.data.sum 是 this.data.numberA 与 this.data.numberB 的和。此时，可以使用数据监听器实现。

```
Component({
  attached: function() {
    this.setData({
      numberA: 1,
      numberB: 2,
    })
  },
  observers: {
    'numberA, numberB': function(numberA, numberB) {
      // 在 numberA 或者 numberB 被设置时，执行这个函数
      this.setData({
        sum: numberA + numberB
      })
    }
  }
})
```

监听字段语法

数据监听器支持监听属性或内部数据的变化，可以同时监听多个。一次 setData 最多触发每个监听器一次。同时，监听器可以监听子数据字段，如下例所示。

```
Component({
  observers: {
    'some.subfield': function(subfield) {
      // 使用 setData 设置 this.data.some.subfield 时触发
      // （除此以外，使用 setData 设置 this.data.some 也会触发）
      subfield === this.data.some.subfield
    },
    'arr[12]': function(arr12) {
      // 使用 setData 设置 this.data.arr[12] 时触发
      // （除此以外，使用 setData 设置 this.data.arr 也会触发）
      arr12 === this.data.arr[12]
```

```
    },
  }
})
```

如果需要监听所有子数据字段的变化，可以使用通配符 **。

```
Component({
  observers: {
    'some.field.**': function(field) {
      // 使用 setData 设置 this.data.some.field 本身或其下任何子数据字段时触发
      // （除此以外，使用 setData 设置 this.data.some 也会触发）
      field === this.data.some.field
    },
  },
  attached: function() {
    // 这样会触发上面的 observer
    this.setData({
      'some.field': { /* ... */ }
    })
    // 这样也会触发上面的 observer
    this.setData({
      'some.field.xxx': { /* ... */ }
    })
    // 这样还是会触发上面的 observer
    this.setData({
      'some': { /* ... */ }
    })
  }
})
```

此外，如果仅使用通配符 ** ，则可以监听全部的 setData 。

```
Component({
  observers: {
    '**': function() {
      // 每次 setData 都触发
    },
  },
}
```

说明：

- 数据监听器监听的是 setData 涉及到的数据字段，即使这些数据字段的值没有发生变化，数据监听器

依然会被触发。

- 如果在数据监听器函数中使用 setData 设置本身监听的数据字段，可能会导致死循环，需要特别留意。
- 数据监听器和属性的 observer 相比，数据监听器更强大且通常具有更好的性能。

基础能力

最近更新时间：2024-03-28 15:39:01

网络

> **说明：**
> 在小程序中使用网络相关的 API 时，需要注意下列问题，请开发者提前了解。

服务器域名配置

每个小程序需要事先设置一个通讯域名，小程序只可以跟指定的域名与进行网络通信。包括普通 HTTPS 请求（ wx.request ）、上传文件（ wx.uploadFile ）。

配置流程

服务器域名请在小程序后台 > 设置 > 开发设置 > 服务器域名中进行配置，配置时需要注意：

- 域名只支持 https （ wx.request ）、（ wx.uploadFile ）协议。
- 域名不能使用 IP 地址或 localhost。
- 可以配置端口，如 https://miniApp.com:8080，但是配置后只能向 https://miniApp.com:8080 发起请求。如果向 https://miniApp.com、https://miniApp.com:9091 等 URL 请求则会失败。
- 如果不配置端口。如 https://miniApp.com，那么请求的 URL 中也不能包含端口，甚至是默认的 443 端口也不可以。如果向 https://miniApp.com:443 请求则会失败。

网络请求

超时时间

默认超时时间和最大超时时间都是60s。
超时时间可以在 app.json 中配置。

使用限制

- 网络请求的 referer header 不可设置。其格式固定为 https://appservice.wx.com/{appid}/{version}/page-frame.html，其中 {appid} 为小程序的 appid，{version} 为小程序的版本号，版本号为0表示为开发版、体验版以及审核版本，版本号为 devtools 表示为开发者工具，其余为正式版本。
- wx.request、wx.uploadFile 的最大并发限制是 10 个。
- 小程序进入后台运行后，如果5s内网络请求没有结束，会回调错误信息 fail interrupted；在回到前台之前，网络请求接口调用都会无法调用。

返回值编码

- 建议服务器返回值使用 UTF-8 编码。对于非 UTF-8 编码，小程序会尝试进行转换，但是会有转换失败的可能。
- 小程序会自动对 BOM 头进行过滤（只过滤一个 BOM 头）。

回调函数

只要成功接收到服务器返回，无论 `statusCode` 是多少，都会进入 `success` 回调。请开发者根据业务逻辑对返回值进行判断。

常见问题

HTTPS 证书

小程序必须使用 HTTPS/WSS 发起网络请求。请求时系统会对服务器域名使用的 HTTPS 证书进行校验，如果校验失败，则请求不能成功发起。由于系统限制，不同平台对于证书要求的严格程度不同。为了保证小程序的兼容性，建议开发者按照最高标准进行证书配置，并使用相关工具检查现有证书是否符合要求。
对证书要求如下：

- HTTPS 证书必须有效；
  - 证书必须被系统信任，即根证书被已系统内置。
  - 部署 SSL 证书的网站域名必须与证书颁发的域名一致。
  - 证书必须在有效期内。
  - 证书的信任链必需完整（需要服务器配置）。
- `iOS` 不支持自签名证书。
- `iOS` 下证书必须满足苹果 App Transport Security (ATS) 的要求。
- TLS 必须支持 1.2 及以上版本。部分旧 `Android` 机型还未支持 TLS 1.2，请确保 HTTPS 服务器的 TLS 版本支持 1.2 及以下版本。
- 部分 CA 可能不被操作系统信任，请开发者在选择证书时注意小程序和各系统的相关通告。

> **说明：**
> - 证书有效性可以使用 `openssl s_client -connect example.com:443` 命令验证，也可以使用其他在线工具。
> - 除了网络请求 API 外，小程序中其他 HTTPS 请求如果出现异常，也请按上述流程进行检查。如 https 的图片无法加载、音视频无法播放等。

跳过域名校验

在开发者工具中，可以临时开启 `开发环境不校验请求域名` 、 `TLS版本及 HTTPS 证书` 选项，跳过服务器域名的校验。此时，在开发者工具中及手机开启调试模式时，不会进行服务器域名的校验。
在服务器域名配置成功后，建议开发者关闭此选项进行开发，并在各平台下进行测试，以确认服务器域名配置正确。

说明：

- 如果手机上出现 “打开调试模式可以发出请求，关闭调试模式无法发出请求” 的现象，请确认是否跳过了域名校验，并确认服务器域名和证书配置是否正确。
- 如遇到“服务器内部错误”，请尝试以下办法自查 ①更换浏览器 ②刷新网页 ③排查文件是否放在正确的服务器根目录下。 排查办法：按照以下规则拼接网址，并且自行尝试访问是否成功：https://m.da9c.cn/文件名。

存储

每个小程序都可以有自己的本地缓存，可以通过 wx.setStorage/qq.setStorageSync、wx.getStorage/wx.getStorageSync、qq.clearStorage/wx.clearStorageSync，wx.removeStorage/wx.removeStorageSync 对本地缓存进行读写和清理。
同一个用户，同一个小程序 storage 上限为 10MB。storage 以用户维度隔离，同一台设备上，A 用户无法读取到 B 用户的数据。

说明：
如果用户储存空间不足，应用会清空最近最久未使用的小程序的本地缓存，因此并不建议将关键信息全部存在 storage，以防储存空间不足或用户换设备的情况。

Canvas画布

所有在 <canvas> 中的画图必须用 JavaScript 完成：
WXML：在接下来的示例中如无特殊声明都会用这个 WXML 为模板，不再重复。

```
<canvas canvas-id="myCanvas" style="border: 1px solid;" />
```

JS：我们在接下来的例子中会将 JS 放在 onLoad 中

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 75)
ctx.draw()
```

第一步：创建一个 Canvas 绘图上下文
首先，我们需要创建一个 Canvas 绘图上下文 CanvasContext。CanvasContext 是小程序内建的一个对象，有一些绘图的方法：

```
const ctx = wx.createCanvasContext('myCanvas')
```

第二步：使用 Canvas 绘图上下文进行绘图描述

接着，我们来描述要在 Canvas 中绘制什么内容。设置绘图上下文的填充色为红色：

```
ctx.setFillStyle('red')
```

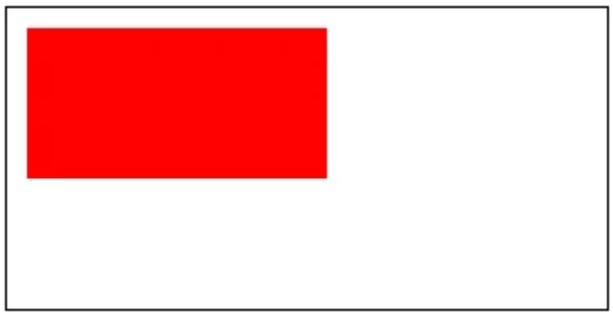
用 `fillRect(x, y, width, height)` 方法画一个矩形，填充为刚刚设置的红色：

```
ctx.fillRect(10, 10, 150, 75)
```

第三步：画图

告诉 <canvas> 组件，您要将刚刚的描述绘制上去：

```
ctx.draw()
```



坐标系

可以在 <canvas> 中加上一些事件，来观测它的坐标系：

```
<canvas
  canvas-id="myCanvas"
  style="margin: 5px; border:1px solid #d3d3d3;"
  bindtouchstart="start"
  bindtouchmove="move"
  bindtouchend="end"
/>

<view hidden="{{hidden}}">
  Coordinates: ({{x}}, {{y}})
</view>
```

```
Page({
  data: {
```

```
    x: 0,
    y: 0,
    hidden: true
  },
  start(e) {
    this.setData({
      hidden: false,
      x: e.touches[0].x,
      y: e.touches[0].y
    })
  },
  move(e) {
    this.setData({
      x: e.touches[0].x,
      y: e.touches[0].y
    })
  },
  end(e) {
    this.setData({
      hidden: true
    })
  }
})
```

当您把手指放到 canvas 中，就会在下边显示出触碰点的坐标：

渐变

渐变能用于填充一个矩形，圆，线，文字等。填充色可以不固定为固定的一种颜色。
我们提供了两种颜色渐变的方式：

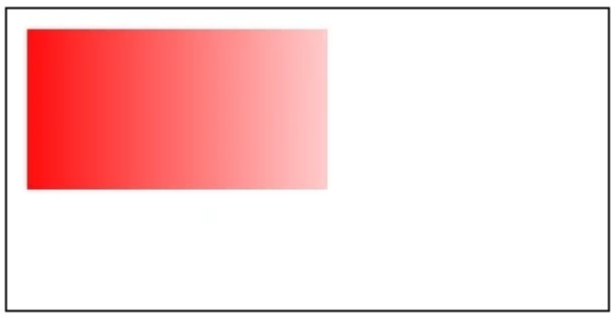
- `createLinearGradient(x, y, x1, y1)` 创建一个线性的渐变。
- `createCircularGradient(x, y, r)` 创建一个从圆心开始的渐变。

创建了一个渐变对象，就必须添加两个颜色渐变点。
`addColorStop(position, color)` 方法用于指定颜色渐变点的位置和颜色，位置必须位于0到1之间。

可以用 setFillStyle 和 setStrokeStyle 方法设置渐变，然后进行画图描述。

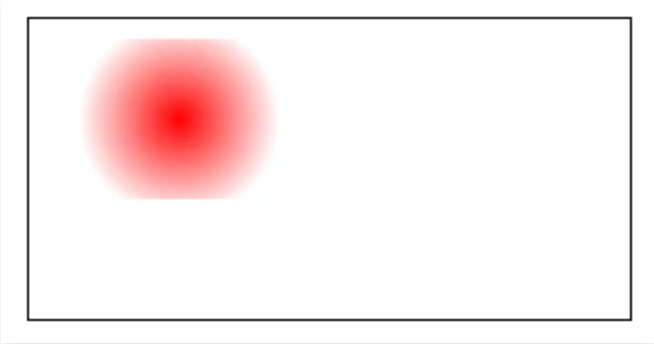
- **使用 createLinearGradient()**

```
const ctx = wx.createCanvasContext('myCanvas')

// Create linear gradient
const grd = ctx.createLinearGradient(0, 0, 200, 0)
grd.addColorStop(0, 'red')
grd.addColorStop(1, 'white')

// Fill with gradient
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```



- **使用** createCircularGradient()

```
const ctx = wx.createCanvasContext('myCanvas')

// Create circular gradient
const grd = ctx.createCircularGradient(75, 50, 50)
grd.addColorStop(0, 'red')
grd.addColorStop(1, 'white')

// Fill with gradient
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```



自定义 tabBar

自定义 tabBar 可以让开发者更加灵活地设置 tabBar 样式，以满足更多个性化的场景。
在自定义 tabBar 模式下

- 为了保证低版本兼容以及区分哪些页面是 tab 页，tabBar 的相关配置项需完整声明，但这些字段不会作用于自定义 tabBar 的渲染。
- 此时需要开发者提供一个自定义组件来渲染 tabBar，所有 tabBar 的样式都由该自定义组件渲染。推荐用 fixed 在底部的 <cover-view> + <cover-image> 组件渲染样式，以保证 tabBar 层级相对较高。
- 与 tabBar 样式相关的接口，如 wx.setTabBarItem 等将失效。
- 每个 tab 页下的自定义 tabBar 组件实例是不同的，可通过自定义组件下的 getTabBar 接口，获取当前页面的自定义 tabBar 组件实例。

> **说明：**
> 如需实现 tab 选中态，要在当前页面下，通过 getTabBar 接口获取组件实例，并调用 setData 更新选中态。可参考下文 配置信息 中的示例代码。

配置信息

在 `app.json` 中的 `tabBar` 项指定 `custom` 字段，同时其余 `tabBar` 相关配置也补充完整。
所有 tab 页的 json 里需声明 `usingComponents` 项，也可以在 `app.json` 全局开启。
示例代码：

```
{
 "tabBar": {
   "custom": true,
   "color": "#000000",
   "selectedColor": "#000000",
   "backgroundColor": "#000000",
   "list": [
     {
       "pagePath":
"page/develop/miniprogram/develop/miniprogram/component/index",
       "text": "组件"
```

```
    },
    {
      "pagePath": "page/develop/miniprogram/API/index",
      "text": "接口"
    }
  ]
 },
 "usingComponents": {}
}
```

添加 tabBar 代码

在代码根目录下添加入口文件：

```
custom-tab-bar/index.js
custom-tab-bar/index.json
custom-tab-bar/index.wxml
custom-tab-bar/index.wxss
```

编写 tabBar 代码

用自定义组件的方式编写即可，该自定义组件完全接管 tabBar 的渲染。另外，自定义组件新增 `getTabBar` 接口，可获取当前页面下的自定义 tabBar 组件实例。
示例代码：

```
// page/index/index.js

Page({
 onShow:function(){

   // 取当前页面的TabBar实例，设置选中态
   if (typeof this.getTabBar === 'function' && this.getTabBar()) {
     this.getTabBar().setData({
       selected: 0
     })
   }
 }
})
```

分包加载

最近更新时间：2023-12-06 15:14:26

某些情况下，开发者需要将小程序划分成不同的子包，在构建时打包成不同的分包，用户在使用时按需进行加载。
在构建小程序分包项目时，构建会输出一个或多个分包。每个使用分包小程序必定含有一个主包。所谓的主包，即放置默认启动页面/TabBar 页面，以及一些所有分包都需用到公共资源/JS 脚本；而分包则是根据开发者的配置进行划分。
在小程序启动时，默认会下载主包并启动主包内页面，当用户用户进入分包内某个页面时，客户端会把对应分包下载下来，下载完成后再进行展示。
目前小程序分包大小有以下限制：

- 整个小程序所有分包大小不超过 24M；
- 单个分包/主包大小不能超过 2M。

对小程序进行分包，可以优化小程序首次启动的下载时间，以及在多团队共同开发时可以更好的解耦协作。可参见如下三种分包方式：

方法一：使用分包

配置方法

假设支持分包的小程序目录结构如下：

```
├── app.js
├── app.json
├── app.wxss
├── packageA
│   └── pages
│       ├── cat
│       └── dog
├── packageB
│   └── pages
│       ├── apple
│       └── banana
├── pages
│   ├── index
│   └── logs
└── utils
```

开发者通过在 app.json subpackages 字段声明项目分包结构（写成 subPackages 也同样支持）：

```
{
  "pages": ["pages/index", "pages/logs"],
```

```
  "subpackages": [
    {
      "root": "packageA",
      "pages": ["pages/cat", "pages/dog"]
    },
    {
      "root": "packageB",
      "name": "pack2",
      "pages": ["pages/apple", "pages/banana"]
    }
  ]
}
```

`subpackages` 中，每个分包的配置有以下几项：

字段	类型	说明
root	String	分包根目录
name	String Array	分包别名，分包预下载 时使用，如将分包配置到 preloadRule 中，则必须指定
pages	String Array	分包页面路径，相对于分包根目录
independent	Boolean	分包是否是 独立分包

打包原则

- 声明 `subpackages` 后，将按 `subpackages` 配置路径进行打包， `subpackages` 配置路径外的目录将被打包到 App（主包）中；
- App（主包）也可以有自己的 pages（即最外层的 pages 字段）；
- `subpackage` 的根目录不能是另外一个 `subpackage` 内的子目录；
- `tabBar` 页面必须在 app（主包）内。

引用原则

- `packageA` 无法 require `packageB` JS 文件，但可以 require app、自己 package 内的 JS 文件；
- `packageA` 无法 import `packageB` 的 template，但可以 require app、自己 package 内的 template；
- `packageA` 无法使用 `packageB` 的资源，但可以使用 App、自己 package 内的资源。

低版本兼容

由 QQ 后台编译来处理旧版本客户端的兼容，后台会编译两份代码包，一份是分包后代码，另外一份是整包的兼容代码。 新客户端用分包，老客户端还是用的整包，完整包会把各个 `subpackage` 里面的路径放到 pages 中。

方法二：独立分包

独立分包是小程序中一种特殊类型的分包，可以独立于主包和其他分包运行。从独立分包中页面进入小程序时，不需要下载主包。当用户进入普通分包或主包内页面时，主包才会被下载。
开发者可以按需将某些具有一定功能独立性的页面配置到独立分包中。当小程序从普通的分包页面启动时，需要首先下载主包；而独立分包不依赖主包即可运行，可以很大程度上提升分包页面的启动速度。
一个小程序中可以有多个独立分包。

限制

独立分包属于分包的一种。普通分包的所有限制都对独立分包有效。独立分包中插件、自定义组件的处理方式同普通分包。
此外，使用独立分包时要注意：

- **独立分包中不能依赖主包和其他分包中的内容**，包括 js 文件、template、wxss、自定义组件、插件等。主包中的 app.wxss 对独立分包无效，应避免在独立分包页面中使用 app.wxss 中的样式；
- App 只能在主包内定义，独立分包中不能定义 App，会造成无法预期的行为；
- 独立分包中暂时不支持使用插件。

配置方法

假设小程序目录结构如下：

```
├── app.js
├── app.json
├── app.wxss
├── moduleA
│   └── pages
│       ├── rabbit
│       └── squirrel
├── moduleB
│   └── pages
│       ├── pear
│       └── pineapple
├── pages
│   ├── index
│   └── logs
└── utils
```

开发者通过在 `app.json` 的 s `ubpackages` 字段中对应的分包配置项中定义 `independent` 字段声明对应分包为独立分包。

```
{
 "pages": ["pages/index", "pages/logs"],
 "subpackages": [
  {
   "root": "moduleA",
   "pages": ["pages/rabbit", "pages/squirrel"]
  },
  {
   "root": "moduleB",
   "pages": ["pages/pear", "pages/pineapple"],
   "independent": true
  }
 ]
}
```

注意事项

- **关于 getApp()**

  与普通分包不同，独立分包运行时，`App` 并不一定被注册，因此 getApp() 也不一定可以获得 App 对象：
  - 当用户从独立分包页面启动小程序时，主包不存在，App 也不存在，此时调用 getApp() 获取到的是 `undefined` 。当用户进入普通分包或主包内页面时，主包才会被下载，App 才会被注册。
  - 当用户是从普通分包或主包内页面跳转到独立分包页面时，主包已经存在，此时调用 getApp() 可以获取到真正的 App。

  由于这一限制，开发者无法通过 App 对象实现独立分包和小程序其他部分的全局变量共享。
  getApp 支持 allowDefault 参数，在 App 未定义时返回一个默认实现。当主包加载，App 被注册时，默认实现中定义的属性会被覆盖合并到真正的 App 中。

  ### 示例代码

  独立分包：

  ```
  const app = getApp({allowDefault: true}) // {}
  app.data = 456
  app.global = {}
  ```

  app.js 中：

  ```
  App({
    data: 123,
    other: 'hello'
  })
  ```

```
console.log(getApp()) // {global: {}, data: 456, other: 'hello'}
```

- **关于 App 生命周期**

  当从独立分包启动小程序时，主包中 App 的 `onLaunch` 和首次 `onShow` 会在从独立分包页面首次进入主包或其他普通分包页面时调用。
  由于独立分包中无法定义 App，小程序生命周期的监听可以使用 `wx.onAppShow`，`wx.onAppHide` 完成。App 上的其他事件可以使用 wx.onError，wx.onPageNotFound 监听。

低版本兼容

> ⚠ **注意：**
> 在兼容模式下，主包中的 app.wxss 可能会对独立分包中的页面产生影响，因此应避免在独立分包页面中使用 app.wxss 中的样式。

方法三：分包预下载

限制

同一个分包中的页面享有共同的预下载大小限额2M，限额会在工具中打包时校验。
如，页面 A 和 B 都在同一个分包中，A 中预下载总大小0.5M的分包，B 中最多只能预下载总大小1.5M的分包。

配置方法

预下载分包行为在进入某个页面时触发，通过在 `app.json` 增加 `preloadRule` 配置来控制。

```
{
 "pages": ["pages/index"],
 "subpackages": [
  {
   "root": "important",
   "pages": ["index"]
  },
  {
   "root": "sub1",
   "pages": ["index"]
  },
  {
   "name": "hello",
   "root": "path/to",
   "pages": ["index"]
  },
  {
   "root": "sub3",
```

```
      "pages": ["index"]
    },
    {
      "root": "indep",
      "pages": ["index"],
      "independent": true
    }
  ],
  "preloadRule": {
    "pages/index": {
      "network": "all",
      "packages": ["important"]
    },
    "sub1/index": {
      "packages": ["hello", "sub3"]
    },
    "sub3/index": {
      "packages": ["path/to"]
    },
    "indep/index": {
      "packages": ["__APP__"]
    }
  }
}
```

preloadRule 中，key 是页面路径，value 是进入此页面的预下载配置，每个配置有以下几项：

字段	类型	必填	默认值	说明
packages	StringArray	是	无	进入页面后预下载分包的 root 或 name。__APP__ 表示主包。
network	String	否	Wi-Fi	在指定网络下预下载，可选值为： • all：不限网络 • Wi-Fi：仅 Wi-Fi 下预下载

可用性

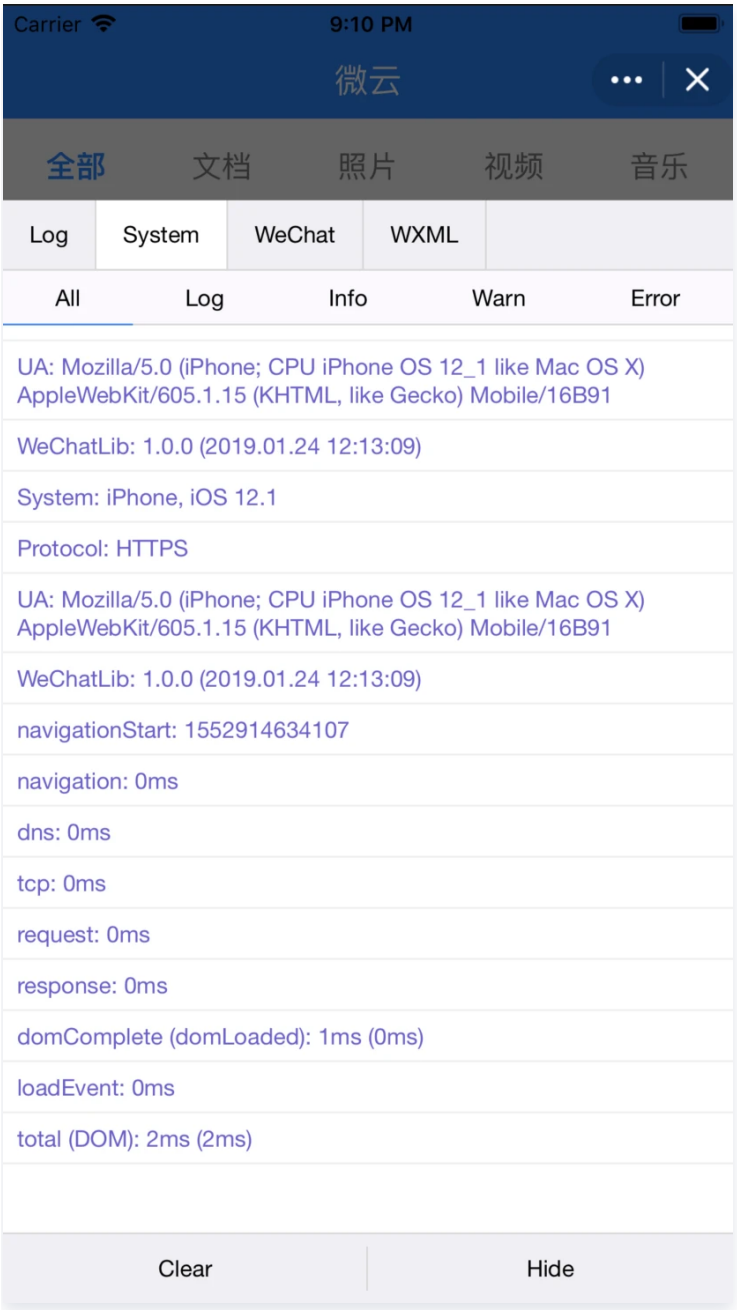
最近更新时间：2024-03-21 15:51:52

调试

vConsole

在真机上，如果想要查看 console API 输出的日志内容和额外的调试信息，单击屏幕右上角的按钮打开的菜单里选择**打开调试**。此时小程序会退出，重新打开后会右下角会出现一个 `vConsole` 按钮。单击 `vConsole` 按钮可以打开日志面板。

小程序的 vConsole 展示内容如下：



vConsole 使用说明

由于实现机制的限制，开发者调用 console API 打印的日志内容，是转换成 `JSON` 字符串后传输给 `vConsole` 的，导致 `vConsole` 中展示的内容会有一些限制：

- 除了 `Number` 、 `String` 、 `Boolean` 、 `null` 外，其他类型都会被作为 `Object` 处理展示，打印对象及原型链中的 Enumerable 属性。
- `Infinity` 和 `NaN` 会显示为 `null` 。
- `undefined` 、 `ArrayBuffer` 、 `Function` 类型无法显示。
- 无法打印存在循环引用的对象。

```
const a = {}
a.b = a
console.log(a)
```

针对上述问题，小程序在使用 vConsole 时做了一些处理。

```
const circular = {x: {}, c: {}}
circular.x = [{promise: Promise.resolve()}]
circular.a = circular
circular.c.x0 = circular.x[0]

console.log(circular)
// "{a: '<Circular: @>', c: {x0: '<Circular: @.x[0]>'}, x: [{promise: '<Promise>'}]}"
```

> **说明：**
> 尽量避免在非调试情景下打印结构过于复杂或内容过长的日志内容，可能会带来额外耗时。

运行环境

小程序运行环境

小程序运行在三端：iOS（iPhone/iPad）、Android 和用于调试的开发者工具。
三端的脚本执行环境以及用于渲染非原生组件的环境是各不相同：

- 在 iOS 上，小程序逻辑层的 javascript 代码运行在 JavaScriptCore 中，视图层是由 WKWebView 来渲染的，环境有 iOS8、iOS9、iOS10。
- 在 Android 上，小程序逻辑层的 javascript 代码运行中 X5 JSCore 中，视图层基于 Mobile Chrome 97 内核实现了 同层渲染 的功能。
- 在 开发工具上，小程序逻辑层的 javascript 代码是运行在 NW.js 中，视图层是由 Chromium 60 Webview 来渲染的。

运行限制

基于安全考虑，小程序中不支持动态执行 JS 代码，即：

- 不支持使用 `eval` 执行 `JS` 代码。
- 不支持使用 `new Function` 创建函数。

平台差异

尽管三端的环境是十分相似的，但是还是有些许区别：

- `JavaScript` 语法和 API 支持不一致：语法上开发者可以通过开启 `ES6` 转 `ES5` 的功能来规避；此外，小程序基础库内置了必要的 Polyfill，来弥补 API 的差异。
- `WXSS` 渲染表现不一致：尽管可以通过开启样式补全来规避大部分的问题，还是建议开发者需要在 iOS 和 Android 上分别检查小程序的真实表现。

> **说明：**
> 开发者工具仅供调试使用，最终的表现以客户端为准。

客户端 ES6 API 支持情况

> **说明：**
> 以下表格中 √ 为支持，× 为该项不支持。

String	iOS 8	iOS9	iOS10	Android
codePointAt	√	√	√	√
normalize	×	×	√	√
includes	√	√	√	√
startsWith	√	√	√	√
endsWith	√	√	√	√
repeat	√	√	√	√
String.fromCodePoint	√	√	√	√

Array	iOS 8	iOS9	iOS10	Android
copyWithin	√	√	√	√
find	√	√	√	√

findIndex	√	√	√	√
fill	√	√	√	√
entries	√	√	√	√
keys	√	√	√	√
values	×	√	√	×
includes	×	√	√	√
Array.from	√	√	√	√
Array.of	√	√	√	√

Number	iOS 8	iOS9	iOS10	Android
isFinite	√	√	√	√
isNaN	√	√	√	√
parseInt	√	√	√	√
parseFloat	√	√	√	√
isInteger	√	√	√	√
EPSILON	√	√	√	√
isSafeInteger	√	√	√	√

Math	iOS 8	iOS9	iOS10	Android
trunc	√	√	√	√
sign	√	√	√	√
cbrt	√	√	√	√
clz32	√	√	√	√
imul	√	√	√	√
fround	√	√	√	√
hypot	√	√	√	√
expm1	√	√	√	√

log1p	√	√	√	√
log10	√	√	√	√
log2	√	√	√	√
sinh	√	√	√	√
cosh	√	√	√	√
tanh	√	√	√	√
asinh	√	√	√	√
acosh	√	√	√	√
atanh	√	√	√	√

Object	iOS 8	iOS9	iOS10	Android
is	√	√	√	√
assign	√	√	√	√
getOwnPropertyDescriptor	√	√	√	√
keys	√	√	√	√
getOwnPropertyNames	√	√	√	√
getOwnPropertySymbols	√	√	√	√

Other	iOS 8	iOS9	iOS10	Android
Symbol	√	√	√	√
Set	√	√	√	√
Map	√	√	√	√
Proxy	×	×	√	×
Reflect	√	√	√	√
Promise	√	√	√	√

运行机制

小程序启动会有两种情况，一种是**冷启动**，一种是**热启动**。 假如用户已经打开过某小程序，然后在一定时间内再次打开该小程序，此时无需重新启动，只需将后台态的小程序切换到前台，这个过程就是热启动；冷启动指的是用户首次打开或小程序被宿主 App 主动销毁后再次打开的情况，此时小程序需要重新加载启动。

更新机制

小程序冷启动时如果发现有新版本，将会异步下载新版本的代码包，并同时用客户端本地的包进行启动，即新版本的小程序需要等下一次冷启动才会应用上。 如果需要马上应用最新版本，可以使用 wx.getUpdateManager API 进行处理。

运行机制

- 小程序没有重启的概念。
- 当小程序进入后台，客户端会维持一段时间的运行状态，超过一定时间后（目前是5分钟）会被容器主动销毁。
- 在 iOS 上，当客户端在一定时间间隔内（目前是 5 秒）连续收到两次及以上系统内存告警时，会主动进行小程序的销毁，并提示用户**该小程序可能导致宿主响应变慢，被终止**。建议做必要的内存清理。

API

API 概览

最近更新时间：2024-05-24 15:58:42

1、基础

1.1、系统能力

名称	功能说明	最低版本
canIUse	判断小程序的 API，回调，参数，组件等是否在当前版本可用	1.5.0
env	环境变量	1.5.1
base64ToArrayBuffer	将 Base64 字符串转成 ArrayBuffer 对象	1.5.1
arrayBufferToBase64	将 ArrayBuffer 对象转成 Base64 字符串	1.5.1

1.2、系统

名称	功能说明	最低版本
getSystemInfo	异步调用获取系统信息	1.5.0
getSystemInfoSync	同步调用获取系统信息	1.5.1
getSystemInfoAsync	异步获取系统信息	1.5.1
getWindowInfo	获取窗口信息	1.5.1
getSystemSetting	获取设备设置	1.5.0
openSystemBluetoothSetting	跳转系统蓝牙设置页	1.5.1
openAppAuthorizeSetting	跳转系统宿主客户端授权管理页	1.5.1
getRendererUserAgent	获取 Webview 小程序的 UserAgent	1.5.1

getDeviceInfo	获取设备基础信息	1.5.1
getAppBaseInfo	获取宿主客户端基础信息	1.5.1
getAppAuthorizeSetting	获取宿主客户端授权设置	1.5.1

1.3、更新

名称	功能说明	最低版本
getUpdateManager	获取全局唯一的版本更新管理器，用于管理小程序更新	1.5.1
UpdateManager.applyUpdate	强制小程序重启并使用新版本	1.5.0
UpdateManager.onCheckForUpdate	监听向微信后台请求检查更新结果事件	1.5.0
UpdateManager.onUpdateFailed	监听小程序更新失败事件	1.5.0
UpdateManager.onUpdateReady	监听小程序有版本更新事件	1.5.0

1.4、小程序

名称	功能说明	最低版本
getLaunchOptionsSync	获取小程序启动时的参数	1.5.0
getEnterOptionsSync	获取本次小程序启动时的参数	1.5.1
onError	监听小程序错误事件	1.5.0
offError	移除小程序错误事件的监听函数	1.5.0
onThemeChange	监听系统主题改变事件	1.5.0
offThemeChange	移除系统主题改变事件的监听函数	1.5.0
onPageNotFound	监听小程序要打开的页面不存在事件	1.5.1
offPageNotFound	移除小程序要打开的页面不存在事件的监听函数	1.5.1
onAppShow	监听小程序切前台事件	1.5.1

offAppShow	移除小程序切前台事件的监听函数	1.5.1
onAppHide	监听小程序切后台事件	1.5.1
offAppHide	移除小程序切前台事件的监听函数	1.5.1
onUnhandledRejection	监听未处理的 Promise 拒绝事件	1.5.1
onUnhandledRejection	移除未处理的 Promise 拒绝事件的监听函数	1.5.1

1.5、调试

名称	功能说明	最低版本
setEnableDebug	设置是否打开调试开关	1.5.0
getLogManager	获取日志管理器对象	1.5.1
console.debug	向调试面板中打印 debug 日志	1.5.0
console.error	向调试面板中打印 error 日志	1.5.0
console.group	在调试面板中创建一个新的分组	1.5.0
console.groupEnd	结束由 console.group 创建的分组	1.5.0
console.info	向调试面板中打印 info 日志	1.5.0
console.log	向调试面板中打印 log 日志	1.5.0
console.warn	向调试面板中打印 warn 日志	1.5.0
LogManager.debug	写 debug 日志	1.5.0
LogManager.info	写 info 日志	1.5.0
LogManager.log	写 log 日志	1.5.0
LogManager.warn	写 warn 日志	1.5.0
RealtimeLogManager.addFilterMsg	添加过滤关键字，暂不支持在插件使用	1.5.16
RealtimeLogManager.error	写 error 日志，暂不支持在插件使用	1.5.16
RealtimeLogManager.getCurrentState	可以获取当前缓存剩余空间	1.5.16

RealtimeLogManager.in	设置实时日志page参数所在的页面，暂不支持在插件使用	1.5.16
RealtimeLogManager.info	写 info 日志，暂不支持在插件使用	1.5.16
RealtimeLogManager.setFilterMsg	设置过滤关键字，暂不支持在插件使用	1.5.16
RealtimeLogManager.tag	获取给定标签的日志管理器实例，目前只支持在插件使用	1.5.16
RealtimeLogManager.warn	写 warn 日志，暂不支持在插件使用	1.5.16
RealtimeTagLogManageraddFilterMsg	添加过滤关键字	1.5.16
RealtimeTagLogManager.error	写 error日志	1.5.16
RealtimeTagLogManager.info	写 info 日志	1.5.16
RealtimeTagLogManager.setFilterMsg	设置过滤关键字	1.5.16
RealtimeTagLogManager.warn	写 warn 日志	1.5.16

2、路由

名称	功能说明	最低版本
switchTab	跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面	1.5.0
reLaunch	关闭所有页面，打开到应用内的某个页面	1.5.0
redirectTo	关闭当前页面，跳转到应用内的某个页面	1.5.0
navigateTo	保留当前页面，跳转到应用内的某个页面	1.5.1
navigateBack	关闭当前页面，返回上一页面或多级页面	1.5.0
EventChannel	页面间事件通信通道	1.5.1

3、跳转

名称	功能说明	最低版本
exitMiniProgram	退出当前小程序	1.5.0
navigateToMiniProgram	打开另一个小程序	1.5.0
navigateBackMiniProgram	返回到上一个小程序。只有在当前小程序是被其他小程序打开时可以调用成功	1.5.1

4、转发

名称	功能说明	最低版本
hideShareMenu	隐藏当前页面的转发按钮	1.5.0
showShareMenu	显示当前页面的转发按钮	1.5.0
updateShareMenu	更新转发属性	1.5.0
showShareImageMenu	打开分享图片弹窗	1.5.35
onCopyUrl	监听用户点击右上角菜单的「复制链接」按钮时触发的事件	1.5.35
offCopyUrl	移除用户点击右上角菜单的「复制链接」按钮时触发的事件的全部监听函数	1.5.35

5、界面

5.1、交互

名称	功能说明	最低版本
hideLoading	隐藏 loading 提示框	1.5.0
hideToast	隐藏消息提示框	1.5.0
showActionSheet	显示操作菜单	1.5.0
showLoading	显示 loading 提示框	1.5.0

showModal	显示模态对话框	1.5.0
showToast	显示消息提示框	1.5.0
enableAlertBeforeUnload	开启小程序页面返回询问对话框	1.5.1
disableAlertBeforeUnload	关闭小程序页面返回询问对话框	1.5.1

5.2、导航栏

名称	功能说明	最低版本
setNavigationBarTitle	动态设置当前页面的标题	1.5.0
setNavigationBarColor	设置页面导航条颜色	1.5.0
hideHomeButton	隐藏返回首页按钮	1.5.0
showNavigationBarLoading	在当前页面显示导航条加载动画	1.5.1
hideNavigationBarLoading	在当前页面隐藏导航条加载动画	1.5.1

5.3、背景

名称	功能说明	最低版本
setBackgroundColor	动态设置窗口的背景色	1.5.0
setBackgroundTextStyle	动态设置下拉背景字体、loading 图的样式	1.5.0

5.4、TabBar

名称	功能说明	最低版本
showTabBar	显示 tabBar	1.5.0
hideTabBar	隐藏 tabBar	1.5.0
setTabBarStyle	动态设置 tabBar 的整体样式	1.5.0
setTabBarItem	动态设置 tabBar 某一项的内容，2.7.0 起图片支持临时文件和网络文件	1.5.0
showTabBarRedDot	显示 tabBar 某一项的右上角的红点	1.5.0

hideTabBarRedDot	隐藏 tabBar 某一项的右上角的红点	1.5.0
setTabBarBadge	为 tabBar 某一项的右上角添加文本	1.5.0
removeTabBarBadge	移除 tabBar 某一项右上角的文本	1.5.0

5.5、下拉刷新

名称	功能说明	最低版本
startPullDownRefresh	开始下拉刷新	1.5.0
stopPullDownRefresh	停止当前页面下拉刷新	1.5.0

5.6、滚动

名称	功能说明	最低版本
pageScrollTo	将页面滚动到目标位置，支持选择器和滚动距离两种方式定位	1.5.0
ScrollViewContext.scrollIntoView	滚动至指定位置	1.5.0
ScrollViewContext.scrollTo	滚动至指定位置	1.5.0

5.7、动画

名称	功能说明	最低版本
createAnimation	创建一个动画实例 animation	1.5.0
Animation.backgroundColor	设置背景色	1.5.0
Animation.bottom	设置 bottom 值	1.5.0
Animation.export	导出动画队列	1.5.0
Animation.height	设置高度	1.5.0
Animation.left	设置 left 值	1.5.0
Animation.matrix	同 transform-function matrix	1.5.0
Animation.matrix3d	同 transform-function matrix3d	1.5.0

Animation.opacity	设置透明度	1.5.0
Animation.right	设置 right 值	1.5.0
Animation.rotate	从原点顺时针旋转一个角度	1.5.0
Animation.rotate3d	从 固定 轴顺时针旋转一个角度	1.5.0
Animation.rotateX	从 X 轴顺时针旋转一个角度	1.5.0
Animation.rotateY	从 Y 轴顺时针旋转一个角度	1.5.0
Animation.rotateZ	从 Z 轴顺时针旋转一个角度	1.5.0
Animation.scale	缩放	1.5.0
Animation.scale3d	缩放	1.5.0
Animation.scaleX	缩放 X 轴	1.5.0
Animation.scaleY	缩放 Y 轴	1.5.0
Animation.scaleZ	缩放 Z 轴	1.5.0
Animation.skew	对 X、Y 轴坐标进行倾斜	1.5.0
Animation.skewX	对 X 轴坐标进行倾斜	1.5.0
Animation.skewY	对 Y 轴坐标进行倾斜	1.5.0
Animation.step	表示一组动画完成	1.5.0
Animation.top	设置 top 值	1.5.0
Animation.translate	平移变换	1.5.0
Animation.translate3d	对 xyz 坐标进行平移变换	1.5.0
Animation.translateX	对 X 轴平移	1.5.0
Animation.translateY	对 Y 轴平移	1.5.0
Animation.translateZ	对 Z 轴平移	1.5.0
Animation.width	设置宽度	1.5.0

5.8、自定义组件

名称	功能说明	最低版本

nextTick	延迟一部分操作到下一个时间片再执行	1.5.0

5.9、菜单

名称	功能说明	最低版本
getMenuButtonBoundingClientRect	获取菜单按钮（右上角胶囊按钮）的布局位置信息	1.5.0

5.10、窗口

名称	功能说明	最低版本
onWindowResize	监听窗口尺寸变化事件	1.5.0
offWindowResize	移除窗口尺寸变化事件的监听函数	1.5.1

5.11、字体

名称	功能说明	最低版本
loadFontFace	动态加载网络字体	1.5.0

6、网络

6.1、发起请求

名称	功能说明	最低版本
request	发起 HTTPS 网络请求	1.5.0
RequestTask.abort	中断请求任务	1.5.0
RequestTask.onChunkReceived	监听 Transfer-Encoding Chunk Received 事件	1.5.1
RequestTask.offChunkReceived	移除 Transfer-Encoding Chunk Received 事件的监听函数	1.5.1
RequestTask.onHeadersReceived	监听 HTTP Response Header 事件	1.5.0
RequestTask.offHeadersReceived	移除 HTTP Response Header 事件的监听函数	1.5.0

6.2、上传

名称	功能说明	最低版本
uploadFile	将本地资源上传到服务器	1.5.0
UploadTask.abort	中断上传任务	1.5.0
UploadTask.onProgressUpdate	监听上传进度变化事件	1.5.0
UploadTask.offProgressUpdate	取消监听上传进度变化事件	1.5.0
UploadTask.onHeadersReceived	监听 HTTP Response Header 事件	1.5.0
UploadTask.offHeadersReceived	移除 HTTP Response Header 事件的监听函数	1.5.0

6.3、下载

名称	功能说明	最低版本
downloadFile	下载文件资源到本地。客户端直接发起一个 HTTPS GET 请求，返回文件的本地临时路径 (本地路径)，单次下载允许的最大文件为200MB	1.5.1
DownloadTask.abort	中断下载任务	1.5.1
DownloadTask.onProgressUpdate	监听下载进度变化事件	1.5.1
DownloadTask.offProgressUpdate	移除下载进度变化事件的监听函数	1.5.1
DownloadTask.onHeadersReceived	监听 HTTP Response Header 事件	1.5.1
DownloadTask.offHeadersReceived	移除 HTTP Response Header 事件的监听函数	1.5.1

6.4、WebSocket

名称	功能说明	最低版本
sendSocketMessage	通过 WebSocket 连接发送数据。需要先	1.5.1

	wx.connectSocket，并在 wx.onSocketOpen 回调之后才能发送	
onSocketOpen	监听 WebSocket 连接打开事件	1.5.1
onSocketMessage	监听 WebSocket 接收到服务器的消息事件	1.5.1
onSocketError	监听 WebSocket 错误事件	1.5.1
onSocketClose	监听 WebSocket 连接关闭事件	1.5.1
connectSocket	创建一个 WebSocket 连接	1.5.1
closeSocket	关闭 WebSocket 连接	1.5.1
SocketTask.close	关闭 WebSocket 连接	1.5.0
SocketTask.onClose	监听 WebSocket 连接关闭事件	1.5.0
SocketTask.onError	监听 WebSocket 错误事件	1.5.0
SocketTask.onMessage	监听 WebSocket 接收到服务器的消息事件	1.5.0
SocketTask.onOpen	监听 WebSocket 连接打开事件	1.5.0
SocketTask.send	通过 WebSocket 连接发送数据	1.5.0

6.5、UDP 通信

名称	功能描述	最低版本
createUDPSocket	创建一个 UDP Socket 实例	1.5.1
UDPSocket.bind	绑定一个系统随机分配的可用端口，或绑定一个指定的端口号	1.5.1
UDPSocket.close	关闭 UDP Socket 实例	1.5.1
UDPSocket.connect	预先连接到指定的 IP 和 port，需要配合 write 方法一起使用	1.5.1
UDPSocket.onClose	监听关闭事件	1.5.1
UDPSocket.offClose	移除关闭事件的监听函数	1.5.1
UDPSocket.onError	监听错误事件	1.5.1
UDPSocket.offError	移除错误事件的监听函数	1.5.1

UDPSocket.onListening	监听开始监听数据包消息的事件	1.5.1
UDPSocket.offListening	移除开始监听数据包消息的事件的监听函数	1.5.1
UDPSocket.onMessage	监听收到消息的事件	1.5.1
UDPSocket.offMessage	移除收到消息的事件的监听函数	1.5.1
UDPSocket.send	向指定的 IP 和 port 发送消息	1.5.1
UDPSocket.setTTL	设置 IP_TTL 套接字选项，用于设置一个 IP 数据包传输时允许的最大跳步数	1.5.16
UDPSocket.write	用法与 send 方法相同	1.5.16

6.6、TCP 通信

名称	功能说明	最低版本
createTCPSocket	创建一个 TCP Socket 实例	1.5.1
TCPSocket.bindWifi	将 TCP Socket 绑定到当前 Wi-Fi 网络，成功后会触发 onBindWifi 事件（仅 Android 支持）	1.5.1
TCPSocket.close	关闭连接	1.5.1
TCPSocket.connect	在给定的套接字上启动连接	1.5.1
TCPSocket.onClose	监听一旦 socket 完全关闭就发出该事件	1.5.1
TCPSocket.offClose	移除一旦 socket 完全关闭就发出该事件的监听函数	1.5.1
TCPSocket.onConnect	监听当一个 socket 连接成功建立的时候触发该事件	1.5.1
TCPSocket.offConnect	移除当一个 socket 连接成功建立的时候触发该事件的监听函数	1.5.1
TCPSocket.onError	监听当错误发生时触发	1.5.1
TCPSocket.offError	移除当错误发生时触发的监听函数	1.5.1
TCPSocket.onMessage	监听当接收到数据的时触发该事件	1.5.1
TCPSocket.offMessage	移除当接收到数据的时触发该事件的监听函数	1.5.1
TCPSocket.onBindWifi	监听当一个 socket 绑定当前 Wi-Fi 网络成功时触发该事件	1.5.1
TCPSocket.offBindWifi	移除当一个 socket 绑定当前 Wi-Fi 网络成功时触发该	1.5.1

	事件的监听函数	
TCPSocket.write	在 socket 上发送数据	1.5.1

6.7、mDNS

名称	功能说明	最低版本
stopLocalServiceDiscovery	停止搜索 mDNS 服务	1.5.1
startLocalServiceDiscovery	开始搜索局域网下的 mDNS 服务	1.5.1
offLocalServiceResolveFail	移除 mDNS 服务解析失败的事件的监听函数	1.5.1
onLocalServiceResolveFail	监听 mDNS 服务解析失败的事件	1.5.1
offLocalServiceLost	移除 mDNS 服务离开的事件的监听函数	1.5.1
onLocalServiceLost	监听 mDNS 服务离开的事件	1.5.1
offLocalServiceFound	移除 mDNS 服务发现的事件的监听函数	1.5.1
onLocalServiceFound	监听 mDNS 服务发现的事件	1.5.1
offLocalServiceDiscoveryStop	移除 mDNS 服务停止搜索的事件的监听函数	1.5.1
onLocalServiceDiscoveryStop	监听 mDNS 服务停止搜索的事件	1.5.1

7、数据缓存

名称	功能说明	最低版本
setStorage	将数据存储在本地缓存中指定的 key 中	1.5.0
setStorageSync	将数据存储在本地缓存中指定的 key 中	1.5.0
revokeBufferURL	根据 URL 销毁存在内存中的数据	1.5.35
removeStorage	从本地缓存中移除指定 key	1.5.0

removeStorageSync	removeStorage的同步版本	1.5.0
getStorage	从本地缓存中异步获取指定 key 的内容	1.5.0
createBufferURL	根据传入的 buffer 创建一个唯一的 URL 存在内存中	1.5.35
getStorageSync	从本地缓存中同步获取指定 key 的内容	1.5.0
getStorageInfo	异步获取当前 storage 的相关信息	1.5.0
getStorageInfoSync	getStorageInfo的同步版本	1.5.0
clearStorage	清理本地数据缓存	1.5.0
clearStorageSync	clearStorage的同步版本	1.5.0
batchSetStorage	将数据批量存储在本地缓存中指定的 key 中	1.5.16
batchSetStorageSync	batchSetStorage 的同步版本	1.5.16
batchGetStorage	从本地缓存中异步批量获取指定 key 的内容	1.5.16
batchGetStorageSync	batchGetStorage 的同步版本	1.5.16

8、数据分析

名称	功能说明	最低版本
reportEvent	事件上报	1.5.16

9、画布

9.1、画布概要

名称	功能说明	最低版本
createOffscreenCanvas	创建离屏 canvas 实例	1.5.19
createCanvasContext	创建 canvas 的绘图上下文 CanvasContext 对象	1.5.0
canvasToTempFilePath	把当前画布指定区域的内容导出生成指定大小的图片	1.5.0

canvasPutImageData	将像素数据绘制到画布	1.5.0
canvasGetImageData	获取 canvas 区域隐含的像素数据	1.5.0
Image	图片对象	1.5.16
ImageData	ImageData 对象	1.5.16
RenderingContext	Canvas 绘图上下文	1.5.16

9.2、canvasContext

名称	功能说明	最低版本
CanvasContext.arc	创建一条弧线	1.5.0
CanvasContext.arcTo	根据控制点和半径绘制圆弧路径	1.5.0
CanvasContext.beginPath	开始创建一个路径	1.5.0
CanvasContext.bezierCurveTo	创建三次方贝塞尔曲线路径	1.5.0
CanvasContext.clearRect	清除画布上在该矩形区域内的内容	1.5.0
CanvasContext.clip	从原始画布中剪切任意形状和尺寸	1.5.0
CanvasContext.closePath	关闭一个路径	1.5.0
CanvasContext.createCircularGradient	创建一个圆形的渐变颜色	1.5.0
CanvasContext.createLinearGradient	创建一个线性的渐变颜色	1.5.0
CanvasContext.createPattern	对指定的图像创建模式的方法，可在指定的方向上重复元图像	1.5.0
CanvasContext.draw	将之前在绘图上下文中的描述（路径、变形、样式）画到 canvas 中	1.5.0
CanvasContext.drawImage	绘制图像到画布	1.5.0
CanvasContext.fill	对当前路径中的内容进行填充	1.5.0
CanvasContext.fillRect	填充一个矩形	1.5.0
CanvasContext.fillText	在画布上绘制被填充的文本	1.5.0

CanvasContext.lineTo	增加一个新点，然后创建一条从上次指定点到目标点的线	1.5.0
CanvasContext.measureText	测量文本尺寸信息	1.5.0
CanvasContext.moveTo	把路径移动到画布中的指定点，不创建线条	1.5.0
CanvasContext.quadraticCurveTo	创建二次贝塞尔曲线路径	1.5.0
CanvasContext.rect	创建一个矩形路径	1.5.0
CanvasContext.restore	恢复之前保存的绘图上下文	1.5.0
CanvasContext.rotate	以原点为中心顺时针旋转当前坐标轴	1.5.0
CanvasContext.save	保存绘图上下文	1.5.0
CanvasContext.scale	在调用后，之后创建的路径其横纵坐标会被缩放	1.5.0
CanvasContext.setFillStyle	设置填充色	1.5.0
CanvasContext.setFontSize	设置字体的字号	1.5.0
CanvasContext.setGlobalAlpha	设置全局画笔透明度	1.5.0
CanvasContext.setLineCap	设置线条的端点样式	1.5.0
CanvasContext.setLineDash	设置虚线样式	1.5.0
CanvasContext.setLineJoin	设置线条的交点样式	1.5.0
CanvasContext.setLineWidth	设置线条的宽度	1.5.0
CanvasContext.setMiterLimit	设置最大斜接长度	1.5.0
CanvasContext.setShadow	设定阴影样式	1.5.0

CanvasContext.setStrokeStyle	设置描边颜色	1.5.0
CanvasContext.setTextAlign	设置文字的对齐	1.5.0
CanvasContext.setTextBaseline	设置文字的竖直对齐	1.5.0
CanvasContext.setTransform	使用矩阵重新设置（覆盖）当前变换的方法	1.5.0
CanvasContext.stroke	画出当前路径的边框	1.5.0
CanvasContext.strokeRect	画一个矩形(非填充)	1.5.0
CanvasContext.strokeText	给定的 (x, y) 位置绘制文本描边的方法	1.5.0
CanvasContext.transform	使用矩阵多次叠加当前变换的方法	1.5.0
CanvasContext.translate	对当前坐标系的原点 (0, 0) 进行变换	1.5.0

9.3、CanvasGradient

名称	功能说明	最低版本
CanvasGradient.addColorStop	添加颜色的渐变点	1.5.0

9.4、Canvas

名称	功能说明	最低版本
Canvas.cancelAnimationFrame	取消由 requestAnimationFrame 添加到计划中的动画请求	1.5.19
Canvas.createImageData	创建一个ImageData 对象	1.5.19
Canvas.createImage	创建一个图片对象	1.5.19
Canvas.createPath2D	创建 Path2D 对象	1.5.19
Canvas.getContext	该方法返回 Canvas 的绘图上下文	1.5.19
Canvas.requestAnimationFrame	在下次进行重绘时执行	1.5.19

Canvas.toDataURL	返回一个包含图片展示的 data URI	1.5.19

9.5、OffscreenCanvas

名称	功能说明	最低版本
OffscreenCanvas.createImage	该方法返回 OffscreenCanvas 的绘图上下文	1.5.19
OffscreenCanvas.getContext	创建一个图片对象	1.5.19

9.6、Path2D

名称	功能说明	最低版本
Path2D.addPath	添加路径到当前路径	1.5.19
Path2D.arc	添加一段圆弧路径	1.5.19
Path2D.arcTo	通过给定控制点添加一段圆弧路径	1.5.19
Path2D.bezierCurveTo	添加三次贝塞尔曲线路径	1.5.19
Path2D.closePath	闭合路径到起点	1.5.19
Path2D.ellipse	添加椭圆弧路径	1.5.19
Path2D.lineTo	添加直线路径	1.5.19
Path2D.moveTo	移动路径开始点	1.5.19
Path2D.quadraticCurveTo	添加二次贝塞尔曲线路径	1.5.19
Path2D.rect	添加方形路径	1.5.19

10、媒体

10.1、地图

名称	功能说明	最低版本
createMapContext	创建 map 上下文 MapContext 对象	1.5.0
MapContext.addArc	添加弧线	1.5.16

MapContext.addCustomLayer	添加个性化图层	1.5.16
MapContext.addGroundOverlay	创建自定义图片图层	1.5.16
MapContext.addMarkers	添加 marker	1.5.0
MapContext.eraseLines	擦除或置灰已添加到地图中的线段	1.5.16
MapContext.fromScreenLocation	获取屏幕上的点对应的经纬度，坐标原点为地图左上角	1.5.0
MapContext.getCenterLocation	获取当前地图中心的经纬度	1.5.0
MapContext.getRegion	获取当前地图的视野范围	1.5.0
MapContext.getRotate	获取当前地图的旋转角	1.5.0
MapContext.getScale	获取当前地图的缩放级别	1.5.0
MapContext.getSkew	获取当前地图的倾斜角	1.5.0
MapContext.includePoints	缩放视野展示所有经纬度	1.5.0
MapContext.initMarkerCluster	初始化点聚合的配置	1.5.16
MapContext.moveAlong	沿指定路径移动 marker，用于轨迹回放等场景	1.5.0
MapContext.moveToLocation	将地图中心移置当前定位点，此时需设置地图组件 show-location 为 true	1.5.0
MapContext.on	监听地图事件	1.5.16
MapContext.openMapApp	拉起地图 APP 选择导航	1.5.0
MapContext.removeArc	删除弧线	1.5.16
MapContext.removeCustomLayer	移除个性化图层	1.5.16
MapContext.removeGroundOverlay	移除自定义图片图层	1.5.16

MapContext.removeMarkers	移除 marker	1.5.0
MapContext.setBoundary	限制地图的显示范围	1.5.16
MapContext.setCenterOffset	设置地图中心点偏移	1.5.16
MapContext.setLocMarkerIcon	设置定位点图标	1.5.16
MapContext.toScreenLocation	获取经纬度对应的屏幕坐标	1.5.16
MapContext.translateMarker	平移marker，带动画	1.5.0
MapContext.updateGroundOverlay	更新自定义图片图层	1.5.16

10.2、图片

名称	功能说明	最低版本
chooseImage	从本地相册选择图片或使用相机拍照	1.5.0
compressImage	压缩图片接口，可选压缩质量	1.5.1
getImageInfo	获取图片信息	1.5.0
previewImage	在新页面中全屏预览图片	1.5.0
previewMedia	预览图片和视频	1.5.1
saveImageToPhotosAlbum	保存图片到系统相册	1.5.1
chooseMessageFile	从客户端会话选择文件	1.5.1

10.3、实时音视频

名称	功能	最低版本
createLivePusherContext	创建 live-pusher 上下文 LivePusherContext 对象	1.5.0
createLivePlayerContext	创建 live-player 上下文 LivePlayerContext 对象	1.5.0

LivePlayerContext.exitfullscreen	退出全屏	1.5.0
LivePlayerContext.mute	静音	1.5.0
LivePlayerContext.pause	暂停	1.5.0
LivePlayerContext.play	播放	1.5.0
LivePlayerContext.requestFullScreen	进入全屏	1.5.0
LivePlayerContext.resume	恢复	1.5.0
LivePlayerContext.snapshot	截图	1.5.0
LivePlayerContext.stop	停止	1.5.0
LivePusherContext.start	开始推流，同时开启摄像头预览	1.5.0
LivePusherContext.stop	停止推流，同时停止摄像头预览	1.5.0
LivePusherContext.pause	暂停推流	1.5.0
LivePusherContext.resume	恢复推流	1.5.0
LivePusherContext.switchCamera	切换前后摄像头	1.5.0
LivePusherContext.snapshot	快照	1.5.0
LivePusherContext.toggleTorch	切换手电筒	1.5.0
LivePusherContext.playBGM	播放背景音	1.5.0
LivePusherContext.stopBGM	停止背景音	1.5.0
LivePusherContext.pauseBGM	暂停背景音	1.5.0
LivePusherContext.resumeBGM	恢复背景音	1.5.0

LivePusherContext.setBGMPosition	设置背景音进度	1.5.0
LivePusherContext.setMICVolume	设置麦克风音量	1.5.0
LivePusherContext.setBGMVolume	设置背景音量	1.5.0
LivePusherContext.setAudioReverbType	设置混音类型	1.5.0
LivePusherContext.startPreview	开启摄像头预览	1.5.0
LivePusherContext.stopPreview	关闭摄像头预览	1.5.0
LivePusherContext.startAudioRecord	开始录音	1.5.0
LivePusherContext.stopAudioRecord	结束录音	1.5.0

10.4、音频

名称	功能说明	最低版本
stopVoice	结束播放语音	1.5.1
playVoice	开始播放语音	1.5.1
pauseVoice	暂停正在播放的语音	1.5.1
setInnerAudioOption	设置 InnerAudioContext 的播放选项	1.5.0
getAvailableAudioSources	获取当前支持的音频输入源	1.5.1
createAudioContext	创建内部 'audio'组件 上下文 InnerAudioContext 对象	1.5.0
createInnerAudioContext	创建内部 'audio'组件 上下文 InnerAudioContext 对象	1.5.0
AudioContext.pause	暂停音频	1.5.1
AudioContext.play	播放音频	1.5.1

AudioContext.seek	跳转到指定位置	1.5.1
AudioContext.setSrc	设置音频地址	1.5.1
InnerAudioContext.destroy	销毁当前实例	1.5.0
InnerAudioContext.offCanplay	移除音频进入可以播放状态的事件的监听函数	1.5.0
InnerAudioContext.offEnded	移除音频自然播放至结束的事件的监听函数	1.5.0
InnerAudioContext.offError	移除音频播放错误事件的监听函数	1.5.0
InnerAudioContext.offPause	移除音频暂停事件的监听函数	1.5.0
InnerAudioContext.offPlay	移除音频播放事件的监听函数	1.5.0
InnerAudioContext.offSeeked	移除音频完成跳转操作的事件的监听函数	1.5.0
InnerAudioContext.offSeeking	移除音频进行跳转操作的事件的监听函数	1.5.0
InnerAudioContext.offStop	移除音频停止事件的监听函数	1.5.0
InnerAudioContext.offTimeUpdate	移除音频播放进度更新事件的监听函数	1.5.0
InnerAudioContext.offWaiting	移除音频加载中事件的监听函数	1.5.0
InnerAudioContext.onCanplay	监听音频进入可以播放状态的事件	1.5.0
InnerAudioContext.onEnded	监听音频自然播放至结束的事件	1.5.0
InnerAudioContext.onError	监听音频播放错误事件	1.5.0
InnerAudioContext.onPause	监听音频暂停事件	1.5.0

InnerAudioContext.onPlay	监听音频播放事件	1.5.0
InnerAudioContext.onSeeked	监听音频完成跳转操作的事件	1.5.0
InnerAudioContext.onSeeking	监听音频进行跳转操作的事件	1.5.0
InnerAudioContext.onStop	监听音频停止事件	1.5.0
InnerAudioContext.onTimeUpdate	监听音频播放进度更新事件	1.5.0
InnerAudioContext.onWaiting	监听音频加载中事件	1.5.0
InnerAudioContext.pause	暂停	1.5.0
InnerAudioContext.play	播放	1.5.0
InnerAudioContext.seek	跳转到指定位置	1.5.0
InnerAudioContext.stop	停止	1.5.0

10.5、视频

名称	说明	最低版本
saveVideoToPhotosAlbum	保存视频到系统相册	1.5.0
createVideoContext	创建 video 上下文 VideoContext 对象	1.5.0
chooseVideo	拍摄视频或从手机相册中选视频	1.5.0
compressVideo	压缩视频接口	1.5.1
chooseMedia	拍摄或从手机相册中选择图片或视频	1.5.1
VideoContext.exitFullScreen	退出全屏	1.5.0
VideoContext.exitBackgroundPlayback	退出后台音频播放模式	1.5.16
VideoContext.exitPictureInPicture	退出小窗，该方法可在任意页面调用	1.5.16

VideoContext.hideStatusBar	隐藏状态栏，仅在 iOS 全屏下有效	1.5.0
VideoContext.pause	暂停视频	1.5.0
VideoContext.play	播放视频	1.5.0
VideoContext.requestFullScreen	进入全屏	1.5.0
VideoContext.requestBackgroundPlayback	进入后台音频播放模式	1.5.16
VideoContext.seek	跳转到指定位置	1.5.0
VideoContext.sendDanmu	发送弹幕	1.5.0
VideoContext.showStatusBar	显示状态栏，仅在 iOS 全屏下有效	1.5.0
VideoContext.stop	停止视频	1.5.0

10.6、透明视频

名称	说明	最低版本
createAnimationVideo	创建 animation-video 上下文 AnimationVideoContext 对象	1.5.0
AnimationVideoContext.play	播放视频	1.5.0
AnimationVideoContext.pause	暂停视频	1.5.0
AnimationVideoContext.seek	跳转到指定位置	1.5.0

10.7、相机

名称	功能说明	最低版本
createCameraContext	创建 camera 上下文 CameraContext 对象	1.5.0
CameraContext.onCameraFrame	获取 Camera 实时帧数据	1.5.0
CameraContext.setZoom	设置缩放级别	1.5.0

CameraContext.startRecord	开始录像	1.5.0
CameraContext.stopRecord	结束录像	1.5.0
CameraContext.takePhoto	拍摄照片	1.5.0
CameraFrameListener.start	开始监听帧数据	1.5.0
CameraFrameListener.stop	停止监听帧数据	1.5.0

10.8、录音

名称	功能	最低版本
startRecord	开始录音	1.5.1
stopRecord	停止录音	1.5.1
getRecorderManager	获取全局唯一的录音管理器 RecorderManager	1.5.0
RecorderManager.onError	监听录音错误事件	1.5.0
RecorderManager.onFrameRecorded	监听已录制完指定帧大小的文件事件	1.5.0
RecorderManager.onInterruptionBegin	监听录音因为受到系统占用而被中断开始事件	1.5.0
RecorderManager.onInterruptionEnd	监听录音中断结束事件	1.5.0
RecorderManager.onPause	监听录音暂停事件	1.5.0
RecorderManager.onResume	监听录音继续事件	1.5.0
RecorderManager.onStart	监听录音开始事件	1.5.0
RecorderManager.onStop	监听录音结束事件	1.5.0

RecorderManager.pause	暂停录音	1.5.0
RecorderManager.resume	继续录音	1.5.0
RecorderManager.start	开始录音	1.5.0
RecorderManager.stop	暂停录音	1.5.0

10.9、背景音频

名称	说明	最低版本
stopBackgroundAudio	停止播放音乐	1.5.1
seekBackgroundAudio	控制音乐播放进度	1.5.1
playBackgroundAudio	使用后台播放器播放音乐	1.5.1
pauseBackgroundAudio	暂停播放音乐监听音乐停止事件	1.5.1
onBackgroundAudioStop	监听音乐停止事件	1.5.1
onBackgroundAudioPlay	监听音乐播放事件	1.5.1
onBackgroundAudioPause	监听音乐暂停事件	1.5.1
getBackgroundAudioPlayerState	获取后台音乐播放状态	1.5.1
getBackgroundAudioManager	获取全局唯一的背景音频管理器	1.5.1
BackgroundAudioManager.onCanplay	监听背景音频进入可播放状态事件	1.5.16
BackgroundAudioManager.onEnded	监听背景音频自然播放结束事件	1.5.16
BackgroundAudioManager.onError	监听背景音频播放错误事件	1.5.16
BackgroundAudioManager.onNext	监听用户在系统音乐播放面板点击下一曲事件（仅iOS）	1.5.16
BackgroundAudioManager.onPause	监听背景音频暂停事件	1.5.16

BackgroundAudioManager.onPlay	监听背景音频播放事件	1.5.16
BackgroundAudioManager onPrev	监听用户在系统音乐播放面板点击上一曲事件（仅 iOS）	1.5.16
BackgroundAudioManager.onSeeked	监听背景音频完成跳转操作事件	1.5.16
BackgroundAudioManager.orSeeking	监听背景音频开始跳转操作事件	1.5.16
BackgroundAudioManager.onStop	监听背景音频停止事件	1.5.16
BackgroundAudioManager.onTimeUpdate	监听背景音频播放进度更新事件	1.5.16
BackgroundAudioManager.onWaiting	监听音频加载中事件	1.5.16
BackgroundAudioManager.pause	暂停音乐	1.5.16
BackgroundAudioManager.play	播放音乐	1.5.16
BackgroundAudioManager.seek	跳转到指定位置	1.5.16
BackgroundAudioManager.stop	停止音乐	1.5.16

10.10、富文本

名称	说明	最低版本
EditorContext	EditorContext 实例	1.5.16
EditorContext.blur	编辑器失焦，同时收起键盘	1.5.16
EditorContext.clear	清空编辑器内容	1.5.16
EditorContext.format	修改样式	1.5.16
EditorContext.getContents	获取编辑器内容	1.5.16

EditorContext.getSelectionText	获取编辑器已选区域内的纯文本内容	1.5.16
EditorContext.insertDivider	插入分割线	1.5.16
EditorContext.insertImage	插入图片	1.5.16
EditorContext.insertText	覆盖当前选区，设置一段文本	1.5.16
EditorContext.redo	恢复	1.5.16
EditorContext.removeFormat	清除当前选区的样式	1.5.16
EditorContext.scrollIntoView	使得编辑器光标处滚动到窗口可视区域内	1.5.16
EditorContext.setContents	初始化编辑器内容	1.5.16
EditorContext.undo	撤销	1.5.1

10.11、画面录制器

名称	说明	最低版本
createMediaRecorder	创建 WebGL 画面录制器	1.5.16
MediaRecorder.destroy	销毁录制器	1.5.16
MediaRecorder.off	取消监听录制事件	1.5.16
MediaRecorder.on	注册监听录制事件的回调函数	1.5.16
MediaRecorder.pause	暂停录制	1.5.16
MediaRecorder.requestFrame	请求下一帧录制	1.5.16
MediaRecorder.resume	恢复录制	1.5.16
MediaRecorder.start	开始录制	1.5.16
MediaRecorder.stop	暂停录制	1.5.16

11、文件

11.1、文件概要

名称	说明	最低版本
saveFile	保存文件到本地	1.5.0
removeSavedFile	删除本地缓存文件	1.5.0
openDocument	新开页面打开文档	1.5.0
getSavedFileList	获取该小程序下已保存的本地缓存文件列表	1.5.0
getSavedFileInfo	获取本地文件的文件信息	1.5.0
getFileSystemManager	获取全局唯一的文件管理器	1.5.0
getFileInfo	获取文件信息	1.5.0
ReadResult	文件读取结果	1.5.16
WriteResult	文件写入结果	1.5.16

11.2、FileSystemManager

API	说明	最低版本
FileSystemManager.access	判断文件/目录是否存在	1.5.0
FileSystemManager.accessSync	FileSystemManager.access 的同步版本	1.5.0
FileSystemManager.appendFile	在文件结尾追加内容	1.5.0
FileSystemManager.appendFileSync	FileSystemManager.appendFile 的同步版本	1.5.0
FileSystemManager.close	关闭文件	1.5.16
FileSystemManager.closeSync	同步关闭文件	1.5.16
FileSystemManager.copyFile	复制文件	1.5.0

FileSystemManager.copyFileSync	FileSystemManager.copyFile 的同步版本	1.5.0
FileSystemManager.fstat	获取文件的状态信息	1.5.16
FileSystemManager.fstatSync	同步获取文件的状态信息	1.5.16
FileSystemManager.ftruncate	对文件内容进行截断操作 FileSystemManager.open	1.5.16
FileSystemManager.ftruncateSync	对文件内容进行截断操作	1.5.16
FileSystemManager.getFileInfo	获取该小程序下的 本地临时文件 或 本地缓存文件 信息	1.5.0
FileSystemManager.getSavedFileList	获取该小程序下已保存的本地缓存文件列表	1.5.0
FileSystemManager.mkdir	创建目录	1.5.0
FileSystemManager.mkdirSync	FileSystemManager.mkdir 的同步版本	1.5.0
FileSystemManager.open	打开文件，返回文件描述符	1.5.16
FileSystemManager.openSync	同步打开文件，返回文件描述符	1.5.16
FileSystemManager.read	读文件	1.5.16
FileSystemManager.readSync	读文件	1.5.16
FileSystemManager.readCompressedFile	读取指定压缩类型的本地文件内容	1.5.16
FileSystemManager.readCompressedFileSync	同步读取指定压缩类型的本地文件内容	1.5.16
FileSystemManager.readdir	读取目录内文件列表	1.5.0
FileSystemManager.readdirSync	FileSystemManager.readdir 的同步版本	1.5.0
FileSystemManager.readFile	读取本地文件内容	1.5.0

FileSystemManager.readFileSync	FileSystemManager.readFile 的同步版本	1.5.0
FileSystemManager.readZipEntry	读取压缩包内的文件	1.5.16
FileSystemManager.removeSavedFile	删除该小程序下已保存的本地缓存文件	1.5.0
FileSystemManager.rename	重命名文件	1.5.0
FileSystemManager.renameSync	FileSystemManager.rename 的同步版本	1.5.0
FileSystemManager.rmdir	删除目录	1.5.0
FileSystemManager.rmdirSync	FileSystemManager.rmdir 的同步版本	1.5.0
FileSystemManager.saveFile	保存临时文件到本地	1.5.0
FileSystemManager.saveFileSync	FileSystemManager.saveFile 的同步版本	1.5.0
FileSystemManager.stat	获取文件 Stats 对象	1.5.0
FileSystemManager.statSync	FileSystemManager.stat 的同步版本	1.5.0
FileSystemManager.truncate	对文件内容进行截断操作	1.5.16
FileSystemManager.truncateSync	对文件内容进行截断操作 (truncate 的同步版本)	1.5.16
FileSystemManager.unlink	删除文件	1.5.0
FileSystemManager.unlinkSync	FileSystemManager.unlink 的同步版本	1.5.0
FileSystemManager.unzip	解压文件	1.5.0
FileSystemManager.write	写入文件	1.5.16
FileSystemManager.write	同步写入文件	1.5.16

Sync		
FileSystemManager.writeFile	写文件	1.5.0
FileSystemManager.writeFileSync	FileSystemManager.writeFile 的同步版本	1.5.0

11.3、Stats

API	说明	最低版本
Stats.isDirectory	判断当前文件是否一个目录	1.5.0
Stats.isFile	判断当前文件是否一个普通文件	1.5.0

12、开放接口

12.1、登录

名称	功能说明	最低版本
login	调用接口获取登录凭证（code）	1.5.1
checkSession	检查登录态是否过期	1.5.1

12.2、账号信息

名称	功能说明	最低版本
getAccountInfoSync	获取当前账号信息	1.5.1

12.3、用户信息

名称	功能说明	最低版本
getUserProfile	获取用户信息	1.5.1
getUserInfo	获取用户信息，在使用过程中需要用户授权 scope.userInfo	1.5.1
userInfo	用户信息	1.5.0

12.4、设置

名称	功能说明	最低版本
AuthSetting	用户授权设置信息	1.5.0
getSetting	获取用户的当前设置	1.5.0
openSetting	调起客户端小程序设置界面，返回用户设置的操作结果	1.5.0

12.5、生物认证

名称	功能说明	最低版本
AuthSetting	用户授权设置信息	1.5.0
getSetting	获取用户的当前设置	1.5.0
openSetting	调起客户端小程序设置界面，返回用户设置的操作结果	1.5.0

12.6、授权

名称	功能说明	最低版本
authorize	提前向用户发起授权请求	1.5.0

13、位置

名称	功能说明	最低版本
getLocation	获取当前的地理位置、速度	1.5.0
choosePoi	打开 POI 列表选择位置，支持模糊定位（精确到市）和精确定位混选	1.5.0
chooseLocation	打开地图选择位置	1.5.0
stopLocationUpdate	关闭监听实时位置变化，前后台都停止消息接收	1.5.0
startLocationUpdateBackground	开启小程序进入前后台时均接收位置消息，需引导用户开启'授权'	1.5.0
startLocationUpdate	开启小程序进入前台时接收位置消息	1.5.0
openLocation	使用微信内置地图查看位置	1.5.0

onLocationChangeError	监听持续定位接口返回失败时触发	1.5.0
onLocationChange	监听实时地理位置变化事件，需结合 startLocationUpdateBackground、startLocationUpdate 使用	1.5.0
offLocationChangeError	移除持续定位接口返回失败时触发	1.5.0
offLocationChange	取消监听实时地理位置变化事件	1.5.0
getFuzzyLocation	获取当前的模糊地理位置	1.5.0

14、设备

14.1、蓝牙通用

名称	功能说明	最低版本
stopBluetoothDevicesDiscovery	停止搜寻附近的蓝牙外围设备	1.5.0
startBluetoothDevicesDiscovery	开始搜寻附近的蓝牙外围设备	1.5.0
openBluetoothAdapter	初始化蓝牙模块	1.5.1
getConnectedBluetoothDevices	根据主服务 UUID 获取已连接的蓝牙设备	1.5.1
getBluetoothDevices	获取在蓝牙模块生效期间所有搜索到的蓝牙设备	1.5.1
getBluetoothAdapterState	获取本机蓝牙适配器状态	1.5.1
closeBluetoothAdapter	关闭蓝牙模块	1.5.1
onBluetoothDeviceFound	监听搜索到新设备的事件	1.5.0
offBluetoothDeviceFound	移除搜索到新设备的事件的全部监听函数	1.5.0
onBluetoothAdapterStateChange	监听蓝牙适配器状态变化事件	1.5.0
offBluetoothAdapterStateChange	移除蓝牙适配器状态变化事件的全部监听函数	1.5.0
makeBluetoothPair	蓝牙配对接口，仅 Android 支持	1.5.0

isBluetoothDevicePaired	查询蓝牙设备是否配对，仅 Android 支持	1.5.0

14.2、蓝牙-低功耗中心设备

名称	功能说明	最低版本
writeBLECharacteristicValue	向蓝牙低功耗设备特征值中写入二进制数据	1.5.1
readBLECharacteristicValue	读取蓝牙低功耗设备特征值的二进制数据	1.5.1
setBLEMTU	协商设置蓝牙低功耗的最大传输单元	1.5.1
getBLEMTU	获取蓝牙低功耗的最大传输单元	1.5.1
onBLEMTUChange	监听蓝牙低功耗的最大传输单元变化事件（仅 Android 触发）	1.5.1
offBLEMTUChange	移除蓝牙低功耗的最大传输单元变化事件的监听函数	1.5.1
onBLEConnectionStateChange	监听蓝牙低功耗连接状态改变事件	1.5.1
offBLEConnectionStateChange	移除蓝牙低功耗连接状态改变事件的监听函数	1.5.1
onBLECharacteristicValueChange	监听蓝牙低功耗设备的特征值变化事件	1.5.1
offBLECharacteristicValueChange	移除蓝牙低功耗设备的特征值变化事件的全部监听函数	1.5.1
notifyBLECharacteristicValueChange	启用蓝牙低功耗设备特征值变化时的 notify 功能，订阅特征	1.5.1
getBLEDeviceServices	获取蓝牙低功耗设备所有服务 (service)	1.5.1
getBLEDeviceRSSI	获取蓝牙低功耗设备的信号强度	1.5.1
getBLEDeviceCharacteristics	获取蓝牙低功耗设备某个服务中所有特征	1.5.1
createBLEConnection	连接蓝牙低功耗设备	1.5.1
closeBLEConnection	断开与蓝牙低功耗设备的连接	1.5.1

14.3、蓝牙-低功耗外围设备

名称	功能说明	最低版本
onBLEPeripheralConnectionStateChanged	监听当前外围设备被连接或断开连接事件	1.5.1
offBLEPeripheralConnectionStateChanged	移除当前外围设备被连接或断开连接事件的监听函数	1.5.1
createBLEPeripheralServer	建立本地作为蓝牙低功耗外围设备的服务端	1.5.1
BLEPeripheraServer.addService	添加服务	1.5.1
BLEPeripheraServer.removeService	移除服务	1.5.1
BLEPeripheraServer.startAdvertising	开始广播本地创建的外围设备	1.5.1
BLEPeripheraServer.stopAdvertising	停止广播	1.5.1
BLEPeripheraServer..writeCharacteristicValue	往指定特征写入二进制数据值，并通知已连接的主机	1.5.1
BLEPeripheraServer.onCharacteristicWriteRequest	监听已连接的设备请求写当前外围设备的特征值事件	1.5.1
BLEPeripheraServer.offCharacteristicWriteRequest	移除已连接的设备请求写当前外围设备的特征值事件的监听函数	1.5.1
BLEPeripheraServer.onCharacteristicReadRequest	监听已连接的设备请求读当前外围设备的特征值事件	1.5.1
BLEPeripheraServer.offCharacteristicReadRequest	移除已连接的设备请求读当前外围设备的特征值事件的监听函数	1.5.1
BLEPeripheraServer.onCharacteristicSubscribed	监听特征订阅事件，仅 iOS 支持	1.5.1
BLEPeripheraServer.offCharacteristicSubscribed	移除特征订阅事件的监听函数	1.5.1
BLEPeripheraServer.onCharacteristicUnsubscribed	监听取消特征订阅事件，仅 iOS 支持	1.5.1

BLEPeripheraServer.offCharacteristicUnsubscribed	移除取消特征订阅事件的监听函数	1.5.1

14.4、蓝牙-信标

名称	功能说明	最低版本
stopBeaconDiscovery	停止搜索附近的 Beacon 设备	1.5.1
startBeaconDiscovery	开始搜索附近的 Beacon 设备	1.5.1
onBeaconUpdate	监听 Beacon 设备更新事件，仅能注册一个监听	1.5.1
offBeaconUpdate	移除 Beacon 设备更新事件的全部监听函数	1.5.1
onBeaconServiceChange	监听 Beacon 服务状态变化事件，仅能注册一个监听	1.5.1
offBeaconServiceChange	移除 Beacon 服务状态变化事件的全部监听函数	1.5.1
getBeacons	获取所有已搜索到的 Beacon 设备	1.5.1
BeaconInfo	Beacon 设备	1.5.1

14.5、NFC

14.5.1、getNFCAdapter

名称	功能说明	最低版本
getNFCAdapter	获取 NFC 实例	1.5.0

14.5.2、NFCAdapter

名称	功能说明	最低版本
NFCAdapter.startDiscovery	开启设备	1.5.0
NFCAdapter.stopDiscovery	关闭设备	1.5.0
NFCAdapter.getNdef	获取 Ndef 实例，实例支持对 NDEF 格式的 NFC 标签上的 NDEF 数据的读写	1.5.0
NFCAdapter.getNfcA	获取 NfcA 实例，实例支持NFC-A (ISO 14443-3A)标准的读写	1.5.0

NFCAdapter.getNfcB	获取 NfcB 实例，实例支持NFC-B (ISO 14443-3B)标准的读写	1.5.0
NFCAdapter.getNfcF	获取 NfcF 实例，实例支持NFC-F (JIS 6319-4)标准的读写	1.5.0
NFCAdapter.getNfcV	获取 NfcV 实例，实例支持NFC-V (ISO 15693)标准的读写	1.5.0
NFCAdapter.getIsoDep	获取 IsoDep 实例，实例支持ISO-DEP (ISO 14443-4)标准的读写	1.5.0
NFCAdapter.getMifareClassic	获取 MifareClassic 实例，实例支持MIFARE Classic标签的读写	1.5.0
NFCAdapter.getMifareUltralight	获取 MifareUltralight 实例，实例支持MIFARE Ultralight标签的读写	1.5.0
NFCAdapter.onDiscovered	监听 NFC Tag	1.5.0
NFCAdapter.offDiscovered	移除 NFC Tag的监听函数	1.5.0

14.5.3、IsoDep

名称	功能说明	最低版本
IsoDep.connect	连接 NFC 标签	1.5.0
IsoDep.close	断开连接	1.5.0
IsoDep.setTimeout	设置超时时间	1.5.0
IsoDep.isConnected	检查是否已连接	1.5.0
IsoDep.getMaxTransceiveLength	获取最大传输长度	1.5.0
IsoDep.transceive	发送数据	1.5.0
IsoDep.getHistoricalBytes	获取复位信息	1.5.0

14.5.4、MifareClassic

名称	功能说明	最低版本

MifareClassic.connect	连接 NFC 标签	1.5.0
MifareClassic.close	断开连接	1.5.0
MifareClassic.setTimeout	设置超时时间	1.5.0
MifareClassic.isConnected	检查是否已连接	1.5.0
MifareClassic.getMaxTransceiveLength	获取最大传输长度	1.5.0
MifareClassic.transceive	发送数据	1.5.0

14.5.5、MifareUltralifght

名称	功能说明	最低版本
MifareUltralight.connect	连接 NFC 标签	1.5.0
MifareUltralight.close	断开连接	1.5.0
MifareUltralight.setTimeout	设置超时时间	1.5.0
MifareUltralight.isConnected	检查是否已连接	1.5.0
MifareUltralight.getMaxTransceiveLength	获取最大传输长度	1.5.0
MifareUltralight.transceive	发送数据	1.5.0

14.5.6、Ndef

名称	功能说明	最低版本
Ndef.connect	连接 NFC 标签	1.5.0
Ndef.close	断开连接	1.5.0
Ndef.setTimeout	设置超时时间	1.5.0
Ndef.isConnected	检查是否已连接	1.5.0
Ndef.offNdefMessage	取消监听 Ndef 消息	1.5.0

Ndef.onNdefMessage	监听 Ndef 消息	1.5.0
Ndef.writeNdefMessage	重写 Ndef 标签内容	1.5.0

14.5.7、NfcA

名称	功能说明	最低版本
NfcA.connect	连接 NFC 标签	1.5.0
NfcA.close	断开连接	1.5.0
NfcA.setTimeout	设置超时时间	1.5.0
NfcA.isConnected	检查是否已连接	1.5.0
NfcA.getMaxTransceiveLength	获取最大传输长度	1.5.0
NfcA.transceive	发送数据	1.5.0
NfcA.getAtqa	获取 ATQA 信息	1.5.0
NfcA.getSak	获取 SAK 信息	1.5.0

14.5.8、NfcB

名称	功能说明	最低版本
NfcB.connect	连接 NFC 标签	1.5.0
NfcB.close	断开连接	1.5.0
NfcB.setTimeout	设置超时时间	1.5.0
NfcB.isConnected	检查是否已连接	1.5.0
NfcB.getMaxTransceiveLength	获取最大传输长度	1.5.0
NfcB.transceive	发送数据	1.5.0

14.5.9、NfcF

名称	功能说明	最低版本
NfcF.connect	连接 NFC 标签	1.5.0

NfcF.close	断开连接	1.5.0
NfcF.setTimeout	设置超时时间	1.5.0
NfcF.isConnected	检查是否已连接	1.5.0
NfcF.getMaxTransceiveLength	获取最大传输长度	1.5.0
NfcF.transceive	发送数据	1.5.0

14.5.10、NfcV

名称	功能说明	最低版本
NfcV.connect	连接 NFC 标签	1.5.0
NfcV.close	断开连接	1.5.0
NfcV.setTimeout	设置超时时间	1.5.0
NfcV.isConnected	检查是否已连接	1.5.0
NfcV.getMaxTransceiveLength	获取最大传输长度	1.5.0
NfcV.transceive	发送数据	1.5.0

14.6、WiFi

名称	功能说明	最低版本
onWifiConnected	监听连接上 Wi-Fi 的事件	1.5.1
offWifiConnected	移除连接上 Wi-Fi 的事件的监听函数	1.5.1
stopWifi	关闭 Wi-Fi 模块	1.5.0
startWifi	初始化 Wi-Fi 模块	1.5.0
onGetWifiList	监听获取到 Wi-Fi 列表数据事件	1.5.0
offGetWifiList	移除获取到 Wi-Fi 列表数据事件的监听函数	1.5.0
getConnectedWifi	获取已连接中的 Wi-Fi 信息	1.5.0
WifiInfo	Wifi 信息	1.5.1

14.7、日历

名称	功能说明	最低版本
addPhoneRepeatCalendar	向系统日历添加重复事件	1.5.1
addPhoneCalendar	向系统日历添加事件	1.5.1

14.8、剪切板

名称	功能说明	最低版本
setClipboardData	设置系统剪贴板的内容	1.5.1
getClipboardData	获取系统剪贴板的内容	1.5.1

14.9、网络

名称	功能说明	最低版本
onNetworkStatusChange	监听网络状态变化事件	1.5.1
getNetworkType	获取网络类型	1.5.1
offNetworkStatusChange	移除网络状态变化事件的监听函数	1.5.1

14.10、加密

名称	功能说明	最低版本
getRandomValues	获取密码学安全随机数	1.5.1

14.11、键盘

名称	功能说明	最低版本
onKeyboardHeightChange	监听键盘高度变化事件	1.5.1
hideKeyboard	在input、textarea等 focus 拉起键盘之后，手动调用此接口收起键盘	1.5.1
offKeyboardHeightChange	移除键盘高度变化事件的监听函数	1.5.1

getSelectedTextRange	在input、textarea等 focus 之后，获取输入框的光标位置	1.5.1

14.12、电话

名称	功能说明	最低版本
makePhoneCall	拨打电话	1.5.1

14.13、加速计

名称	功能说明	最低版本
startAccelerometer	开始监听加速度数据	1.5.1
stopAccelerometer	停止监听加速度数据	1.5.1
onAccelerometerChange	监听加速度数据事件	1.5.1
offAccelerometerChange	移除加速度数据事件的监听函数	1.5.1

14.14、罗盘

名称	功能说明	最低版本
startCompass	开始监听罗盘数据	1.5.0
stopCompass	停止监听罗盘数据	1.5.0
onCompassChange	监听罗盘数据变化事件	1.5.0
offCompassChange	移除罗盘数据变化事件的监听函数	1.5.0

14.15、陀螺仪

名称	功能说明	最低版本
startGyroscope	开始监听陀螺仪数据	1.5.1
stopGyroscope	停止监听陀螺仪数据	1.5.1
onGyroscopeChange	监听陀螺仪数据变化事件	1.5.1
offGyroscopeChange	移除陀螺仪数据变化事件的监听函数	1.5.1

14.16、扫码

名称	功能说明	最低版本
scanCode	调起客户端扫码界面进行扫码	1.5.0

14.17、短信

名称	功能说明	最低版本
sendSms	拉起手机发送短信界面	1.5.1

14.18、振动

名称	功能说明	最低版本
vibrateShort	使手机发生较短时间的振动（15 ms）	1.5.1
vibrateLong	使手机发生较长时间的振动（400 ms）	1.5.1

14.19、联系人

名称	功能说明	最低版本
chooseContact	拉起手机通讯录，选择联系人	1.5.1
addPhoneContact	添加手机通讯录联系人	1.5.1

14.20、电量

名称	功能说明	最低版本
getBatteryInfoSync	wx.getBatteryInfo 的同步版本	1.5.1
getBatteryInfo	获取设备电量	1.5.1

14.21、屏幕

名称	功能说明	最低版本
setVisualEffectOnCapture	设置截屏/录屏时屏幕表现	1.5.1
setScreenBrightness	设置屏幕亮度	1.5.1
setKeepScreenOn	设置是否保持常亮状态	1.5.1
onUserCaptureScreen	监听用户主动截屏事件	1.5.1

onScreenRecordingStateChanged	监听用户录屏事件	1.5.1
offUserCaptureScreen	用户主动截屏事件	1.5.1
offScreenRecordingStateChanged	移除用户录屏事件的监听函数	1.5.1
getScreenRecordingState	查询用户是否在录屏	1.5.1
getScreenBrightness	获取屏幕亮度	1.5.1

14.22、设备方向

名称	功能说明	最低版本
stopDeviceMotionListening	停止监听设备方向的变化	1.5.1
startDeviceMotionListening	开始监听设备方向的变化	1.5.1
onDeviceMotionChange	监听设备方向变化事件	1.5.1
offDeviceMotionChange	移除设备方向变化事件的监听函数	1.5.1

14.23、内存

名称	功能说明	最低版本
onMemoryWarning	监听内存不足告警事件	1.5.1
offMemoryWarning	移除内存不足告警事件的监听函数	1.5.1

14.24、无障碍

名称	功能说明	最低版本
checkIsOpenAccessibility	检测是否开启视觉无障碍功能	1.5.1

15、WXML

15.1、WXML 概要

名称	功能说明	最低版本
createSelectorQuery	返回一个 SelectorQuery 对象实例	1.5.0
createIntersectionObserver	创建并返回一个 IntersectionObserver 对象实例	1.5.0

15.2、IntersectionObserve

名称	功能说明	最低版本
IntersectionObserve.relativeTo	使用选择器指定一个节点，作为参照区域之一	1.5.0
IntersectionObserve.relativeToViewport	指定页面显示区域作为参照区域之一	1.5.0
IntersectionObserve.disconnect	停止监听，回调函数将不再触发	1.5.0
IntersectionObserve.observe	指定目标节点并开始监听相交状态变化情况	1.5.0

15.3、NodesRef

名称	功能说明	最低版本
NodesRef.fields	获取节点的相关信息	1.5.0
NodesRef.boundingClientRect	添加节点的布局位置的查询请求	1.5.0
NodesRef.scrollOffset	添加节点的滚动位置查询请求	1.5.0
NodesRef.context	添加节点的 Context 对象查询请求	1.5.0
NodesRef.node	获取 Node 节点实例	1.5.0

15.4、SelectorQuery

名称	功能说明	最低版本
SelectorQuery.exec	执行所有的请求	1.5.0
SelectorQuery.in	将选择器的选取范围更改为自定义组件 component 内	1.5.0

SelectorQuery.select	在当前页面下选择第一个匹配选择器 selector 的节点	1.5.0
SelectorQuery.selectAll	在当前页面下选择匹配选择器 selector 的所有节点	1.5.0
SelectorQuery.selectViewport	选择显示区域	1.5.0

15.5、meidaQueryObserver

名称	功能说明	最低版本
MediaQueryObserver.disconnect	停止监听	1.5.16
MediaQueryObserver.observe	开始监听页面 media query 变化情况	1.5.16

16、自定义API

名称	功能说明	最低版本
invokeNativePlugin	自定义API	1.5.0

基础

最近更新时间：2024-02-26 10:34:51

canIUse

> 该 API 使用方法为 Boolean wx.canIUse(string schema)

- **功能说明：**判断小程序的 API，回调，参数，组件等是否在当前版本可用。
- **参数及说明：**String schema
  - **使用** `${API}.${method}.${param}.${options}` **或者** `${component}.${attribute}.${option}` **方式来调用。**
  - **参数说明：**

参数值	说明
${API}	代表 API 名字
${method}	代表调用方式，有效值为 return，success，object，callback
${param}	代表参数或者返回值
${options}	代表参数的可选值
${component}	代表组件名字
${attribute}	代表组件属性
${option}	代表组件属性的可选值

- **返回值：**Boolean，表达当前版本是否可用。
- **示例代码：**

```
wx.canIUse('openBluetoothAdapter')
wx.canIUse('getSystemInfoSync.return.screenWidth')
wx.canIUse('getSystemInfo.success.screenWidth')
wx.canIUse('showToast.object.image')
wx.canIUse('onCompassChange.callback.direction')
wx.canIUse('request.object.method.GET')

wx.canIUse('live-player')
wx.canIUse('text.selectable')
wx.canIUse('button.open-type.contact')
```

env

> 该 API 使用方法为 wx.env

- **功能说明：**环境变量。
- **参数及说明：**string USER_DATA_PATH，文件系统中的用户目录路径（本地路径）。

base64ToArrayBuffer

> 该 API 使用方法为 ArrayBuffer wx.base64ToArrayBuffer(string base64)

- **功能说明：**将 Base64 字符串转成 ArrayBuffer 对象。
- **参数及说明：**string base64，要转化成 ArrayBuffer 对象的 Base64 字符串。
- **返回值：**ArrayBuffer，ArrayBuffer 对象。
- **示例代码：**

```
const base64 = 'CxYh'
const arrayBuffer = wx.base64ToArrayBuffer(base64)
```

arrayBufferToBase64

> 该 API 使用方法为 string wx.arrayBufferToBase64(ArrayBuffer arrayBuffer)

- **功能说明：**将 ArrayBuffer 对象转成 Base64 字符串。
- **参数及说明：**ArrayBuffer arrayBuffer，要转换成 Base64 字符串的 ArrayBuffer 对象。
- **返回值：**string，Base64 字符串。
- **示例代码：**

```
const arrayBuffer = new Uint8Array([11, 22, 33])
const base64 = wx.arrayBufferToBase64(arrayBuffer)
```

系统

getSystemInfo

> 该 API 使用方法为 wx.getSystemInfo(Object object)

- **功能说明：**获取系统信息。
- **参数及说明：**Object Object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（无论成功与否都执行）

- **Object.success 的 res 回调结果**

属性	类型	说明
brand	String	设备品牌
model	String	设备型号
pixelRatio	Number	设备像素比
screenWidth	Number	屏幕宽度，单位 px
screenHeight	Number	屏幕高度，单位 px
windowWidth	Number	可使用窗口宽度，单位 px
windowHeight	Number	可使用窗口高度，单位 px
statusBarHeight	Number	状态栏的高度，单位 px
language	String	语言
version	String	版本号
system	String	操作系统及版本
platform	String	客户端平台，其合法值如下 • ios：iOS 客户端（包含 iPhone、iPad） • android：Android 客户端 • devtools：TCMPP 开发者工具
SDKVersion	String	客户端基础库版本

AppPlatform	String	App 平台
safeArea	Object	在竖屏正方向下的安全区域
theme	String	系统当前主题，取值为 light 或 dark，全局配置 "darkmode":true 时才能获取，否则为 undefined

- **res.safeArea 的结构：**

属性	类型	说明
left	Number	安全区域左上角横坐标
right	Number	安全区域右下角横坐标
top	Number	安全区域左上角纵坐标
bottom	Number	安全区域右下角纵坐标
width	Number	安全区域的宽度，单位逻辑像素
height	Number	安全区域的高度，单位逻辑像素

- **示例代码：**

```
wx.getSystemInfo({
  success(res) {
    console.log(res.model)
    console.log(res.pixelRatio)
    console.log(res.windowWidth)
    console.log(res.windowHeight)
    console.log(res.language)
    console.log(res.version)
    console.log(res.platform)
  }
})
```

```
try {
  const res = wx.getSystemInfoSync()
  console.log(res.model)
  console.log(res.pixelRatio)
  console.log(res.windowWidth)
  console.log(res.windowHeight)
  console.log(res.language)
```

```
  console.log(res.version)
  console.log(res.platform)
} catch (e) {
  // Do something when catch error
}
```

getSystemInfoSync

> 该 API 使用方法为 Object wx.getSystemInfoSync()

- **功能说明：** wx.getSystemInfo 的同步版本。
- **Object.success 的 res 回调结果**

属性	类型	说明
brand	String	设备品牌
model	String	设备型号
pixelRatio	Number	设备像素比
screenWidth	Number	屏幕宽度，单位 px
screenHeight	Number	屏幕高度，单位 px
windowWidth	Number	可使用窗口宽度，单位 px
windowHeight	Number	可使用窗口高度，单位 px
statusBarHeight	Number	状态栏的高度，单位 px
language	String	语言
version	String	版本号
system	String	操作系统及版本
platform	String	客户端平台，其合法值如下 • ios：iOS 客户端（包含 iPhone、iPad） • android：Android 客户端 • devtools：TCMPP 开发者工具
SDKVersion	String	客户端基础库版本
AppPlatform	String	App 平台

safeArea	Object	在竖屏正方向下的安全区域
theme	String	系统当前主题，取值为 light 或 dark ，全局配置 "darkmode":true 时才能获取，否则为 undefined

- **res.safeArea 的结构：**

属性	类型	说明
left	Number	安全区域左上角横坐标
right	Number	安全区域右下角横坐标
top	Number	安全区域左上角纵坐标
bottom	Number	安全区域右下上角纵坐标
width	Number	安全区域的宽度，单位逻辑像素
height	Number	安全区域的高度，单位逻辑像素

- **示例代码：**

```
wx.getSystemInfo({
  success(res) {
    console.log(res.model)
    console.log(res.pixelRatio)
    console.log(res.windowWidth)
    console.log(res.windowHeight)
    console.log(res.language)
    console.log(res.version)
    console.log(res.platform)
  }
})
```

```
try {
  const res = wx.getSystemInfoSync()
  console.log(res.model)
  console.log(res.pixelRatio)
  console.log(res.windowWidth)
  console.log(res.windowHeight)
  console.log(res.language)
  console.log(res.version)
  console.log(res.platform)
```

```
} catch (e) {
  // Do something when catch error
}
```

getSystemInfoAsync

> 该 API 使用方法为 wx.getSystemInfoAsync(Object object)

- **功能说明：**异步获取系统信息。需要一定的宿主客户端版本支持，在不支持的客户端上，会使用同步实现来返回。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

  - **object.success 回调函数参数：**Object res

属性	类型	说明
brand	string	设备品牌
model	string	设备型号。新机型刚推出一段时间会显示 unknown，我们会尽快进行适配
pixelRatio	number	设备像素比
screenWidth	number	屏幕宽度，单位 px
screenHeight	number	屏幕高度，单位 px
windowWidth	number	可使用窗口宽度，单位 px
windowHeight	number	可使用窗口高度，单位 px

statusBarHeight	number	任务状态栏高度，单位 px
language	string	应用内设置的语言
version	string	应用版本号
system	string	操作系统及版本
platform	string	客户端平台，其合法值为 • iOS：iOS 客户端（包括 iPhone、iPad） • Android：Android 客户端 • Windows：Windows 客户端 • Mac：MacOS 客户端 • devtools：IDE 开发者工具
fontSizeSetting	number	用户字体大小，单位 px
SDKVersion	string	客户端基础库版本
benchmarkLevel	number	设备性能等级（仅 Android）。取值为： • -2 或 0（该设备无法运行小游戏） • -1（性能未知） • >=1（设备性能值，该值越高，设备性能越好，目前最高不到50）
albumAuthorized	boolean	允许客户端使用相册的开关（仅 iOS 有效）
albumAuthorized	boolean	允许无需使用摄像头的开关
locationAuthorized	boolean	允许客户端使用定位的开关
microphoneAuthorized	boolean	允许客户端使用麦克风的开关
notificationAuthorized	boolean	允许客户端通知的开关

notificationAlertAuthorized	boolean	允许客户端通知带有提醒的开关（仅 iOS 有效）
notificationBadgeAuthorized	boolean	允许客户端通知带有标记的开关（仅 iOS 有效）
notificationSoundAuthorized	boolean	允许客户端通知带有声音的开关（仅 iOS 有效）
phoneCalendarAuthorized	boolean	允许客户端使用日历的开关
bluetoothEnabled	boolean	蓝牙的系统开关
locationEnabled	boolean	地理位置的系统开关
wifiEnabled	boolean	Wi-Fi 的系统开关
safeArea	object	在竖屏正方向下的安全区域。部分机型没有安全区域概念，也不会返回 safeArea 字段，开发者需自行兼容。返回值请参考下表 **safeArea 返回值**
locationReducedAccuracy	boolean	`true` 表示模糊定位，`false` 表示精确定位，仅 iOS 支持
theme	string	系统当前主题，全局配置 darkmode:true 时才能获取，否则为 undefined（不支持小游戏），其合法值为 • light：明亮 • dark：黑暗
host	object	当前小程序运行的宿主环境。结构为 string类型的 appid，表示客户端 App 对应的 appid
enableDebug	boolean	是否已打开调试。可通过右上角菜单或 wx.setEnableDebug 打开调试
deviceOrientation	string	设备方向： • portrait：竖屏 • landscape：横屏

- **safeArea 返回值**

结构属性	类型	说明

left	number	安全区域左上角横坐标
right	number	安全区域右下角横坐标
top	number	安全区域左上角纵坐标
bottom	number	安全区域右下角纵坐标
width	number	安全区域的宽度，单位逻辑像素
height	number	安全区域的高度，单位逻辑像素

- **示例代码：**

```
wx.getSystemInfoAsync({
  success (res) {
    console.log(res.model)
    console.log(res.pixelRatio)
    console.log(res.windowWidth)
    console.log(res.windowHeight)
    console.log(res.language)
    console.log(res.version)
    console.log(res.platform)
  }
})
```

getWindowInfo

> 该 API 使用方法为 Object wx.getWindowInfo()

- **功能说明：**获取窗口信息。
- **参数及说明：**Object。

属性	类型	说明
pixelRatio	number	设备像素比
screenWidth	number	屏幕宽度，单位 px
screenHeight	number	屏幕高度，单位 px
windowWidth	number	可使用窗口宽度，单位 px
windowHeight	number	可使用窗口高度，单位 px

statusBarHeight	number	状态栏的高度，单位 px
safeArea	object	在竖屏正方向下的安全区域。部分机型没有安全区域概念，也不会返回 safeArea 字段，开发者需自行兼容
screenTop	number	窗口上边缘的 y 值

safeArea 返回值

结构属性	类型	说明
left	number	安全区域左上角横坐标
right	number	安全区域右下角横坐标
top	number	安全区域左上角纵坐标
bottom	number	安全区域右下角纵坐标
width	number	安全区域的宽度，单位逻辑像素
height	number	安全区域的高度，单位逻辑像素

- **示例代码：**

```
const windowInfo = wx.getWindowInfo()

console.log(windowInfo.pixelRatio)
console.log(windowInfo.screenWidth)
console.log(windowInfo.screenHeight)
console.log(windowInfo.windowWidth)
console.log(windowInfo.windowHeight)
console.log(windowInfo.statusBarHeight)
console.log(windowInfo.safeArea)
console.log(windowInfo.screenTop)
```

getSystemSetting

> 该 API 使用方法为 Object wx.getSystemSetting()

- **功能说明：**获取设备设置。
- **参数及说明：**Object。

属性	类型	说明

bluetoothEnabled	boolean	蓝牙的系统开关
locationEnabled	boolean	地理位置的系统开关
wifiEnabled	boolean	Wi-Fi 的系统开关
deviceOrientation	string	设备方向，其合法值为： • portrait：竖屏 • landscape：横屏

- **示例代码：**

```
const systemSetting = wx.getSystemSetting()

console.log(systemSetting.bluetoothEnabled)
console.log(systemSetting.deviceOrientation)
console.log(systemSetting.locationEnabled)
console.log(systemSetting.wifiEnabled)
```

openSystemBluetoothSetting

> 该 API 使用方法为 wx.openSystemBluetoothSetting(Object object)

- **功能说明：**跳转系统蓝牙设置页。仅支持 Android 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.openSystemBluetoothSetting({
  success (res) {
    console.log(res)
```

```
  }
})
```

openAppAuthorizeSetting

> 该 API 使用方法为 wx.openAppAuthorizeSetting(Object object)

- **功能说明：**跳转系统宿主客户端授权管理页。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.openAppAuthorizeSetting({
  success (res) {
    console.log(res)
  }
})
```

getRendererUserAgent

> 该 API 使用方法为 Promise wx.getRendererUserAgent(Object object)

- **功能说明：**获取 Webview 小程序的 UserAgent。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数

complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**string userAgent，UserAgent。
- **返回值：**Promise.<string>。
- **示例代码：**

```
wx.getRendererUserAgent().then(userAgent => console.log(userAgent))
wx.getRendererUserAgent({
  success(res) { console.log(res.userAgent) }
})
```

getDeviceInfo

> 该 API 使用方法为 Object wx.getDeviceInfo()

- **功能说明：**获取设备基础信息。
- **返回值：**Object。

属性	类型	说明
abi	string	宿主 App 二进制接口类型（仅 Android 支持）
deviceAbi	string	设备二进制接口类型（仅 Android 支持）
benchmarkLevel	number	设备性能等级（仅 Android 支持），合法值为： • -2 或 0：该设备无法运行小游戏 • -1：性能未知 • >=1：设备性能值，该值越高，设备性能越好，目前最高不到50
brand	string	设备品牌
model	string	设备型号。新机型刚推出一段时间会显示 unknown，我们会尽快进行适配
system	string	操作系统及版本
platform	string	客户端平台
cpuType	string	设备 CPU 型号（仅 Android 支持）GPU 型号可通过 WebGLRenderingContext.getExtension('WEBGL_debug_renderer_info') 来获取
menoryS	string	设备内存大小，单位为 MB

ize	

- **示例代码：**

```
const deviceInfo = wx.getDeviceInfo()

console.log(deviceInfo.abi)
console.log(deviceInfo.benchmarkLevel)
console.log(deviceInfo.brand)
console.log(deviceInfo.model)
console.log(deviceInfo.platform)
console.log(deviceInfo.system)
```

getAppBaseInfo

> 该 API 使用方法为 Object wx.getAppBaseInfo()

- **功能说明：**获取宿主客户端基础信息。
- **参数及说明：**Object。

属性	类型	说明
SDKVersion	string	宿主客户端基础库版本
enableDebug	boolean	是否已打开调试。可通过右上角菜单或 wx.setEnableDebug 打开调试
host	Object	当前小程序运行的宿主环境。 结构属性为 string 类型的 appId，用于宿主客户端（第三方 App）对应的 appId（当小程序运行在第三方 App 环境时才返回）
language	string	宿主客户端设置的语言
version	string	宿主客户端版本号
theme	string	系统当前主题，取值为`light：明亮模式` 或 `dark：暗黑模式`，全局配置 `"darkmode":true`时才能获取，否则为 undefined（不支持小游戏）

- **示例代码：**

```
const appBaseInfo = wx.getAppBaseInfo()
```

```
console.log(appBaseInfo.SDKVersion)
console.log(appBaseInfo.enableDebug)
console.log(appBaseInfo.host)
console.log(appBaseInfo.language)
console.log(appBaseInfo.version)
console.log(appBaseInfo.theme)
```

getAppAuthorizeSetting

> 该 API 使用方法为 Object wx.getAppAuthorizeSetting()

- **功能说明：**获取宿主客户端授权设置。
- **参数及说明：**Object。

属性	类型	说明
albumAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端使用相册的开关（仅 iOS 有效）
bluetoothAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端使用蓝牙的开关（仅 iOS 有效）
cameraAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端使用摄像头的开关
locationAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端使用定位的开关
locationReducedAccuracy	boolean	定位准确度。true 表示模糊定位，false 表示精确定位（仅 iOS 有效）
microphoneAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端使用麦克风的开关
notificationAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端通知的开关
notificationAlertAuthorized	'authorized'/'denied'/'not	允许宿主客户端通知带有提醒的开关（仅 iOS 有效）

	determined'	
notificationBadgeAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端通知带有标记的开关（仅 iOS 有效）
notificationSoundAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端通知带有声音的开关（仅 iOS 有效）
phoneCalendarAuthorized	'authorized'/'denied'/'not determined'	允许宿主客户端读写日历的开关

- **示例代码：**

```
const appAuthorizeSetting = wx.getAppAuthorizeSetting()

console.log(appAuthorizeSetting.albumAuthorized)
console.log(appAuthorizeSetting.bluetoothAuthorized)
console.log(appAuthorizeSetting.cameraAuthorized)
console.log(appAuthorizeSetting.locationAuthorized)
console.log(appAuthorizeSetting.locationReducedAccuracy)
console.log(appAuthorizeSetting.microphoneAuthorized)
console.log(appAuthorizeSetting.notificationAlertAuthorized)
console.log(appAuthorizeSetting.notificationAuthorized)
console.log(appAuthorizeSetting.notificationBadgeAuthorized)
console.log(appAuthorizeSetting.notificationSoundAuthorized)
console.log(appAuthorizeSetting.phoneCalendarAuthorized)
```

更新

getUpdateManager

> 该 API 使用方法为 UpdateManager wx.getUpdateManager

- **功能说明**：获取全局唯一的版本更新管理器，用于管理小程序更新，使用方法为 UpdateManager wx.getUpdateManager。
- **返回值**：UpdateManager，更新管理器对象。

UpdateManager

.applyUpdate

该 方法 使用方式为 UpdateManager.applyUpdate()

- **功能说明：**强制小程序重启并使用新版本。在小程序新版本下载完成后（即收到 `onUpdateReady` 回调）调用。

.onCheckForUpdate

该 方法 使用方式为 UpdateManager.onCheckForUpdate(function listener)

- **功能说明：**监听向后台请求检查更新结果事件。在小程序冷启动时自动检查更新，不需由开发者主动触发。
- **参数及说明：**function callback，向后台请求检查更新结果事件的回调函数。

属性	类型	说明
hasUpdate	Boolean	是否有新版本

.onUpdateFailed

该 方法 使用方式为 UpdateManager.onUpdateFailed(function listener)

- **功能说明：**监听小程序有版本更新事件。客户端主动触发下载（无需开发者触发），下载成功后回调。
- **参数及说明：**function callback，小程序更新失败事件的回调函数。

.onUpdateReady

该 方法 使用方式为 UpdateManager.onUpdateReady(function listener)

- **功能说明：**监听小程序更新失败事件。小程序有新版本，客户端主动触发下载（无需开发者触发），下载失败（可能是网络原因等）后回调。
- **参数及说明：**function callback，小程序有新版本更新事件的回调函数。
- **示例代码：**

```
const updateManager = wx.getUpdateManager()

updateManager.onCheckForUpdate(function (res) {
  // 请求完新版本信息的回调
  console.log(res.hasUpdate)
})

updateManager.onUpdateReady(function () {
  wx.showModal({
```

```
    title: '更新提示',
    content: '新版本已经准备好，是否重启应用？',
    success(res) {
      if (res.confirm) {
        // 新的版本已经下载好，调用 applyUpdate 应用新版本并重启
        updateManager.applyUpdate()
      }
    }
  })
})

updateManager.onUpdateFailed(function () {
  // 新版本下载失败
})
```

小程序

生命周期

getLaunchOptionsSync

> 该 API 使用方法为 Object wx.getLaunchOptionsSync()

注意：

部分版本在无 `referrerInfo` 的时候会返回 `undefined`，建议使用 `options.referrerInfo &amp;&amp; options.referrerInfo.appId` 进行判断。

- **功能说明：**获取小程序启动时的参数 。
- **返回值：**Object，启动参数。

属性	类型	说明
path	String	启动小程序的路径
scene	Number	启动小程序的 场景值
query	Object	启动小程序的 query 参数
referrerInfo	Object	来源信息，从另一个小程序、公众号或 App 进入小程序时返回。否则返回{}
forwardMaterials	Array.<Object>	打开的文件信息数组，只有从聊天素材场景

		打开（scene 为1173）才会携带该参数

- referrerInfo 的结构

属性	类型	说明
appId	String	来源小程序、公众号或 App 的 appId
extraData	Object	来源小程序传过来的数据，scene=1037或1038时支持

- forwardMaterials 的结构

属性	类型	说明
type	string	文件的 mimetype 类型
name	Object	文件名
path	string	文件路径
size	number	文件大小

- 返回有效 `referrerInfo` 的场景

属性	类型	appId 含义
1020	公众号 profile 页相关小程序列表	来源公众号
1035	公众号自定义菜单	来源公众号
1036	App 分享消息卡片	来源 App
1037	小程序打开小程序	来源小程序
1038	从另一个小程序返回	来源小程序
1043	公众号模板消息	来源公众号

getEnterOptionsSync

> 该 API 使用方法为 Object wx.getEnterOptionsSync()

> ⚠ **注意：**

> 部分版本在无 `referrerInfo` 的时候会返回 `undefined`，建议使用 `options.referrerInfo &amp;&amp; options.referrerInfo.appId` 进行判断。

- **功能说明：**获取本次小程序启动时的参数。如果当前是冷启动，则返回值与 App.onLaunch 的回调参数一致；如果当前是热启动，则返回值与 App.onShow 一致。
- **返回值：**Object

属性	类型	说明
path	string	启动小程序的路径（代码包路径）
scene	number	启动小程序的 场景值
query	Object	启动小程序的 query 参数
shareTicket	string	详细可参考微信文档 转发
referrInfo	Object	来源信息。从另一个小程序、公众号或 App 进入小程序时返回。否则返回{}，具体参见表格前文的注意
forwardMaterials	Array.<Object>	打开的文件信息数组，只有从聊天素材场景打开（scene为1173）才会携带该参数
chatType	number	从宿主客户端群聊/单聊打开小程序时，chatType 表示具体宿主客户端里的群聊/单聊类型，合法值为： • 1：宿主客户端联系人单聊 • 2：企业宿主客户端联系人单聊 • 3：普通宿主客户端群聊 • 4：企业宿主客户端互通群聊

  - referrerInfo 的结构

属性	类型	说明
appId	String	来源小程序、公众号或 App 的 appId
extraData	Object	来源小程序传过来的数据，scene=1037或1038时支持

  - forwardMaterials 的结构

属性	类型	说明

type	String	文件的 mimetype 类型
name	Object	文件名
path	String	文件路径（如果是 webview 则是 url）
size	Number	文件大小

- **返回有效 referrerInfo 的场景**

属性	类型	appId 含义
1020	公众号 profile 页相关小程序列表	来源公众号
1035	公众号自定义菜单	来源公众号
1036	App 分享消息卡片	来源 App
1037	小程序打开小程序	来源小程序
1038	从另一个小程序返回	来源小程序
1043	公众号模板消息	来源公众号

应用级事件

onError

> 该 API 使用方法为 wx.onError(function callback)

- **功能说明：**监听小程序错误事件。如脚本错误或 API 调用报错等。
- **参数及说明：**function callback，小程序错误事件的回调函数。
- **返回参数：**string error，错误信息，包含堆栈。

offError

> 该 API 使用方法为 wx.offError(function callback)

- **功能说明：**取消监听小程序错误事件。
- **参数及说明：**function callback，小程序错误事件的回调函数。

onThemeChange

> 该 API 使用方法为 wx.onThemeChange(function listener)

> **说明：**
> 只有在全局配置 darkmode：true 时才会触发此事件。

- **功能说明：**监听系统主题改变事件。
- **参数及说明：**function listener，系统主题改变事件的监听函数。
- **返回参数：**Object res

属性	合法值及说明	类型	说明
theme	dark：深色主题 light：浅色主题	String	系统当前的主题，取值为 light 或 dar

offThemeChange

该 API 使用方法为 wx.offThemeChange(function listener)

- **功能说明：**移除系统主题改变事件的监听函数 。
- **参数及说明：**function listener，onThemeChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onThemeChange(listener)
wx.offThemeChange(listener) // 需传入与监听时同一个的函数对象
```

onPageNotFound

该 API 使用方法为 wx.onPageNotFound(function listener)

> **注意：**
> - 开发者可以在回调中进行页面重定向，但必须在回调中同步处理，异步处理（例如 setTimeout 异步执行）无效。
> - 若开发者没有调用 wx.onPageNotFound 绑定监听，也没有声明 App.onPageNotFound，当跳转页面不存在时，将推入宿主客户端原生的页面不存在提示页面。
> - 如果回调中又重定向到另一个不存在的页面，将推入宿主客户端原生的页面不存在提示页面，并且不再第二次回调。

- **功能说明：**监听小程序要打开的页面不存在事件。该事件与 App.onPageNotFound 的回调时机一致。
- **参数及说明：**Object res 参数，function listener，小程序要打开的页面不存在事件的监听函数。

属性	类型	说明
path	string	不存在页面的路径 (代码包路径)
query	Object	打开不存在页面的 query 参数
isEntryPage	boolean	是否本次启动的首个页面（例如从分享等入口进来，首个页面是开发者配置的分享页面）

offPageNotFound

该 API 使用方法为 wx.offPageNotFound(function listener)

- **功能说明：**移除小程序要打开的页面不存在事件的监听函数。
- **参数及说明：**function listener，onPageNotFound 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

wx.onPageNotFound(listener)
wx.offPageNotFound(listener) // 需传入与监听时同一个的函数对象
```

onAppShow

该 API 使用方法为 wx.onAppShow(function listener)

> ⚠ **注意：**
> 部分版本在无 referrerInfo 的时候会返回 undefined，建议使用 `options.referrerInfo &amp;&amp; options.referrerInfo.appId` 进行判断。

- **功能说明：**监听小程序切前台事件。该事件与 App.onShow 的回调参数一致。
- **参数及说明：**Object res 参数，function listener，小程序切前台事件的监听函数。

属性	类型	说明
path	string	启动小程序的路径（代码包路径）
scene	number	启动小程序的场景值
query	Object	启动小程序的 query 参数

shareTicket	string	shareTicket
referrerInfo	Object	来源信息。从另一个小程序、公众号或 App 进入小程序时返回。否则返回{}，具体参见表格前文的注意
forwardMaterials	Array.<Object>	打开的文件信息数组，只有从聊天素材场景打开（scene为1173）才会携带该参数
chatType	number	从宿主客户端群聊/单聊打开小程序时，chatType 表示具体宿主客户端内群聊/单聊类型，合法值为： • 1：宿主客户端内联系人单聊 • 2：宿主客户端内企业联系人单聊 • 3：宿主客户端内普通群聊 • 4：宿主客户端内企业互通群聊

- referrerInfo 的结构

属性	类型	说明
appId	string	来源小程序、公众号或 App 的 appId
extraData	Object	来源小程序传过来的数据，scene=1037或1038时支持

- forwardMaterials 的结构

属性	类型	说明
type	String	文件的 mimetype 类型
name	Object	文件名
path	String	文件路径（如果是 webview 则是 url）
size	Number	文件大小

- **返回有效 referrerInfo 的场景**

场景值	场景	appId含义
1020	公众号 profile 页相关小程序列表	来源公众号
1035	公众号自定义菜单	来源公众号
1036	App 分享消息卡片	来源 App
1037	小程序打开小程序	来源小程序

1038	从另一个小程序返回	来源小程序
1043	公众号模板消息	来源公众号

offAppShow

该 API 使用方法为 wx.offAppShow(function listener)

- **功能说明：**移除小程序切前台事件的监听函数。
- **参数及说明：**function listener，onAppShow 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

wx.onAppShow(listener)
wx.offAppShow(listener) // 需传入与监听时同一个的函数对象
```

onAppHide

该 API 使用方法为 wx.onAppHide(function listener)

- **功能说明：**监听小程序切后台事件。该事件与 App.onHide 的回调时机一致。
- **参数及说明：**function listener，小程序切后台事件的监听函数。

offAppHide

该 API 使用方法为 wx.offAppHide(function listener)

- **功能说明：**移除小程序切前台事件的监听函数。
- **参数及说明：**function listener，onAppHide 传入的监听函数。不传此参数则移除所有监听函数。

```
const listener = function (res) { console.log(res) }

wx.onAppHide(listener)
wx.offAppHide(listener) // 需传入与监听时同一个的函数对象
```

onUnhandledRejection

该 API 使用方法为 wx.onUnhandledRejection(function listener)

注意：

> 所有的 unhandledRejection 都可以被这一监听捕获，但只有 Error 类型的才会在小程序后台触发报警。

- **功能说明：**监听未处理的 Promise 拒绝事件。
- **参数及说明：**Object res 参数，function listener，未处理的 Promise 拒绝事件的监听函数。

属性	类型	说明
reason	string	拒绝原因，一般是一个 Error 对象
promise	Promise.<any>	被拒绝的 Promise 对象

offUnhandledRejection

> 该 API 使用方法为 wx.offUnhandledRejection(function listener)

- **功能说明：**移除未处理的 Promise 拒绝事件的监听函数 。
- **参数及说明：**function listener，onUnhandledRejection 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

wx.onUnhandledRejection(listener)
wx.offUnhandledRejection(listener) // 需传入与监听时同一个的函数对象
```

调试

setEnableDebug

> 该 API 使用方法为 wx.setEnableDebug(Object object)

- **功能说明：**设置是否打开调试开关。此开关对正式版也能生效。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

enableDebug	boolean	–	是	是否打开调试
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
// 打开调试
wx.setEnableDebug({
  enableDebug: true
})

// 关闭调试
wx.setEnableDebug({
  enableDebug: false
})
```

> **说明：**
> 在正式版打开调试还可以先在开发版或体验版打开调试，再切到正式版就能看到 vConsol。

getRealtimeLogManager

该 API 使用方法为 RealtimeLogManager wx.getRealtimeLogManager()

- **功能说明：**获取实时日志管理器对象。
- **返回值：**RealtimeLogManager 。
- **示例代码：**

```
// 小程序端
const logger = wx.getRealtimeLogManager()
logger.info({str: 'hello world'}, 'info log', 100, [1, 2, 3])
logger.error({str: 'hello world'}, 'error log', 100, [1, 2, 3])
logger.warn({str: 'hello world'}, 'warn log', 100, [1, 2, 3])

// 插件端，基础库 2.16.0 版本后支持，只允许采用 key-value 的新格式上报
const logManager = wx.getRealtimeLogManager()
```

```
const logger = logManager.tag('plugin-log1')
logger.info('key1', 'value1')
logger.error('key2', {str: 'value2'})
logger.warn('key3', 'value3')
```

getLogManager

> 该 API 使用方法为 LogManager wx.getLogManager(Object object)

- **功能说明：**获取日志管理器对象。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
level	number	0	否	取值为0/1，取值为0表示是否会把 App 、 Page 的生命周期函数和命名空间下的函数调用写入日志，取值为1则不会。默认值是 0

- **返回值：**LogManager 。
- **示例代码：**

```
const logger = wx.getLogManager({level: 1})
logger.log({str: 'hello world'}, 'basic log', 100, [1, 2, 3])
logger.info({str: 'hello world'}, 'info log', 100, [1, 2, 3])
logger.debug({str: 'hello world'}, 'debug log', 100, [1, 2, 3])
logger.warn({str: 'hello world'}, 'warn log', 100, [1, 2, 3])
```

console

console 是一个可以直接访问的全局对象，其具体方法集如下。可以在 IDE 调试面板中打印日志，在客户端中利用 vConsole 输出日志。

debug

> 该 方法 使用方式为 console.debug()

- **功能说明：**向调试面板中打印 debug 日志。
- **参数及说明：**any ... args，日志内容，可以有任意多个。

error

> 该 方法 使用方式为 console.error()

- **功能说明：**向调试面板中打印 error 日志。
- **参数及说明：**any ... args，日志内容，可以有任意多个。

group

> 该 方法 使用方式为 console.group()

> **说明：**
> 仅在工具中有效，在 vConsole 中为空函数实现。

- **功能说明：**在调试面板中创建一个新的分组。随后输出的内容都会被添加一个缩进，表示该内容属于当前分组。调用 console.groupEnd 之后分组结束。
- **参数及说明：**string label，分组标记，可选。

groupEnd

> 该 方法 使用方式为 console.groupEnd()

> **说明：**
> 仅在工具中有效，在 vConsole 中为空函数实现。

功能说明：结束由 console.group 创建的分组。

info

> 该 方法 使用方式为 console.info

- **功能说明：**向调试面板中打印 info 日志。
- **参数及说明：**any ... args，日志内容，可以有任意多个。

log

> 该 方法 使用方式为 console.log()

- **功能说明：**向调试面板中打印 log 日志。
- **参数及说明：**any ... args，日志内容，可以有任意多个。

warn

> 该 方法 使用方式为 console.warn()

> **说明：**
> - 由于 vConsole 功能有限，以及不同客户端对 console 方法的支持情况有差异，建议开发者

在小程序中只使用本文档中提供的方法；
- 部分内容展示的限制请参见 可用性 – 调试 。

- **功能说明：**向调试面板中打印 warn 日志。
- **参数及说明：**any ... args，日志内容，可以有任意多个。

LogManager

日志管理器实例，可以通过 getLogManager 获取，具体方法集如下。

debug

该 方法 使用方式为 LogManager.debug()

- **功能说明：**写 debug 日志。
- **参数及说明：**Object|Array.\ |number|string ... args，日志内容，可以有任意多个。每次调用的参数的总大小不超过100Kb。

info

该 方法 使用方式为 LogManager.info()

- **功能说明：**写 info 日志。
- **参数及说明：**Object|Array.\ |number|string ...args，日志内容，可以有任意多个。每次调用的参数的总大小不超过100Kb。

log

该 方法 使用方式为 LogManager.log()

- **功能说明：**写 log 日志。
- **参数及说明：**Object|Array.\ |number|string ...args，日志内容，可以有任意多个。每次调用的参数的总大小不超过100Kb。

warn

该 方法 使用方式为 LogManager.warn()

说明：
- 最多保存5M的日志内容，超过5M后，旧的日志内容会被删除；
- 用户可以通过使用 `<Button>` 组件的 `open-type="feedback"` 来上传打印的日志。开发者可以通过小程序管理后台左侧菜单“反馈管理”页面查看；
- 基础库默认会把 `App` 、 `Page` 的生命周期函数和命名空间下的函数调用写入日志。

- **功能说明：**写 warn 日志。
- **参数及说明：**Object|Array.\ |number|string ...args，日志内容，可以有任意多个。每次调用的参数的总大小不超过100Kb。

RealtimeLogManager

addFilterMsg

该 方法 使用方式为 RealtimeLogManager.addFilterMsg(string msg)

- **功能说明：**添加过滤关键字，暂不支持在插件使用。
- **参数及说明：**string msg，是setFilterMsg的添加接口。用于设置多个过滤关键字。

error

该 方法 使用方式为 RealtimeLogManager.error()

- **功能说明：**写 error 日志，暂不支持在插件使用。
- **参数及说明：**Object|Array.<any>|number|string ...args，日志内容，可以有任意多个。每次调用的参数的总大小不超过5Kb。

getCurrentState

该 方法 使用方式为 Object RealtimeLogManager.getCurrentState()

> ⚠ **注意：**
> 基础库内部在对日志进行上报时会补充一些结构化数据，如果遇到上报溢出的情况也会补充警告日志，所以此方法获取到的当前占用信息会比预期的大一些。

- **功能说明：**实时日志会将一定时间间隔内缓存的日志聚合上报，如果该时间内缓存的内容超出限制，则会被丢弃。此方法可以获取当前缓存剩余空间。
- **返回值：**Object

属性	类型	说明
size	number	当前缓存中已使用空间，以字节为单位
maxSize	number	当前缓存最大可用空间，以字节为单位
logCount	number	当前缓存中的日志条数
maxLogCount	number	当前缓存中最大可存日志条数

in

> 该 方法 使用方式为 RealtimeLogManager.in(Page pageInstance)

- **功能说明：**设置实时日志page参数所在的页面，暂不支持在插件使用。
- **参数及说明：**Page pageInstance，page 实例。

info

> 该 方法 使用方式为 RealtimeLogManager.info()

- **功能说明：**写 info 日志，暂不支持在插件使用。
- **参数及说明：**Object|Array.<any>|number|string ...args,日志内容，可以有任意多个。每次调用的参数的总大小不超过5Kb。

setFilterMsg

> 该 方法 使用方式为 RealtimeLogManagersetFilterMsg(string msg)

- **功能说明：**设置过滤关键字，暂不支持在插件使用。
- **参数及说明：**string msg，过滤关键字，最多不超过1Kb，可以在小程序管理后台根据设置的内容搜索得到对应的日志。

tag

> 该 方法 使用方式为 RealtimeTagLogManager RealtimeLogManager.tag(string tagName)

- **功能说明：**获取给定标签的日志管理器实例，目前只支持在插件使用。
- **参数及说明：**string tagName，标签名。
- **返回值：**RealtimeTagLogManager

warn

> 该 方法 使用方式为 RealtimeLogManager.warn()

- **功能说明：**写 warn 日志，暂不支持在插件使用。
- **参数及说明：**Object|Array.<any>|number|string ...args，日志内容，可以有任意多个。每次调用的参数的总大小不超过5Kb。

RealtimeTagLogManager

addFilterMsg

该 方法 使用方式为 RealtimeTagLogManager.addFilterMsg(string msg)

- **功能说明：**添加过滤关键字。
- **参数及说明：**string msg，是setFilterMsg的添加接口。用于设置多个过滤关键字。

error

该 方法 使用方式为 RealtimeTagLogManager.error(string key, Object]Array.<any> number string value)

- **功能说明：**写 error 日志。
- **参数及说明：**
  - string key，日志的 key；
  - Object|Array.<any>|number|string value，日志的值，每次调用的参数的总大小不超过 5Kb。

info

该 方法 使用方式为 RealtimeTagLogManager.info(string key, Object Array.<any> number string value)

- **功能说明：**写 info 日志。
- **参数及说明：**
  - string key，日志的 key；
  - Object|Array.<any>|number|string value，日志的值，每次调用的参数的总大小不超过 5Kb。

setFilterMsg

该 方法 使用方式为 RealtimeTagLogManager.setFilterMsg(string msg)

- **功能说明：**设置过滤关键字。
- **参数及说明：**string msg，过滤关键字，最多不超过1Kb，可以在小程序管理后台根据设置的内容搜索得到对应的日志。

warn

该 方法 使用方式为 RealtimeTagLogManager.warn(string key, ObjectJArray.< any> number string value)

- **功能说明：**写 warn 日志。
- **参数及说明：**
  - string key，日志的 key；
  - Object|Array.<any>|number|string value，日志的值，每次调用的参数的总大小不超过

5Kb。

路由

最近更新时间：2023-10-20 15:19:15

switchTab

> 该 API 使用方法为 wx.switchTab(Object object)

- **功能说明：**跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	string	-	是	需要跳转的 tabBar 页面的路径 (代码包路径) (需在 app.json 的 tabBar 字段定义的页面) ，路径后不能带参数
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用成功的回调函数
complete	function	-	否	接口调用结束的回调函数 (调用成功、失败都会执行)

- **示例代码：**

```
// app.json
{
  "tabBar": {
    "list": [{
      "pagePath": "index",
      "text": "首页"
    },{
      "pagePath": "other",
      "text": "其他"
    }]
  }
}
```

```
wx.switchTab({
  url: '/index'
```

```
})
```

reLaunch

> 该 API 使用方法为 wx.reLaunch(Object object)

- **功能说明：**关闭所有页面，打开到应用内的某个页面。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	string	–	是	需要跳转的应用内非 tabBar 的页面的路径 (代码包路径)，路径后可以带参数。参数与路径之间使用 `?` 分隔，参数键与参数值用 `=` 相连，不同参数用 `&` 分隔；如 `'path?key=value&key2=value2'`
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.redirectTo({
  url: 'test?id=1'
})
```

redirectTo

> 该 API 使用方法为 wx.redirectTo(Object object)

- **功能说明：**关闭所有页面，打开到应用内的某个页面。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	string	–	是	需要跳转的应用内页面路径 (代码包路径)，路径后可以带参

				数。参数与路径之间使用 ? 分隔，参数键与参数值用=相连，不同参数用 & 分隔；如 'path?key=value&key2=value2'
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.reLaunch({
  url: 'test?id=1'
})
```

```
// test
Page({
  onLoad (option) {
    console.log(option.query)
  }
})
```

navigateTo

> 该 API 使用方法为 wx.navigateTo(Object object)

- **功能说明：**保留当前页面，跳转到应用内的某个页面，但是不能跳到 tabbar 页面。使用 wx.navigateBack 可以返回到原页面，小程序中页面栈最多十层。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	string	–	是	需要跳转的应用内非 tabBar 的页面的路径 (代码包路径), 路径后可以带参数。参数与路径之间使用 ? 分隔，参数键与参数值用 = 相连，不同参数用 & 分隔；如 'path?key=value&key2=value2'

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.navigateTo({
  url: 'test?id=1',
})


//test.js
Page({
  onLoad: (option){
     console.log(option.query)
   }
})
```

navigateBack

> 该 API 使用方法为 wx.navigateBack(Object object)

- **功能说明：**关闭当前页面，返回上一页面或多级页面。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
delta	number	1	否	返回的页面数，如果 delta 大于现有页面数，则返回到首页
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
// 注意：调用 navigateTo 跳转时，调用该方法的页面会被加入堆栈，而 redirectTo 方法则不会。见下方示例代码

// 此处是 A 页面
wx.navigateTo({
  url: 'B?id=1'
})

// 此处是 B 页面
wx.navigateTo({
  url: 'C?id=1'
})

// 在 C 页面内 navigateBack，将返回 A 页面
wx.navigateBack({
  delta: 2
})
```

EventChannel

页面间事件通信通道。

EventChannel.emit

> 该 方法 使用方式为 EventChannel.emit(string eventName, any args)

- **功能说明：**触发一个事件。
- **参数及说明：**string eventName，事件名称；any args，事件参数。

EventChannel.off

> 该 方法 使用方式为 EventChannel.off(string eventName, EventCallback fn)

- **功能说明：**取消监听一个事件。给出第二个参数时，只取消给出的监听函数，否则取消所有监听函数，使用方法为 EventChannel.off(string eventName, function fn)。
- **参数及说明：**string eventName，事件名称；function fn，事件监听函数；any args，触发事件参数。

EventChannel.on

> 该 方法 使用方式为 EventChannel.on(string eventName, EventCallback fn)

- **功能说明：**持续监听一个事件，使用方法为 EventChannel.on(string eventName, function fn)。

- **参数及说明：**string eventName，事件名称；function fn，事件监听函数；any args，触发事件参数。

EventChannel.once

> 该 方法 使用方式为 EventChannel.once(string eventName, EventCallback fn)

- **功能说明：**监听一个事件一次，触发后失效，使用方法为 EventChannel.once(string eventName, function fn)。
- **参数及说明：**string eventName，事件名称；function fn，事件监听函数；any args，触发事件参数。

跳转

最近更新时间：2023-10-20 15:19:15

exitMiniProgram

> 该 API 使用方法为 wx.exitMiniProgram(Object object)

- **功能说明：**退出当前小程序。必须有单击行为才能调用成功。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

navigateToMiniProgram

> 该 API 使用方法为 wx.navigateToMiniProgram(Object object)

- **功能说明：**打开另一个小程序。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
appId	string	–	是	要打开的小程序 appId
path	string	–	否	打开的页面路径，如果为空则打开首页
extraData	object	–	否	需要传递给目标小程序的数据，目标小程序可在 `App.onLaunch`，`App.onShow` 中获取到这份数据
envVersion	string	release	否	要打开的小程序版本。仅在当前小程序为开发版或体验版时此参数有效。如果当前小程序是正式版，则打开的小程序必定是正式版，其合法值如下： • develop：开发版 • trial：体验版

				• release：正式版
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

使用限制

需要用户触发跳转

若用户未单击小程序页面任意位置，则开发者将无法调用此接口自动跳转至其他小程序。

需要用户确认跳转

在跳转至其他小程序前，将统一增加弹窗，询问是否跳转，用户确认后才可以跳转其他小程序。如果用户单击**取消**，则回调 `fail cancel` 。

每个小程序可跳转的其他小程序数量限制为不超过10个

开发者提交新版小程序代码时，如使用了跳转其他小程序功能，则需要在代码配置中声明将要跳转的小程序名单，限定不超过10个，否则将无法通过审核。该名单可在发布新版时更新，不支持动态修改。配置方法详见 配置 。调用此接口时，所跳转的 appId 必须在配置列表中，否则回调 `fail appId "${appId}" is not in navigateToMiniProgramAppIdList` 。

示例代码：

```
wx.navigateToMiniProgram({
  appId: '',
  path: 'page/index/index?id=123',
  extraData: {
    foo: 'bar'
  },
  envVersion: 'develop',
  success(res) {
    // 打开成功
  }
})
```

navigateBackMiniProgram

> 该 API 使用方法为 wx.navigateBackMiniProgram(Object object)

- **功能说明：**返回到上一个小程序。只有在当前小程序是被其他小程序打开时可以调用成功。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
extraData	Object	{}	否	需要返回给上一个小程序的数据，上一个小程序可在 App.onShow 中获取到这份数据
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.navigateBackMiniProgram({
  extraData: {
    foo: 'bar'
  },
  success(res) {
    // 返回成功
  }
})
```

转发

最近更新时间：2024-02-26 10:34:51

hideShareMenu

> 该 API 使用方法为 wx.hideShareMenu(Object object)

- **功能说明：**单击右上角菜单时隐藏 "分享给好友" 按钮。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
hideShareItems	Array.<string>	['qq', 'qzone', 'wechatFriends', 'wechatMoment']	否	• qq：控制是否隐藏"转发" • qzone：控制是否隐藏"分享到宿主客户端 qq 空间" • wechatFriends：控制是否隐藏"分享到宿主客户端微信朋友" • 'wechatMoment：控制是否隐藏"分享到宿主客户端微信朋友圈"，不带 hideShareItems 参数默认"转发"、"分享到宿主客户端 qq 空间"、"分享到宿主客户端微信朋友"、"控制是否控制隐藏"全隐藏
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.hideShareMenu({
  hideShareItems: ['qq', 'qzone', 'wechatFriends', 'wechatMoment']
})
```

showShareMenu

> 该 API 使用方法为 wx.showShareMenu(Object object)

- **功能说明：**单击右上角菜单时显示当前页面的"分享给好友"、"分享到宿主客户端 qq 空间"、"分享到宿主客户端微信好友"、"分享到宿主客户端微信朋友圈"按钮。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
showShareItems	Array.<string>	['qq', 'qzone', 'wechatFriends', 'wechatMoment']	否	请查看下面 object.showShareItems 参数说明
withShareTicket	boolean	false	否	是否使用带 shareTicket 的转发，Android 从基础库1.4.0版本、iOS 从基础库1.4.0版本开始支持
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.showShareItems 参数**

值	说明
'qq'	控制是否展示"转发"
'qzone'	控制是否展示"分享到宿主客户端 qq 空间"
'wechatFriends'	控制是否展示"宿主客户端微信好友"
'wechatMoment'	控制是否控制隐藏"分享到宿主客户端微信朋友圈"，不带 hideShareItems 参数默认"转发"、"分享到宿主客户端 qq 空间"、"分享到宿主客户端微信朋友"、"控制是否控制隐藏"全隐藏

- **示例代码：**

```
wx.showShareMenu({
  showShareItems: ['qq', 'qzone', 'wechatFriends', 'wechatMoment']
})
```

updateShareMenu

> 该 API 使用方法为 wx.updateShareMenu(Object object)

- **功能说明：**更新转发属性。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
withShareTicket	boolean	false	否	是否使用带 shareTicket 的转发
success	function	–	否	接口调用成功的回调函
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.updateShareMenu({
  withShareTicket: true
})
```

showShareImageMenu

> 该 API 使用方法为 wx.showShareImageMenu(Object object)

- **功能说明：**打开分享图片弹窗，可以将图片发送给朋友、收藏或下载。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
path	string	–	是	要分享的图片地址，必须为本地路径或临时路径
style	string	default	否	分享样式，可选 v2

needShowEntrance	string	false	否	分享的图片消息是否要带小程序入口 (仅部分小程序类目可用)
entrance	string	false	否	从消息小程序入口打开小程序的路径，如果当前页面允许分享给朋友，则默认为当前页面路径，否则默认为小程序首页
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.downloadFile({
  url: 'https://res.wx.qq.com/wxdoc/dist/assets/img/demo.ef5c5bef.jpg',
  success: (res) => {
    wx.showShareImageMenu({
      path: res.tempFilePath
    })
  }
})
```

onCopyUrl

> 该 API 使用方法为 wx.onCopyUrl(function listener)

- **功能说明：**监听用户点击右上角菜单的「复制链接」按钮时触发的事件。本接口为 Beta 版本，暂只在 Android 平台支持。
- **参数及说明：**
  - function listener，用户点击右上角菜单的「复制链接」按钮时触发的事件的监听函数。
  - Object res，

属性	类型	说明
query	string	用短链打开小程序时当前页面携带的查询字符串。小程序中使用时，应在进入页面时调用 wx.onCopyUrl 自定义 query，退出页面时调用 wx.offCopyUrl，防止影响其它页面

- **示例代码**

```
// 绑定分享参数
wx.onCopyUrl(() => {
  return { query: 'a=1&b=2' }
})

// 取消绑定分享参数
wx.offCopyUrl()
```

offCopyUrl

> 该 API 使用方法为 wx.offCopyUrl()

- **功能说明：**移除用户点击右上角菜单的「复制链接」按钮时触发的事件的全部监听函数。
- **示例代码**

```
// 绑定分享参数
wx.onCopyUrl(() => {
  return { query: 'a=1&b=2' }
})

// 取消绑定分享参数
wx.offCopyUrl()
```

界面
交互

最近更新时间：2023-10-20 15:19:16

hideLoading

> 该 API 使用方法为 wx.hideLoading(Object object)

- **功能说明：**隐藏 loading 提示框。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

hideToast

> 该 API 使用方法为 wx.hideToast(Object object)

- **功能描述：**隐藏消息提示框。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

showActionSheet

> 该 API 使用方法为 wx.showActionSheet(Object object)

> **注意：**
> - Android 6.7.2 以下版本，点击取消或蒙层时，回调 fail, errMsg 为 "fail cancel"；
> - Android 6.7.2 及以上版本 和 iOS 点击蒙层不会关闭模态弹窗，所以尽量避免使用「取消」分支中实现业务逻辑

- **功能描述：**显示操作菜单。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
itemList	array.<string>	–	是	按钮的文字数组，数组长度最大为6
itemColor	string	#000000	否	按钮的文字颜色
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
tapIndex	number	用户单击的按钮序号，从上到下的顺序，从0开始

- **示例代码**

```
wx.showActionSheet({
  itemList: ['A', 'B', 'C'],
  success (res) {
    console.log(res.tapIndex)
  },
  fail (res) {
    console.log(res.errMsg)
  }
})
```

showLoading

> 该 API 使用方法为 wx.showLoading(Object object)

- **功能说明：**显示 loading 提示框。需主动调用 wx.hideLoading 才能关闭提示框 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
title	string	–	是	提示的内容
mask	boolean	false	否	是否显示透明蒙层，防止触摸穿透
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.showLoading({
  title: '加载中',
})

setTimeout(function () {
  wx.hideLoading()
}, 2000)
```

> ⚠ **注意：**
>
> - wx.showLoading 和 wx.showToast 同时只能显示一个。
> - wx.showLoading 应与 wx.hideLoading 配对使用。

showModal

> 该 API 使用方法为 wx.showModal(Object object)

> ⚠ **注意：**
>
> - Android 6.7.2 以下版本，单击取消或蒙层时，回调 fail, errMsg 为 "fail cancel"。

- Android 6.7.2 及以上版本和 iOS 单击蒙层不会关闭模态弹窗，所以尽量避免使用**取消**分支中实现业务逻辑。

- **功能说明：**显示模态对话框。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
title	string	–	否	提示的标题
content	string	–	否	提示的内容
showCancel	boolean	true	否	是否显示取消按钮
cancelText	string	取消	否	取消按钮的文字，最多4个字符
cancelColor	string	#000000	否	取消按钮的文字颜色，必须是16进制格式的颜色字符串
confirmText	string	确定	否	确认按钮的文字，最多4个字符
confirmColor	string	#576B95	否	确认按钮的文字颜色，必须是16进制格式的颜色字符串
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
confirm	boolean	为 true 时，表示用户单击了确定按钮
cancel	boolean	为 true 时，表示用户单击了取消（用于 Android 系统区分单击蒙层关闭还是单击取消按钮关闭）

- **示例代码**

```
wx.showModal({
  title: '提示',
  content: '这是一个模态弹窗',
  success (res) {
    if (res.confirm) {
      console.log('用户点击确定')
    } else if (res.cancel) {
      console.log('用户点击取消')
    }
  }
})
```

showToast

> 该 API 使用方法为 wx.showToast(Object object)

- **功能说明：**显示模态对话框。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
title	string	–	是	提示的内容
icon	string	success	否	图标，其合法值为： • success：显示失败图标，此时 title 文本最多显示 7 个汉字长度 • error：显示失败图标，此时 title 文本最多显示 7 个汉字长度 • loading：显示加载图标，此时 title 文本最多显示 7 个汉字长度 • none：不显示图标，此时 title 文本最多可显示两行
image	string	–	否	自定义图标的本地路径，image 的优先级高于 icon
duration	number	1500	否	提示的延迟时间
mask	boolean	false	否	是否显示透明蒙层，防止触摸穿透

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.showToast({
  title: '成功',
  icon: 'success',
  duration: 2000
})
```

> ⚠ **注意：**
> - wx.showLoading 和 wx.showToast 同时只能显示一个。
> - wx.showLoading 应与 wx.hideLoading 配对使用。

enableAlertBeforeUnload

> 该 API 使用方法为 wx.enableAlertBeforeUnload(Object object)

- **功能说明：**开启小程序页面返回询问对话框。
- **弹窗条件：**
  - 当用户在小程序内非首页页面/最底层页。
  - 官方导航栏上的返回。
  - 全屏模式下自绘返回键。
  - android 系统 back 键时。

> ⚠ **注意：**
> - 手势滑动返回时不做拦截。
> - 在任何场景下，此功能都不应拦住用户退出小程序的行为。

- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

message	string	–	是	询问对话框内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

disableAlertBeforeUnload

> 该 API 使用方法为 wx.disableAlertBeforeUnload(Object object)

- **功能说明：**关闭小程序页面返回询问对话框。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

导航栏

最近更新时间：2023-10-20 15:19:16

setNavigationBarTitle

> 该 API 使用方法为 wx.setNavigationBarTitle(Object object)

- **功能说明：**隐藏分享面板的返回首页按钮。当用户打开的小程序最底层页面是非首页时，分享面板默认展示“返回首页”按钮，开发者可调用 hideHomeButton 进行隐藏。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
title	string	–	是	页面标题
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

```
wx.setNavigationBarTitle({
  title: '当前页面'
})
```

setNavigationBarColor

> 该 API 使用方法为 wx.setNavigationBarColor(Object object)

- **功能说明：**设置页面导航条颜色。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
frontColor	string	–	是	前景颜色值，包括按钮、标题、状态栏的颜色，仅支持 #ffffff 和 #000000
backgrou	stri	–	是	背景颜色值，有效值为十六进制颜色

ndColor	ng			
animation	object	-	否	动画效果
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.animation 的结构属性**

属性	类型	默认值	必填	说明
duration	number	0	否	动画变化时间，单位ms
timingFunc	string	'linear'	否	动画变化方式

- **object.animation.timingFunc 的合法值**

值	说明
'linear'	动画从头到尾的速度是相同的
'easeIn'	动画以低速开始
'easeOut'	动画以低速结束
'easeInOut'	动画以低速开始和结束

- **示例代码**

```
wx.setNavigationBarColor({
  frontColor: '#ffffff',
  backgroundColor: '#ff0000',
```

```
  animation: {
    duration: 400,
    timingFunc: 'easeIn'
  }
})
```

hideHomeButton

> 该 API 使用方法为 wx.hideHomeButton(Object object)

- **功能说明：**动态设置当前页面的标题。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.setNavigationBarTitle({
  title: '当前页面'
})
```

showNavigationBarLoading

> 该 API 使用方法为 wx.showNavigationBarLoading(Object object)

- **功能说明：**在当前页面显示导航条加载动画。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

hideNavigationBarLoading

> 该 API 使用方法为 wx.hideNavigationBarLoading(Object object)

- **功能说明：**在当前页面隐藏导航条加载动画。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

背景

最近更新时间：2023-10-20 15:19:16

setBackgroundColor

> 该 API 使用方法为 wx.setBackgroundColor(Object object)

- **功能说明：**动态设置窗口的背景色。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
backgroundColor	string	–	否	窗口的背景色，必须为十六进制颜色值
backgroundColorTop	string	–	否	顶部窗口的背景色，必须为十六进制颜色值，仅 iOS 支持
backgroundColorBottom	string	–	否	底部窗口的背景色，必须为十六进制颜色值，仅 iOS 支持
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.setBackgroundColor({
  backgroundColor: '#ffffff', // 窗口的背景色为白色
})

wx.setBackgroundColor({
  backgroundColorTop: '#ffffff', // 顶部窗口的背景色为白色
  backgroundColorBottom: '#ffffff', // 底部窗口的背景色为白色
})
```

setBackgroundTextStyle

> 该 API 使用方法为 wx.setBackgroundTextStyle(Object object)

- **功能说明：**动态设置下拉背景字体、loading 图的样式。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
textStyle	string	–	是	下拉背景字体、loading 图的样式。
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

  - **object.textStyle 的合法值**

值	说明
dark	dark 样式
light	light 样式

- **示例代码**

```
wx.setBackgroundTextStyle({
  textStyle: 'dark' // 下拉背景字体、loading 图的样式为dark
})
```

TabBar

最近更新时间：2023-10-20 15:19:16

showTabBar

> 该 API 使用方法为 wx.showTabBar(Object object)

- **功能说明：**显示 tabBar。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
animation	boolean	false	否	是否需要动画效果
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

hideTabBar

> 该 API 使用方法为 wx.hideTabBar(Object object)

- **功能说明：**隐藏 tabBar。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
animation	boolean	false	否	是否需要动画效果
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

setTabBarStyle

> 该 API 使用方法为 wx.setTabBarStyle(Object object)

- **功能说明：**动态设置 tabBar 的整体样式。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
color	string	–	否	tab 上的文字默认颜色，HexColor
selectedColor	string	–	否	tab 上的文字选中时的颜色，HexColor
backgroundColor	string	–	否	tab 的背景颜色，HexColor
borderStyle	string	–	否	tab 上边框的颜色，仅支持 black/white
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

setTabBarItem

> 该 API 使用方法为 wx.setTabBarItem(Object object)

- **功能说明：**动态设置 tabBar 某一项的内容。
- **参数及说明：**Object object。

属性	类型	默认	必填	说明

		值		
index	number	–	是	tabBar 的哪一项，从左边算起
text	string	–	否	tab 上的按钮文字
iconPath	string	–	否	图片路径，icon 大小限制为 40kb，建议尺寸为 81px * 81px，当 postion 为 top 时，此参数无效
selectedIconPath	string	–	否	选中时的图片路径，icon 大小限制为 40kb，建议尺寸为 81px * 81px ，当 postion 为 top 时，此参数无效
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.setTabBarItem({
  index: 0,
  text: 'text',
  iconPath: '/path/to/iconPath',
  selectedIconPath: '/path/to/selectedIconPath'
})
```

showTabBarRedDot

> 该 API 使用方法为 wx.showTabBarRedDot(Object object)

- **功能说明：**显示 tabBar 某一项的右上角的红点。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

index	number	–	是	tabBar 的哪一项，从左边算起
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

hideTabBarRedDot

> 该 API 使用方法为 wx.hideTabBarRedDot(Object object)

- **功能说明：**隐藏 tabBar 某一项的右上角的红点。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
index	number	–	是	tabBar 的哪一项，从左边算起
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

setTabBarBadge

> 该 API 使用方法为 wx.setTabBarBadge(Object object)

- **功能说明：**为 tabBar 某一项的右上角添加文本。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
index	number	–	是	tabBar 的哪一项，从左边算起

text	string	–	是	显示的文本，超过 4 个字符则显示成 ...
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.setTabBarBadge({
  index: 0,
  text: '1'
})
```

removeTabBarBadge

> 该 API 使用方法为 wx.removeTabBarBadge(Object object)

- **功能说明：**移除 tabBar 某一项右上角的文本。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
index	number	–	是	tabBar 的哪一项，从左边算起
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

下拉刷新

最近更新时间：2023-10-20 15:19:16

startPullDownRefresh

> 该 API 使用方法为 wx.startPullDownRefresh(Object object)

- **功能说明：**开始下拉刷新。调用后触发下拉刷新动画，效果与用户手动下拉刷新一致。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.startPullDownRefresh()
```

stopPullDownRefresh

> 该 API 使用方法为 wx.stopPullDownRefresh(Object object)

- **功能说明：**停止当前页面下拉刷新。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
Page({
  onPullDownRefresh () {
    wx.stopPullDownRefresh()
  }
})
```

滚动

最近更新时间：2023-10-20 15:19:16

pageScrollTo

> 该 API 使用方法为 wx.pageScrollTo(Object object)

- **功能说明：**将页面滚动到目标位置。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
scrollTop	number	–	是	滚动到页面的目标位置，单位 px
duration	number	300	否	滚动动画的时长，单位 ms
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.pageScrollTo({
  scrollTop: 0,
  duration: 300
})
```

ScrollViewContext

- **功能说明：**增强 ScrollView 实例，可通过 wx.createSelectorQuery 的 NodesRef.node 方法获取。仅在 scroll-view 组件开启 enhanced 属性后生效。
- **属性及说明**

属性	类型	说明
scrollEnabled	boolean	滚动开关

bounces	boolean	设置滚动边界弹性 (暂不支持)
showScrollbar	boolean	设置是否显示滚动条
pagingEnabled	boolean	分页滑动开关
fastDeceleration	boolean	设置滚动减速速率
decelerationDisabled	boolean	取消滚动惯性 (仅在 iOS 下生效)

- **示例代码：**

```
wx.createSelectorQuery()
  .select('#scrollview')
  .node()
  .exec((res) => {
    const scrollView = res[0].node;
    scrollView.scrollEnabled = false;
  })
```

.scrollIntoView

> 该 方法 使用方式为 ScrollViewContext.scrollIntoView(string selector, object ScrollIntoViewOptions)

- **功能说明：**滚动至指定位置。
- **参数及说明：**string selector，元素选择器。

.scrollTo

> 该 方法 使用方式为 ScrollViewContext.scrollTo(Object object)

- **功能描述：**滚动至指定位置。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
top	number	-	否	顶部距离
left	number	-	否	左边界距离
velocity	number	-	否	初始速度
duration	number	-	否	滚动动画时长

animated	boolean	-	否	是否启用滚动动画

动画

最近更新时间：2023-10-20 15:19:16

createAnimation

该 API 使用方法为 wx.createAnimation(Object object)

- **功能说明：**创建一个动画实例 animation 。调用实例的方法来描述动画。最后通动画实例的 export 方法导出动画数据传递给组件的 animation 属性。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
duration	number	400	否	动画持续时间，单位：ms
timingFunction	string	'linear'	否	动画的效果
delay	number	0	否	动画延迟时间，单位：ms
transformOrigin	string	'50% 50% 0'	否	-

- **timingFunction 的合法值**

值	说明
'linear'	动画从头到尾的速度是相同的
'ease'	动画以低速开始，然后加快，在结束前变慢
'ease-in'	动画以低速开始
'ease-in-out'	动画以低速开始和结束
'ease-out'	动画以低速结束
'step-start'	动画第一帧就跳至结束状态直到结束
'step-end'	动画一直保持开始状态，最后一帧跳到结束状态

- **返回值：**Animation 。

Animation

动画对象

示例代码

对应的 WXML 文件

```
<view
  animation="{{animationData}}"
  style="background:red;height:100rpx;width:100rpx"
></view>
```

对应的 js 文件

```
Page({
  data: {
    animationData: {}
  },
  onShow() {
    const animation = wx.createAnimation({
      duration: 1000,
      timingFunction: 'ease',
    })

    this.animation = animation

    animation.scale(2, 2).rotate(45).step()

    this.setData({
      animationData: animation.export()
    })

    setTimeout(function () {
      animation.translate(30).step()
      this.setData({
        animationData: animation.export()
      })
    }.bind(this), 1000)
  },
  rotateAndScale() {
    // 旋转同时放大
    this.animation.rotate(45).scale(2, 2).step()
    this.setData({
      animationData: this.animation.export()
    })
```

```
  },
  rotateThenScale() {
    // 先旋转后放大
    this.animation.rotate(45).step()
    this.animation.scale(2, 2).step()
    this.setData({
      animationData: this.animation.export()
    })
  },
  rotateAndScaleThenTranslate() {
    // 先旋转同时放大，然后平移
    this.animation.rotate(45).scale(2, 2).step()
    this.animation.translate(100, 100).step({duration: 1000})
    this.setData({
      animationData: this.animation.export()
    })
  }
})
```

.backgroundColor

> 该 方法 使用方式为 Animation Animation.backgroundColor(string value)

- **功能说明：**设置背景色。
- **参数及说明：**string value，颜色值。
- **返回值：**Animation 。

.bottom

> 该 方法 使用方式为 Animation Animation.bottom(number|string value)

- **功能说明：**设置 bottom 值。
- **参数及说明：**number|string value，长度值，如果传入 number 则默认使用 px，可传入其他自定义单位的长度值。
- **返回值：**Animation 。

.export

> 该 API 方法的使用方法为 Object Animation.export()

- **功能说明：**导出动画队列。export 方法每次调用后会清掉之前的动画操作。
- **返回值：**Object，animationData。

属性	类型
actions	Array.<Object>

.height

> 该 方法 使用方式为 Animation Animation.height(number|string value)

- **功能说明：**设置高度。
- **参数及说明：**number|string value，长度值，如果传入 number 则默认使用 px，可传入其他自定义单位的长度值。
- **返回值：**Animation 。

.left

> 该 方法 使用方式为 Animation Animation.left(number|string value)

- **功能说明：**设置 left 值。
- **参数及说明：**number|string value，长度值，如果传入 number 则默认使用 px，可传入其他自定义单位的长度值。
- **返回值：**Animation 。

.matrix

> 该 API 使用方法为 Animation Animation.matrix()

- **功能说明：**同 transform-function matrix 一致。
- **返回值：**Animation 。

.matrix3d

> 该 API 使用方法为 Animation Animation.matrix3d()

- **功能说明：**同 transform-function matrix 一致。
- **返回值：**Animation 。

.opacity

> 该 方法 使用方式为 Animation Animation.opacity(number value)

- **功能说明：**设置透明度。
- **参数及说明：**number value，透明度，范围 0-1。

- **返回值：** Animation 。

.right

> 该 方法 使用方式为 Animation Animation.right(number|string value)

- **功能说明：** 设置 right 值。
- **参数及说明：** number|string value，长度值，如果传入 number 则默认使用 px，可传入其他自定义单位的长度值。
- **返回值：** Animation 。

.rotate

> 该 API 使用方法为

- **功能说明：** 从原点顺时针旋转一个角度，使用方法为 Animation Animation.rotate(number angle)。
- **参数及说明：** number angle，旋转的角度。范围 [-180, 180]。
- **返回值：** Animation 。

.rotate3d

> 该 API 使用方法为 Animation Animation.rotate3d(number x, number y, number z, number angle)

- **功能说明：** 从固定轴顺时针旋转一个角度。
- **参数及说明：** number x，旋转轴的 x 坐标；number y，旋转轴的 y 坐标；number z，旋转轴的 z 坐标；number angle，旋转的角度。范围 [-180, 180]。
- **返回值：** Animation 。

.rotateX

> 该 API 使用方法为 Animation Animation.rotateX(number angle)

- **功能说明：** 从 X 轴顺时针旋转一个角度。
- **参数及说明：** number angle，旋转的角度。范围 [-180, 180]。
- **返回值：** Animation 。

.rotateY

> 该 方法 使用方式为 Animation Animation.rotateY(number angle)

- **功能说明：** 从 Y 轴顺时针旋转一个角度。

- **参数及说明：**number angle，旋转的角度。范围 [-180, 180]。
- **返回值：**Animation 。

.rotateZ

该 方法 使用方式为 Animation Animation.rotateZ(number angle)

- **功能说明：**从 Z 轴顺时针旋转一个角度。
- **参数及说明：**number angle，旋转的角度。范围 [-180, 180]。
- **返回值：**Animation 。

.scale

该 方法 使用方式为 Animation Animation.scale(number sx, number sy)

- **功能说明：**缩放。
- **参数及说明：**number sx，当仅有 sx 参数时，表示在 X 轴、Y 轴同时缩放sx倍数；number sy，在 Y 轴缩放 sy 倍数。
- **返回值：**Animation 。

.scale3d

该 方法 使用方式为 Animation Animation.scale3d(number sx, number sy, number sz)

- **功能说明：**缩放。
- **参数及说明：**number sx，当仅有 sx 参数时，表示在 X 轴、Y 轴同时缩放sx倍数；number sy，在 Y 轴缩放 sy 倍数。
- **返回值：**Animation 。

.scaleX

该 方法 使用方式为 Animation Animation.scaleX(number scale)

- **功能说明：**缩放 X 轴。
- **参数及说明：**number scale，X 轴的缩放倍数。
- **返回值：**Animation 。

scaleY

该 方法 使用方式为 Animation Animation.scaleY(number scale)

- **功能说明：**缩放 Y 轴。

- **参数及说明：**number scale，Y 轴的缩放倍数。
- **返回值：**Animation 。

.scaleZ

该 方法 使用方式为 Animation Animation.scaleZ(number scale)

- **功能说明：**缩放 Z 轴。
- **参数及说明：**number scale，Z 轴的缩放倍数。
- **返回值：**Animation 。

.skew

该 方法 使用方式为 Animation Animation.skew(number ax, number ay)

- **功能说明：**对 X、Y 轴坐标进行倾斜。
- **参数及说明：**number sx，当仅有 sx 参数时，表示在 X 轴、Y 轴同时缩放sx倍数；number sy，在 Y 轴缩放 sy 倍数。
- **返回值：**Animation 。

.skewX

该 方法 使用方式为 Animation Animation.skewX(number angle)

- **功能说明：**对 X 轴坐标进行倾斜。
- **参数及说明：**number scale，X 轴的缩放倍数。
- **返回值：**Animation 。

.skewY

该 方法 使用方式为 Animation Animation.skewY(number angle)

- **功能说明：**对 Y 轴坐标进行倾斜。
- **参数及说明：**number scale，Y 轴的缩放倍数。
- **返回值：**Animation 。

.step

该 API 使用方法为 Animation Animation.step(Object object)

- **功能说明：**表示一组动画完成。可以在一组动画中调用任意多个动画方法，一组动画中的所有动画会同时开始，一组动画完成后才会进行下一组动画。

- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
duration	number	400	否	动画持续时间，单位 ms
timingFunction	string	'linear'	否	动画的效果
delay	number	0	否	动画延迟时间，单位 ms
transformOrigin	string	'50% 50% 0'	否	-

- **timingFunction 的合法值**

值	说明
'linear'	动画从头到尾的速度是相同的
'ease'	动画以低速开始，然后加快，在结束前变慢
'ease-in'	动画以低速开始
'ease-in-out'	动画以低速开始和结束
'ease-out'	动画以低速结束
'step-start'	动画第一帧就跳至结束状态直到结束
'step-end'	动画一直保持开始状态，最后一帧跳到结束状态

- **返回值：** Animation 。

.top

> 该 方法 使用方式为 Animation Animation.top(number|string value)

- **功能说明：** 设置 top 值。
- **参数及说明：** number|string value，长度值，如果传入 number 则默认使用 px，可传入其他自定义单位的长度值。
- **返回值：** Animation 。

.translate

> 该 方法 使用方式为 Animation Animation.translate(number tx, number ty)

- **功能说明：**平移变换。
- **参数及说明：**number tx，当仅有该参数时表示在 X 轴偏移 tx，单位 px；number ty，在 Y 轴平移的距离，单位为 px。
- **返回值：**Animation 。

.translate3d

该 方法 使用方式为 Animation Animation.translate3d(number tx, number ty, number tz)

- **功能说明：**对 xyz 坐标进行平移变换。
- **参数及说明：**number tx，当仅有该参数时表示在 X 轴偏移 tx，单位 px；number ty，在 Y 轴平移的距离，单位为 px。
- **返回值：**Animation 。

.translateX

该 方法 使用方式为 Animation Animation.translateX(number translation)

- **功能说明：**对 X 轴平移。
- **参数及说明：**number translation，在 X 轴平移的距离，单位为 px。
- **返回值：**Animation 。

.translateY

该 方法 使用方式为 Animation Animation.translateY(number translation)

- **功能说明：**对 Y 轴平移。
- **参数及说明：**number translation，在 Y 轴平移的距离，单位为 px。
- **返回值：**Animation 。

.translateZ

该 方法 使用方式为 Animation Animation.translateZ(number translation)

- **功能说明：**对 Z 轴平移。
- **参数及说明：**number translation，在 Z 轴平移的距离，单位为 px。
- **返回值：**Animation 。

.width

该 方法 使用方式为 Animation Animation.width(number|string value)

- **功能说明：**设置宽度，使用方法为。
- **参数及说明：**number|string value，长度值，如果传入 number 则默认使用 px，可传入其他自定义单位的长度值。
- **返回值：**Animation 。

自定义组件

最近更新时间：2023-10-20 15:19:16

nextTick

> 该 API 使用方法为 wx.nextTick(function callback)

- **功能说明：**延迟一部分操作到下一个时间片再执行（类似于 setTimeout）。
- **参数及说明：**function callback，因为自定义组件中的 setData 和 triggerEvent 接口本身是同步的操作，当这几个接口被连续调用时，都是在一个同步流程中执行完的，因此若逻辑不当可能会导致出错。

  一个极端的案例：当父组件的 setData 引发了子组件的 triggerEvent，进而使得父组件又进行了一次 setData，期间有通过 wx:if 语句对子组件进行卸载，就有可能引发错误，所以对于不需要在一个同步流程内完成的逻辑，可以使用此接口延迟到下一个时间片再执行。
- **示例代码**

```
Component({
  doSth() {
    this.setData({ number: 1 }) // 直接在当前同步流程中执行

    wx.nextTick(() => {
      this.setData({ number: 3 }) // 在当前同步流程结束后，下一个时间片执行
    })

    this.setData({ number: 2 }) // 直接在当前同步流程中执行
  }
})
```

菜单

最近更新时间：2023-10-20 15:19:16

getMenuButtonBoundingClientRect

> 该 API 使用方法为 wx.getMenuButtonBoundingClientRect()

- **功能说明：**获取菜单按钮（右上角胶囊按钮）的布局位置信息。坐标信息以屏幕左上角为原点，使用方法为 。
- **返回值：**Object。
- **菜单按钮的布局位置信息：**

属性	类型	说明
width	number	宽度，单位：px
height	number	高度，单位：px
top	number	上边界坐标，单位：px
right	number	右边界坐标，单位：px
bottom	number	下边界坐标，单位：px
left	number	左边界坐标，单位：px

窗口

最近更新时间：2023-10-20 15:19:17

onWindowResize

> 该 API 使用方法为 wx.onWindowResize(function listener)

- **功能说明：**监听窗口尺寸变化事件。
- **参数及说明：**function listener，窗口尺寸变化事件的监听函数。

属性	类型
size	Object

  - **size 结构属性**

结构属性	类型	说明
windowWidth	number	变化后的窗口宽度，单位 px
windowHeight	number	变化后的窗口高度，单位 px

offWindowResize

> 该 API 使用方法为 wx.offWindowResize(function listener)

- **功能说明：**移除窗口尺寸变化事件的监听函数。
- **参数及说明：**function listener，onWindowResize 传入的监听函数。不传此参数则移除所有监听函数。

```
const listener = function (res) { console.log(res) }

wx.onWindowResize(listener)
wx.offWindowResize(listener) // 需传入与监听时同一个的函数对象
```

字体

最近更新时间：2023-10-20 15:19:17

loadFontFace

> 该 API 使用方法为 wx.loadFontFace(Object object)

- **功能说明：**动态加载网络字体，文件地址需为下载类型，需在 app.js 中调用。

> ⚠ **注意：**
> - 字体文件返回的 contet-type 参考 font，格式不正确时会解析失败。
> - 字体链接必须是 https（ iOS 不支持 http ）。
> - 字体链接必须是同源下的，或开启了 cors 支持，小程序的域名是 servicewechat.com。
> - 工具里提示 Faild to load font 可以忽略。
> - 2.10.0以前仅在调用页面生效。

- **参数及说明：**Object object

属性	类型	默认值	必填	说明
global	boolean	false	否	是否全局生效
family	string	–	是	定义的字体名称
source	string	–	是	字体资源的地址。建议格式为 TTF 和 WOFF，WOFF2 在低版本的 iOS 上会不兼容
desc	object	–	否	可选的字体描述符
scopes	Array	–	否	字体作用范围，可选值为 webview / native，默认 webview，设置 native 可在 Canvas 2D 下使用
success	function	–	否	接口调用成功的回调函数
fail	fun	–	否	接口调用成功的回调函数

	ction			
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **desc 结构值**

结构属性	类型	默认值	必填	说明
style	string	normal	否	字体样式，可选值为 normal / italic / oblique
weight	string	normal	否	字体粗细，可选值为 normal / bold / 100 / 200/ 900
variant	string	normal	否	设置小型大写字母的字体显示文本，可选值为 normal / small-caps / inherit

- **object.success 回调函数参数：** Object res

属性	类型	说明
status	string	加载字体结果

- **object.fail 回调函数参数：** Object res

属性	类型	说明
status	string	加载字体结果

- **object.complete 回调函数参数：** Object res

属性	类型	说明
status	string	加载字体结果

- **示例代码**

```
wx.loadFontFace({
  family: 'Bitstream Vera Serif Bold',
  source: 'url("https://sungd.github.io/Pacifico.ttf")',
  success: console.log
})
```

网络

发起请求

最近更新时间：2023-10-20 15:19:17

request

> 该 API 使用方法为 RequestTask wx.request(Object object)

- **功能说明：**发起 HTTPS 网络请求。使用前请注意阅读 相关说明 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	string	-	是	开发者服务器接口地址
data	string/object/ArrayBuffer	-	否	请求的参数
header	Object	-	否	设置请求的 header，header 中不能设置 Referer。content-type 默认为 application/json
method	string	GET	否	HTTP 请求方法
dataType	string	json	否	返回的数据格式
responseType	string	text	否	响应的数据类型
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

  - **object.method 的合法值：**

值	说明
OPTIONS	HTTP 请求 OPTIONS

GET	HTTP 请求 GET
HEAD	HTTP 请求 HEAD
POST	HTTP 请求 POST
PUT	HTTP 请求 PUT
DELETE	HTTP 请求 DELETE
TRACE	HTTP 请求 TRACE
CONNECT	HTTP 请求 CONNECT

- **object.dataType 的合法值：**

值	说明
json	返回的数据为 JSON，返回后会对返回的数据进行一次 JSON.parse
其他	不对返回的内容进行 JSON.parse

- **object.responseType 的合法值**

值	说明
text	响应的数据为文本
arraybuffer	响应的数据为 ArrayBuffer

- **object.success 回调函数参数：**Object res

属性	类型	说明
data	string/Object/Arraybuffer	开发者服务器返回的数据
statusCode	number	开发者服务器返回的 HTTP 状态码
header	Object	开发者服务器返回的 HTTP Response Header

- **返回值：**RequestTask，请求任务对象。
- **data 参数说明：**

  最终发送给服务器的数据是 String 类型，如果传入的 data 不是 String 类型，会被转换成 String。转换规则如下：

  - 对于 `GET` 方法的数据，会将数据转换成 query string（

encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...
)

- 对于 POST 方法且 header['content-type'] 为 application/json 的数据，会对数据进行 JSON 序列化
- 对于 POST 方法且 header['content-type'] 为 application/x-www-form-urlencoded 的数据，会将数据转换成 query string
  (encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...)

- **示例代码：**

```
wx.request({
  url: 'test.php', // 仅为示例，并非真实的接口地址
  data: {
    x: '',
    y: ''
  },
  header: {
    'content-type': 'application/json' // 默认值
  },
  success(res) {
    console.log(res.data)
  }
})
```

RequestTask

.abort

> 该 方法 使用方式为 RequestTask.abort()

- **功能说明：**中断请求任务。

.onChunkReceived

> 该 方法 使用方式为 RequestTask.onChunkReceived(function listener)

- **功能说明：**监听 Transfer-Encoding Chunk Received 事件。当接收到新的chunk时触发。
- **参数及说明：**Object res 参数，function listener，Transfer-Encoding Chunk Received 事件的监听函数。

属性	类型	说明

res	Object	开发者服务器每次返回新 chunk 时的 Response

- **res 结构属性**

结构属性	类型	说明
data	ArrayBuffer	返回的 chunk buffer

.offChunkReceived

该 方法 使用方式为 RequestTask.offChunkReceived(function listener)

- **功能说明：**移除 Transfer-Encoding Chunk Received 事件的监听函数。
- **参数及说明：**function listener，onChunkReceived 传入的监听函数，不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

RequestTask.onChunkReceived(listener)
RequestTask.offChunkReceived(listener) // 需传入与监听时同一个的函数对象
```

.onHeadersReceived

该 方法 使用方式为 RequestTask.onHeadersReceived(function listener)

- **功能说明：**移除 HTTP Response Header 事件的监听函数。
- **参数及说明：**function listener，onHeadersReceived 传入的监听函数，不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

RequestTask.onHeadersReceived(listener)
RequestTask.offHeadersReceived(listener) // 需传入与监听时同一个的函数对象
```

.offHeadersReceived

该 方法 使用方式为 RequestTask.offHeadersReceived(function listener)

- **功能说明：**监听 HTTP Response Header 事件。会比请求完成事件更早。

- **参数及说明：**function listener，HTTP Response Header 事件的监听函数。

属性	类型	说明
header	Object	开发者服务器返回的 HTTP Response Header

上传

最近更新时间：2023-10-20 15:19:17

uploadFile

> 该 API 使用方法为 UploadTask wx.uploadFile(Object object)

- **功能说明：**将本地资源上传到服务器。客户端发起一个 HTTPS POST 请求，其中 `content-type` 为 `multipart/form-data` 。使用前请注意阅读 相关说明 。·
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	string	–	是	开发者服务器址
filePath	string	–	是	要上传文件资源的路径 (本地路径)
name	string	–	是	文件对应的 key，开发者在服务端可以通过这个 key 获取文件的二进制内容
header	object	–	否	HTTP 请求 Header，Header 中不能设置 Referer
formData	object	–	否	HTTP 请求中其他额外的 form data
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明

data	string	开发者服务器返回的数据
statusCode	number	开发者服务器返回的 HTTP 状态码

- **返回值：** UploadTask，一个可以监听上传进度进度变化的事件和取消上传的对象。
- **示例代码：**

```
wx.chooseImage({
  success (res) {
    const tempFilePaths = res.tempFilePaths
    wx.uploadFile({
      url: 'https://example.weixin.qq.com/upload', //仅为示例，非真实的接口地址
      filePath: tempFilePaths[0],
      name: 'file',
      formData: {
        'user': 'test'
      },
      success (res){
        const data = res.data
        //do something
      }
    })
  }
})
```

UploadTask

.abort

该 方法 使用方式为 UploadTask.abort()

- **功能说明：** 中断上传任务。

.onProgressUpdate

该 方法 使用方式为 UploadTask.onProgressUpdate(function listener)

- **功能说明：** 监听上传进度变化事件。
- **参数及说明：** function listener，上传进度变化事件的监听函数。

属性	类型	说明
progress	number	下载进度百分比

totalBytesWritten	number	已经下载的数据长度，单位 Bytes
totalBytesExpectedToWrite	number	预期需要下载的数据总长度，单位 Bytes

.offProgressUpdate

> 该 方法 使用方式为 UploadTask.offProgressUpdate(function listener)

- **功能说明：**移除上传进度变化事件的监听函数。
- **参数及说明：**function listener，onProgressUpdate 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

DownloadTask.onProgressUpdate(listener)
DownloadTask.offProgressUpdate(listener) // 需传入与监听时同一个的函数对象
```

.onHeadersReceived

> 该 方法 使用方式为 UploadTask.onHeadersReceived(function listener)

- **功能说明：**监听 HTTP Response Header 事件。会比请求完成事件更早。
- **参数及说明：**function listener，HTTP Response Header 事件的监听。

属性	类型	说明
header	Object	开发者服务器返回的 HTTP Response Header

.offHeadersReceived

> 该 方法 使用方式为 UploadTask.offHeadersReceived(function listener)

- **功能说明：**移除 HTTP Response Header 事件的监听函数。
- **参数及说明：**function listener，onHeadersReceived 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

UploadTask.onHeadersReceived(listener)
UploadTask.offHeadersReceived(listener) // 需传入与监听时同一个的函数对象
```

示例代码

```
const uploadTask = wx.uploadFile({
  url: 'http://example.weixin.qq.com/upload', //仅为示例，非真实的接口地址
  filePath: tempFilePaths[0],
  name: 'file',
  formData:{
    'user': 'test'
  },
  success (res){
    const data = res.data
    //do something
  }
})

uploadTask.onProgressUpdate((res) => {
  console.log('上传进度', res.progress)
  console.log('已经上传的数据长度', res.totalBytesSent)
  console.log('预期需要上传的数据总长度', res.totalBytesExpectedToSend)
})

uploadTask.abort() // 取消上传任务
```

下载

最近更新时间：2023-10-20 15:19:17

downloadFile

该 API 使用方法为 DownloadTask wx.downloadFile(Object object)

> ⚠ **注意：**
> 请在服务端响应的 header 中指定合理的 Content-Type 字段，以保证客户端正确处理文件类型。

- **功能说明：**下载文件资源到本地。客户端直接发起一个 HTTPS GET 请求，返回文件的本地临时路径 (本地路径)，单次下载允许的最大文件为200MB。使用前请参考阅读微信的 相关说明 。
- **参数及说明：**Object object。

属性	类型	必填	说明
url	string	是	下载资源的 url
header	Object	否	HTTP 请求的 Header，Header 中不能设置 Referer
timeout	number	否	超时时间，单位 ms
filePath	string	否	指定文件下载后存储的路径
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
tempFilePath	string	临时文件路径 (本地路径)。没传入 filePath 指定文件存储路径时会返回，下载后的文件会存储到一个临时文件
filePath	string	用户文件路径 (本地路径)。传入 filePath 时会返回，跟传入的 filePath 一致
statusCode	number	开发者服务器返回的 HTTP 状态码
profile	Object	网络请求过程中一些调试信息

- **profile 结构属性：**

结构属性	类型	说明
redirectStart	number	第一个 HTTP 重定向发生时的时间。有跳转且是同域名内的重定向才算，否则值为0
redirectEnd	number	最后一个 HTTP 重定向完成时的时间。有跳转且是同域名内部的重定向才算，否则值为0
fetchStart	number	组件准备好使用 HTTP 请求抓取资源的时间，这发生在检查本地缓存之前
domainLookupStart	number	DNS 域名查询开始的时间，如果使用了本地缓存（即无 DNS 查询）或持久连接，则与 fetchStart 值相等
domainLookupEnd	number	DNS 域名查询完成的时间，如果使用了本地缓存（即无 DNS 查询）或持久连接，则与 fetchStart 值相等
connectStart	number	HTTP（TCP）开始建立连接的时间，如果是持久连接，则与 fetchStart 值相等。**如果在传输层发生了错误且重新建立连接，则这里显示的是新建立的连接开始的时间**
connectEnd	number	HTTP（TCP）完成建立连接的时间（完成握手），如果是持久连接，则与 fetchStart 值相等。注意如果在传输层发生了错误且重新建立连接，则这里显示的是新建立的连接完成的时间。**这里握手结束，包括安全连接建立完成、SOCKS 授权通过**
SSLconnectionStart	number	SSL建立连接的时间,如果不是安全连接,则值为0

SSLconnectionEnd	number	SSL建立完成的时间,如果不是安全连接,则值为0
requestStart	number	HTTP请求读取真实文档开始的时间（完成建立连接），包括从本地读取缓存。连接错误重连时，这里显示的也是新建立连接的时间
requestEnd	number	HTTP请求读取真实文档结束的时间
responseStart	number	HTTP 开始接收响应的时间（获取到第一个字节），包括从本地读取缓存
responseEnd	number	HTTP 响应全部接收完成的时间（获取到最后一个字节），包括从本地读取缓存
rtt	number	当次请求连接过程中实时 rtt
estimate_nettype	number	评估的网络状态：unknown，offline，slow 2g，2g，3g，4g，last/0，1，2，3，4，5，6
httpRttEstimate	number	协议层根据多个请求评估当前网络的 rtt（仅供参考）
transportRttEstimate	number	协议层根据多个请求评估当前网络的 rtt（仅供参考）
downstreamThroughputKbpsEstimate	number	评估当前网络下载的 kbps
throughputKbps	number	当前网络的实际下载 kbps
peerIP	string	当前请求的 IP
port	number	当前请求的端口
socketReused	boolean	是否复用连接
sendBytesCount	number	发送的字节数
receivedBytedCount	number	收到字节数

protocol	string	使用协议类型，有效值：http1.1，h2，quic，unknown

- **返回值：** DownloadTask
- **示例代码**

```
wx.downloadFile({
  url: 'https://example.com/audio/123', //仅为示例，并非真实的资源
  success (res) {
    // 只要服务器有响应数据，就会把响应内容写入文件并进入 success 回调，业务需要自行判断是否下载到了想要的内容
    if (res.statusCode === 200) {
      wx.playVoice({
        filePath: res.tempFilePath
      })
    }
  }
})
```

DownloadTask

.abort

> 该 方法 使用方式为 DownloadTask.abort()

- **功能说明：** 中断下载任务。

.onProgressUpdate

> 该 方法 使用方式为 DownloadTask.onProgressUpdate(function listener)

- **功能说明：** 监听下载进度变化事件。
- **参数及说明：** function listener，下载进度变化事件的监听函数。

属性	类型	说明
progress	number	下载进度百分比
totalBytesWritten	number	已经下载的数据长度，单位 Bytes
totalBytesExpectedToWrite	number	预期需要下载的数据总长度，单位 Bytes

.offProgressUpdate

> 该 方法 使用方式为 DownloadTask.offProgressUpdate(function listener)

- **功能说明：**移除下载进度变化事件的监听函数。
- **参数及说明：**function listener，onProgressUpdate 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

DownloadTask.onHeadersReceived(listener)
DownloadTask.offHeadersReceived(listener) // 需传入与监听时同一个的函数对象
```

.onHeadersReceived

> 该 方法 使用方式为 DownloadTask.onHeadersReceived(function listener)

- **功能说明：**监听 HTTP Response Header 事件。会比请求完成事件更早。
- **参数及说明：**function listener，HTTP Response Header 事件的监听函数。

属性	类型	说明
header	Object	开发者服务器返回的 HTTP Response Header

.offHeadersReceived

> 该 方法 使用方式为 DownloadTask.offHeadersReceived(function listener)

- **功能说明：**移除 HTTP Response Header 事件的监听函数。
- **参数及说明：**function listener，onHeadersReceived 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

DownloadTask.onHeadersReceived(listener)
DownloadTask.offHeadersReceived(listener) // 需传入与监听时同一个的函数对象
```

示例代码

```
const downloadTask = wx.downloadFile({
  url: 'http://example.com/audio/123', //仅为示例，并非真实的资源
  success (res) {
    wx.playVoice({
      filePath: res.tempFilePath
    })
  }
})

downloadTask.onProgressUpdate((res) => {
  console.log('下载进度', res.progress)
  console.log('已经下载的数据长度', res.totalBytesWritten)
  console.log('预期需要下载的数据总长度', res.totalBytesExpectedToWrite)
})

downloadTask.abort() // 取消下载任务
```

WebSocket

最近更新时间：2023-10-20 15:19:17

> **说明：**
> 推荐使用 SocketTask 的方式去管理 webSocket 链接，每一条链路的生命周期都更加可控，同时存在多个 webSocket 的链接的情况下使用 wx 前缀的方法可能会带来一些和预期不一致的情况。

sendSocketMessage

该 API 使用方法为 wx.sendSocketMessage(Object object)

- **功能说明：**通过 WebSocket 连接发送数据。需要先 wx.connectSocket，并在 wx.onSocketOpen 回调之后才能发送。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	string/ArrayBuffer	–	是	需要发送的内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
let socketOpen = false
let socketMsgQueue = []
wx.connectSocket({
  url: 'test.php'
})

wx.onSocketOpen(function(res) {
  socketOpen = true
  for (let i = 0; i < socketMsgQueue.length; i++){
```

```
    sendSocketMessage(socketMsgQueue[i])
  }
  socketMsgQueue = []
})

function sendSocketMessage(msg) {
  if (socketOpen) {
    wx.sendSocketMessage({
      data:msg
    })
  } else {
    socketMsgQueue.push(msg)
  }
}
```

onSocketOpen

> 该 API 使用方法为 wx.onSocketOpen(function listener)

- **功能说明：**监听 WebSocket 连接打开事件。
- **参数及说明：**function listener，WebSocket 连接打开事件的监听函数。

属性	类型	说明
header	object	连接成功的 HTTP 响应 Header

onSocketMessage

> 该 API 使用方法为 wx.onSocketMessage(function listener)

- **功能说明：**监听 WebSocket 接收到服务器的消息事件。
- **参数及说明：**function listener，WebSocket 接收到服务器的消息事件的监听函数。

属性	类型	说明
data	string/ArrayBuffer	服务器返回的消息

onSocketError

> 该 API 使用方法为 wx.onSocketError(function listener)

- **功能说明：**监听 WebSocket 错误事件。
- **参数及说明：**function listener，WebSocket 错误事件的监听函数。

属性	类型	说明
errMsg	string	错误信息

onSocketClose

> 该 API 使用方法为 wx.onSocketClose(function listener)

- **功能说明：**监听 WebSocket 连接关闭事件。
- **参数及说明：**function listener，WebSocket 连接关闭事件的监听函数。

属性	类型	说明
code	number	一个数字值表示关闭连接的状态号，表示连接被关闭的原因
reason	string	一个可读的字符串，表示连接被关闭的原因

connectSocket

> 该 API 使用方法为 SocketTask wx.connectSocket(Object object)

- **功能说明：**创建一个 WebSocket 连接。使用前请注意阅读微信的相关说明 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	string	-	是	开发者服务器 wss 接口地址
header	Object	-	否	HTTP Header，Header 中不能设置 Referer
protocols	Array.<string>	-	否	子协议数组
tcpNoDelay	boolean	false	否	建立 TCP 连接的时候的 TCP_NODELAY 设置
perMessageDeflate	boolean	false	否	是否开启压缩扩展
timeout	numb	-	否	超时时间，单位为毫秒

	er			
forceCellularNetwork	boolean	false	否	wifi 下使用移动网络发送请求
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **返回值：** SocketTask 。
- **并发数**
  - 1.5.0及以上版本，最多可以同时存在5个 WebSocket 连接。
  - 1.5.0以下版本，一个小程序同时只能有一个 WebSocket 连接，如果当前已存在一个 WebSocket 连接，会自动关闭该连接，并重新创建一个 WebSocket 连接。
- **示例代码**

```
wx.connectSocket({
  url: 'wss://example.qq.com',
  header:{
    'content-type': 'application/json'
  },
  protocols: ['protocol1']
})
```

closeSocket

> 该 API 使用方法为 wx.closeSocket(Object object)

- **功能说明：** 关闭 WebSocket 连接，使用方法为。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
code	number	1000（表示正常关闭连接）	否	一个数字值表示关闭连接的状态号，表示连接被关闭的原因
reas	strin	–	否	一个可读的字符串，表示连接被关闭的原因。这个

on	g			字符串必须是不长于 123 字节的 UTF-8 文本（不是字符）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.connectSocket({
  url: 'test.php'
})

//注意这里有时序问题，
//如果 wx.connectSocket 还没回调 wx.onSocketOpen，而先调用 wx.closeSocket，那么就做不到关闭 WebSocket 的目的。
//必须在 WebSocket 打开期间调用 wx.closeSocket 才能关闭。
wx.onSocketOpen(function() {
  wx.closeSocket()
})

wx.onSocketClose(function(res) {
  console.log('WebSocket 已关闭！')
})
```

SocketTask

.close

> 该 方法 使用方式为 SocketTask.close(Object object)

- **功能说明：**关闭 WebSocket 连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
code	number	1000（表示正常关闭连接）	否	一个数字值表示关闭连接的状态号，表示连接被关闭的原因

reason	string	–	否	一个可读的字符串，表示连接被关闭的原因。这个字符串必须是不长于 123 字节的 UTF-8 文本（不是字符）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.onClose

该 方法 使用方式为 SocketTask.onClose(function listener)

- **功能说明：**监听 WebSocket 连接关闭事件。
- **参数及说明：**function listener，WebSocket 连接关闭事件的监听函数。

属性	类型	说明
code	number	一个数字值表示关闭连接的状态号，表示连接被关闭的原因
reason	string	一个可读的字符串，表示连接被关闭的原因

.onError

该 方法 使用方式为 SocketTask.onError(function listener)

- **功能说明：**监听 WebSocket 错误事件。
- **参数及说明：**function listener，WebSocket 错误事件的监听函数。

属性	类型	说明
errMsg	string	错误信息

.onMessage

该 方法 使用方式为 SocketTask.onMessage(function listener)

- **功能说明：**监听 WebSocket 接收到服务器的消息事件。
- **参数及说明：**function listener，WebSocket 接收到服务器的消息事件的监听函数。

属性	类型	说明
data	string/ArrayBuffer	服务器返回的消息

.onOpen

> 该 方法 使用方式为 SocketTask.onOpen(function listener)

- **功能说明：**监听 WebSocket 连接打开事件。
- **参数及说明：**function listener，WebSocket 连接打开事件的监听函数。

属性	类型	说明
header	Object	连接成功的 HTTP 响应 Header
profile	Object	网络请求过程中一些调试信息

- **profile 结构值**

结构属性	类型	说明
fetchStart	number	组件准备好使用 SOCKET 建立请求的时间，这发生在检查本地缓存之前
domainLookupStart	number	DNS 域名查询开始的时间，如果使用了本地缓存（即无 DNS 查询）或持久连接，则与 fetchStart 值相等
domainLookupEnd	number	DNS 域名查询完成的时间，如果使用了本地缓存（即无 DNS 查询）或持久连接，则与 fetchStart 值相等
connectStart	number	开始建立连接的时间，如果是持久连接，则与 fetchStart 值相等。注意如果在传输层发生了错误且重新建立连接，则这里显示的是新建立的连接开始的时间
connectEnd	number	完成建立连接的时间（完成握手），如果是持久连接，则与 fetchStart 值相等。注意如果在传输层发生了错误且重新建立连接，则这里显示的是新建立的连接完成的时间。注意这里握手结束，包括安全连接建立完成、SOCKS 授权通过
rtt	number	单次连接的耗时，包括 connect，tls
handshakeCost	number	握手耗时

cost	number	上层请求到返回的耗时

.send

> 该 方法 使用方式为 SocketTask.send(Object object)

- **功能说明：**通过 WebSocket 连接发送数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	string/ArrayBuffer	–	是	需要发送的内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

UDP 通信

最近更新时间：2023-11-10 15:39:24

createUDPSocket

> 该 API 使用方法为 UDPSocket wx.createUDPSocket()

- **功能说明：**创建一个 UDP Socket 实例。
- **返回值：**UDPSocket，一个 UDP Socket 实例。

UDPSocket

> **说明：**
> 一个 UDP Socket 实例，默认使用 IPv4 协议。

.bind

> 该 方法 使用方式为 number UDPSocket.bind(number port)

- **功能说明：**绑定一个系统随机分配的可用端口，或绑定一个指定的端口号。
- **参数及说明：**number port，指定要绑定的端口号，不传则返回系统随机分配的可用端口。
- **返回值：**number，绑定成功的端口号。
- **示例代码**

```
const udp = wx.createUDPSocket()
const port = udp.bind()
```

.close

> 该 方法 使用方式为 UDPSocket.close()

- **功能说明：**关闭 UDP Socket 实例，相当于销毁。在关闭之后，UDP Socket 实例不能再发送消息，每次调用 UDPSocket.send 将会触发错误事件，并且 message 事件回调函数也不会再也执行。
  在 UDPSocket 实例被创建后将被 Native 强引用，保证其不被 GC。在 UDPSocket.close 后将解除对其的强引用，让 UDPSocket 实例遵从 GC。

.connect

> 该 方法 使用方式为 UDPSocket.connect(Object object)

- **功能描述：**预先连接到指定的 IP 和 port，需要配合 write 方法一起使用。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
address	string	-	是	要发消息的地址
port	number	-	是	要发送消息的端口号

.onClose

> 该 方法 使用方式为 UDPSocket.onClose(function listener)

- **功能说明：**监听关闭事件。
- **参数及说明：**function listener，关闭事件的监听函数。

.offClose

> 该 方法 使用方式为 UDPSocket.offClose(function listener)

- **功能说明：**移除关闭事件的监听函数。
- **参数及说明：**function listener，onClose 传入的监听函数，不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

UDPSocket.onClose(listener)
UDPSocket.offClose(listener) // 需传入与监听时同一个的函数对象
```

.onError

> 该 方法 使用方式为 UDPSocket.onError(function listener)

- **功能说明：**监听错误事件。
- **参数及说明：**Object res 参数，function listener，错误事件的监听函数。

属性	类型	说明
errMsg	string	错误信息

.offError

> 该 方法 使用方式为 UDPSocket.offError(function listener)

- **功能说明：**移除错误事件的监听函数。
- **参数及说明：**function listener，onError 传入的监听函数，不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

UDPSocket.onError(listener)
UDPSocket.offError(listener) // 需传入与监听时同一个的函数对象
```

.onListening

> 该 方法 使用方式为 UDPSocket.onListening(function listener)

- **功能说明：**监听开始监听数据包消息的事件。
- **参数及说明：**function listener，开始监听数据包消息的事件的监听函数。

.offListening

> 该 方法 使用方式为 UDPSocket.offListening(function listener)

- **功能说明：**移除开始监听数据包消息的事件的监听函数。
- **参数及说明：**function listener，onListening 传入的监听函数，不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

UDPSocket.onListening(listener)
UDPSocket.offListening(listener) // 需传入与监听时同一个的函数对象
```

.onMessage

> 该 方法 使用方式为 UDPSocket.onMessage(function listener)

- **功能说明：**监听收到消息的事件。
- **参数及说明：**Object res 参数，function listener，收到消息的事件的监听函数。

属性	类型	说明
message	ArrayBuffer	收到的消息。消息长度需要小于 4096

remoteInfo	Object	发送端地址信息
localInfo	Object	接收端地址信息，2.18.0起支持

- **remoteInfo 结构属性**

结构属性	类型	说明
address	string	发送消息的 socket 的地址
family	string	使用的协议族，为 IPv4 或者 IPv6
port	number	端口号
size	number	message 的大小，单位：字节

- **localInfo 结构属性**

结构属性	类型	说明
address	string	接收消息的 socket 的地址
family	string	使用的协议族，为 IPv4 或者 IPv6
port	number	端口号

.offMessage

> 该 方法 使用方式为 UDPSocket.offMessage(function listener)

- **功能说明：**移除收到消息的事件的监听函数。
- **参数及说明：**function listener，onMessage 传入的监听函数，不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

UDPSocket.onMessage(listener)
UDPSocket.offMessage(listener) // 需传入与监听时同一个的函数对象
```

.send

> 该 方法 使用方式为 UDPSocket.send(Object object)

- **功能说明：**向指定的 IP 和 port 发送消息。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
address	string	–	是	要发消息的地址。在基础库2.9.3及以下版本必须是和本机同网段的 IP 地址，或安全域名列表内的域名地址；基础库2.9.3以上版本可以是任意 IP 和域名
port	number	–	是	要发送消息的端口号
message	string/ArrayBuffer	–	是	要发送的数据
offset	number	0	否	发送数据的偏移量，仅当 message 为 ArrayBuffer 类型时有效
length	number	message.byteLength	否	发送数据的长度，仅当 message 为 ArrayBuffer 类型时有效

- **示例代码**

```
const udp = wx.createUDPSocket()
 udp.bind()
 udp.send({
   address: '192.168.193.2',
   port: 8848,
   message: 'hello, how are you'
 })
```

.setTTL

> 该 方法 使用方式为 UDPSocket.setTTL(number ttl)

- **功能说明：**设置 IP_TTL 套接字选项，用于设置一个 IP 数据包传输时允许的最大跳步数。
- **参数及说明：**number ttl，ttl 参数可以是 0 到 255 之间。
- **示例代码**

```
const udp = wx.createUDPSocket()
udp.onListening(function () {
  udp.setTTL(64)
})
udp.bind()
```

.write

该 方法 使用方式为 UDPSocket.write()

功能说明：用法与 send 方法相同，如果没有预先调用 connect 则与 send 无差异。

> **说明：**
> 即使调用了 connect 也需要在本接口填入地址和端口参数。

TCP 通信

最近更新时间：2023-10-20 15:19:17

createTCPSocket

该 API 使用方法为 TCPSocket wx.createTCPSocket()

- **功能说明：**创建一个 TCP Socket 实例。
- **返回值：**一个 TCP Socket 实例。
- **链接限制**
  - 允许与局域网内的非本机 IP 通信；
  - 允许与配置过的服务器域名通信；
  - 禁止与以下端口号连接：1024 以下、1099、1433、1521、1719、1720、1723、2049、2375、3128、3306、3389、3659、4045、5060、5061、5432、5984、6379、6000、6566、7001、7002、8000-8100、8443、8888、9200、9300、10051、10080、11211、27017、27018、27019；
  - 每5分钟内最多创建20个 TCPSocket。

TCPSocket

说明：

- 一个 TCP Socket 实例，默认使用 IPv4 协议。
- 当 errCode 为 -2 时，errMsg 里应该会有相应的 errno ，开发者可以根据 errno 到 Linux 代码里的 errno-base.h 和 errno.h 中查看具体的报错信息。

.bindWifi

该 方法 使用方式为 TCPSocket.bindWifi(Object options)

- **功能说明：**将 TCP Socket 绑定到当前 Wi-Fi 网络，成功后会触发 onBindWifi 事件（仅 Android 支持）。
- **参数及说明：**Object options。

属性	类型	默认值	必填	说明
BSSID	string	-	是	当前 Wi-Fi 网络的 BSSID ，可通过 wx.getConnectedWifi 获取

- **示例代码**

```
const tcp = wx.createTCPSocket()
tcp.bindWifi({ BSSID: 'xxx' })
tcp.onBindWifi(() => {})
```

.close

> 该 方法 使用方式为 TCPSocket.close()

- **功能说明：**关闭连接。
- **示例代码**

```
const tcp = wx.createTCPSocket()
tcp.close()
```

.connect

> 该 方法 使用方式为 TCPSocket.connect(Object options)

- **功能说明：**在给定的套接字上启动连接。
- **参数及说明：**Object options。

属性	类型	默认值	必填	说明
address	string	-	是	套接字要连接的地址
port	number	-	是	套接字要连接的端口
timeout	number	2	否	套接字要连接的超时时间，默认为 2s

- **示例代码**

```
const tcp = wx.createTCPSocket()
tcp.connect({address: '192.168.193.2', port: 8848})
```

.onClose

> 该 方法 使用方式为 TCPSocket.onClose(function listener)

- **功能说明：**监听一旦 socket 完全关闭就发出该事件。
- **参数及说明：**function listener，一旦 socket 完全关闭就发出该事件的监听函数。

.offClose

该 方法 使用方式为 TCPSocket.offClose(function listener)

- **功能说明：**移除一旦 socket 完全关闭就发出该事件的监听函数。
- **参数及说明：**function listener，onClose 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

TCPSocket.onConnect(listener)
TCPSocket.offConnect(listener) // 需传入与监听时同一个的函数
```

.onConnect

该 方法 使用方式为 TCPSocket.onConnect(function listener)

- **功能说明：**监听当一个 socket 连接成功建立的时候触发该事件。
- **参数及说明：**function listener，当一个 socket 连接成功建立的时候触发该事件的监听函数。

.offConnect

该 方法 使用方式为 TCPSocket.offConnect(function listener)

- **功能说明：**移除当一个 socket 连接成功建立的时候触发该事件的监听函数。
- **参数及说明：**function listener，onConnect 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

TCPSocket.onConnect(listener)
TCPSocket.offConnect(listener) // 需传入与监听时同一个的函数对象
```

.onError

该 方法 使用方式为 TCPSocket.onError(function listener)

- **功能说明：**监听当错误发生时触发。
- **参数及说明：**function listener 的监听函数。

属性	类型	说明

errMsg	string	错误信息

.offError

该 方法 使用方式为 TCPSocket.offError(function listener)

- **功能说明：**移除当错误发生时触发的监听函数。
- **参数及说明：**function listener，onError 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

TCPSocket.onError(listener)
TCPSocket.offError(listener) // 需传入与监听时同一个的函数对象
```

.onMessage

该 方法 使用方式为 TCPSocket.onMessage(function listener)

- **功能说明：**监听当接收到数据的时触发该事件。
- **参数及说明：**Object res 参数，function listener，当接收到数据的时触发该事件的监听函数。

属性	类型	说明
message	ArrayBufffer	收到的消息
remoteInfo	Object	发送端地址信息
localInfo	Object	接收端地址信息

- **remoteInfo 结构属性**

结构属性	类型	说明
address	string	发送消息的 socket 的地址
family	string	使用的协议族，为 IPv4 或者 IPv6
port	number	端口号
size	number	message 的大小，单位：字节

- **localInfo 结构属性**

结构属性	类型	说明
address	string	发送消息的 socket 的地址
family	string	使用的协议族，为 IPv4 或者 IPv6
port	number	端口号

.offMessage

> 该 方法 使用方式为 TCPSocket.offMessage(function listener)

- **功能说明：**移除当接收到数据的时触发该事件的监听函数。
- **参数及说明：**function listener，onMessage 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

TCPSocket.onMessage(listener)
TCPSocket.offMessage(listener) // 需传入与监听时同一个的函数对象
```

.onBindWifi

> 该 方法 使用方式为 TCPSocket.onBindWifi(function listener)

- **功能说明：**监听当一个 socket 绑定当前 Wi-Fi 网络成功时触发该事件。
- **参数及说明：**function listener，当一个 socket 绑定当前 Wi-Fi 网络成功时触发该事件的监听函数。

.offBindWifi

> 该 方法 使用方式为 TCPSocket.offBindWifi(function listener)

- **功能说明：**移除当一个 socket 绑定当前 Wi-Fi 网络成功时触发该事件的监听函数。
- **参数及说明：**function listener，onBindWifi 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

TCPSocket.onBindWifi(listener)
TCPSocket.offBindWifi(listener) // 需传入与监听时同一个的函数对象
```

.write

该 方法 使用方式为 TCPSocket.write(string|ArrayBuffer data)

- **功能说明：**在 socket 上发送数据。
- **参数及说明：**string|ArrayBuffer data，要发送的数据。
- **示例代码**

```
const tcp = wx.createTCPSocket()
tcp.write('hello, how are you')
```

mDNS

最近更新时间：2023-10-20 15:19:18

stopLocalServiceDiscovery

该 API 使用方法为 wx.stopLocalServiceDiscovery(Object object)

- **功能说明：**停止搜索 mDNS 服务。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：**Object res。

属性	类型	说明
errMsg	string	错误信息，其合法值为 task not found：在当前没有处在搜索服务中的情况下调用 stopLocalServiceDiscovery

startLocalServiceDiscovery

该 API 使用方法为 wx.startLocalServiceDiscovery(Object object)

⚠ 注意：

- iOS 需要接入 TMFMiniAppExtMDNS 扩展库才能使用 mDNS 相关接口。
- wx.startLocalServiceDiscovery 是一个消耗性能的行为，开始 30 秒后会自动 stop 并执行 wx.onLocalServiceDiscoveryStop 注册的回调函数。
- 在调用 wx.startLocalServiceDiscovery 后，在这次搜索行为停止后才能发起下次 wx.startLocalServiceDiscovery。停止本次搜索行为的操作包括调用 wx.stopLocalServiceDiscovery 和30秒后系统自动 stop 本次搜索。

- **功能说明：**开始搜索局域网下的 mDNS 服务。搜索的结果会通过 wx.onLocalService* 事件返回。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
serviceType	string	–	是	要搜索的服务类型
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- object.fail 回调函数参数：Object res。

属性	类型	说明
errMsg	string	错误信息，合法值为 • invalid param：serviceType 为空 • scan task already exist：在当前 startLocalServiceDiscovery 发起的搜索未停止的情况下，再次调用 startLocalServiceDiscovery

- **示例代码**

```
wx.startLocalServiceDiscovery({
  // 当前手机所连的局域网下有一个 _http._tcp. 类型的服务
  serviceType: '_http._tcp.',
  success: console.log,
  fail: console.log
})
```

offLocalServiceResolveFail

> 该 API 使用方法为 wx.offLocalServiceResolveFail(function listener)

- **功能说明：**移除 mDNS 服务解析失败的事件的监听函数。
- **参数及说明：**function listener，onLocalServiceResolveFail 传入的监听函数。不传此参数则移除所有

监听函数。

- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onLocalServiceResolveFail(listener)
wx.offLocalServiceResolveFail(listener) // 需传入与监听时同一个的函数对象
```

onLocalServiceResolveFail

> 该 API 使用方法为 wx.onLocalServiceResolveFail(function listener)

- **功能说明：**监听 mDNS 服务解析失败的事件。
- **参数及说明：**Object res 参数，function listener，mDNS 服务解析失败的事件的监听函数。

属性	类型	说明
serviceType	string	服务的类型
serviceName	string	服务的名称

offLocalServiceLost

> 该 API 使用方法为 wx.offLocalServiceLost(function listener)

- **功能说明：**移除 mDNS 服务离开的事件的监听函数。
- **参数及说明：**function listener，onLocalServiceLost 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onLocalServiceLost(listener)
wx.offLocalServiceLost(listener) // 需传入与监听时同一个的函数对象
```

onLocalServiceLost

> 该 API 使用方法为 wx.onLocalServiceLost(function listener)

- **功能说明：**监听 mDNS 服务离开的事件。
- **参数及说明：**Object res 参数，function listener，mDNS 服务离开的事件的监听函数。

属性	类型	说明
serviceType	string	服务的类型
serviceName	string	服务的名称

offLocalServiceFound

> 该 API 使用方法为 wx.offLocalServiceFound(function listener)

- **功能说明：**移除 mDNS 服务发现的事件的监听函数。
- **参数及说明：**function listener，onLocalServiceFound 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onLocalServiceFound(listener)
wx.offLocalServiceFound(listener) // 需传入与监听时同一个的函数对象
```

onLocalServiceFound

> 该 API 使用方法为 wx.onLocalServiceFound(function listener)

- **功能说明：**监听 mDNS 服务发现的事件。
- **参数及说明：**Object res 参数，function listener，mDNS 服务发现的事件的监听函数。

属性	类型	说明
serviceType	string	服务的类型
serviceName	string	服务的名称
ip	string	服务的 ip 地址
port	number	服务的端口

offLocalServiceDiscoveryStop

> 该 API 使用方法为 wx.offLocalServiceDiscoveryStop(function listener)

- **功能说明：**移除 mDNS 服务停止搜索的事件的监听函数。

- **参数及说明：**function listener，onLocalServiceDiscoveryStop 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onLocalServiceDiscoveryStop(listener)
wx.offLocalServiceDiscoveryStop(listener) // 需传入与监听时同一个的函数对象
```

onLocalServiceDiscoveryStop

> 该 API 使用方法为 wx.onLocalServiceDiscoveryStop(function listener)

- **功能说明：**监听 mDNS 服务停止搜索的事件。
- **参数及说明：**function listener，mDNS 服务停止搜索的事件的监听函数。

数据缓存

最近更新时间：2024-02-26 10:34:51

setStorage

> 该 API 使用方法为 wx.setStorage(Object object)

- **功能说明：**将数据存储在本地缓存中指定的 key 中。会覆盖掉原来该 key 对应的内容。除非用户主动删除或因存储空间原因被系统清理，否则数据都一直可用。单个 key 允许存储的最大数据长度为1MB，所有数据存储上限为10MB 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
key	string	–	是	本地缓存中指定的 key
data	any	–	是	需要存储的内容。只支持原生类型、Date、及能够通过 `JSON.stringify` 序列化的对象
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.setStorage({
  key:"key",
  data:"value"
})
```

```
wx.setStorage({
  key: 'key',
  success (res) {
    console.log(res.data)
```

```
  }
})
```

setStorageSync

该 API 使用方法为 wx.setStorageSync(string key，any data)

> **说明：**
> storage 应只用来进行数据的持久化存储，不应用于运行时的数据传递或全局状态管理。启动过程中过多的同步读写存储，会显著影响启动耗时。

- **功能说明：**将数据存储在本地缓存中指定的 key 中。会覆盖掉原来该 key 对应的内容。除非用户主动删除或因存储空间原因被系统清理，否则数据都一直可用。单个 key 允许存储的最大数据长度为 1MB，所有数据存储上限为 10MB。
- **参数及说明：**string key，本地缓存中指定的 key；any data，需要存储的内容。只支持原生类型、Date、及能够通过 JSON.stringify 序列化的对象。
- **示例代码：**

```
try {
  wx.setStorageSync('key', 'value')
} catch (e) { }
```

revokeBufferURL

该 API 使用方法为 wx.revokeBufferURL(string url)

- **功能说明：**根据 URL 销毁存在内存中的数据。
- **参数及说明：**string url，需要销毁的二进制数据 URL。

removeStorage

该 API 使用方法为 wx.removeStorage(Object object)

- **功能说明：**从本地缓存中移除指定 key 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
key	strin	-	是	本地缓存中指定的 key

	g			
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.removeStorage({
  key: 'key',
  success (res) {
    console.log(res)
  }
})
```

```
try {
  wx.removeStorageSync('key')
} catch (e) {
  // Do something when catch error
}
```

removeStorageSync

> 该 API 使用方法为 wx.removeStorageSync(string key)

- **功能说明：** wx.removeStorage 的同步版本。
- **参数及说明：** string key，本地缓存中指定的 key。
- **示例代码：**

```
wx.removeStorage({
  key: 'key',
  success (res) {
    console.log(res)
  }
})
```

```
try {
  wx.removeStorageSync('key')
} catch (e) {
  // Do something when catch error
}
```

getStorage

> 该 API 使用方法为 wx.getStorage(Object object)

- **功能说明：**从本地缓存中异步获取指定 key 的内容。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
key	string	–	是	本地缓存中指定的 key
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

  - **Object.success 回调函数参数：**Object res。

属性	类型	说明
data	any	key 对应的内容

- **示例代码：**

```
wx.getStorage({
  key: 'key',
  success (res) {
    console.log(res.data)
  }
})
```

```
wx.getStorage({
  key: 'key',
  success (res) {
    console.log(res.data)
  }
})
```

createBufferURL

> 该 API 使用方法为 string wx.createBufferURL(ArrayBuffer|TypedArray buffer)

- **功能说明：**根据传入的 buffer 创建一个唯一的 URL 存在内存中 。
- **参数及说明：**ArrayBuffer|TypedArray buffer，需要存入内存的二进制数据。
- **返回值：**string。

getStorageSync

> 该 API 使用方法为 any wx.getStorageSync(string key)

- **功能说明：**从本地缓存中同步获取指定 key 的内容 。
- **参数及说明：**string key，本地缓存中指定的 key。
- **返回值：**any key，对应的内容。
- **示例代码：**

```
try {
  var value = wx.getStorageSync('key')
  if (value) {
    // Do something with return value
  }
} catch (e) {
  // Do something when catch error
}
```

getStorageInfo

> 该 API 使用方法为 wx.getStorageInfo(Object object)

- **功能说明：**异步获取当前 storage 的相关信息。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object object。

属性	类型	说明
keys	Array.<string>	当前 storage 中所有的 key
currentSize	number	当前占用的空间大小, 单位 KB
limitSize	number	限制的空间大小，单位 KB

- **示例代码：**

```
wx.getStorageInfo({
  success(res) {
    console.log(res.keys)
    console.log(res.currentSize)
    console.log(res.limitSize)
  }
})
```

```
wx.getStorageInfo({
  success(res) {
    console.log(res.keys)
    console.log(res.currentSize)
    console.log(res.limitSize)
  }
})
```

getStorageInfoSync

> 该 API 使用方法为 Object wx.getStorageInfoSync()

- **功能说明：** wx.getStorageInfo 的同步版本 。
- **返回值：** Object object。

属性	类型	说明
keys	Array.	当前 storage 中所有的 key
currentSize	number	当前占用的空间大小, 单位 KB
limitSize	number	限制的空间大小，单位 KB

- **示例代码：**

```
wx.getStorageInfo({
  success (res) {
    console.log(res.keys)
    console.log(res.currentSize)
    console.log(res.limitSize)
  }
})
```

```
try {
  const res = wx.getStorageInfoSync()
  console.log(res.keys)
  console.log(res.currentSize)
  console.log(res.limitSize)
} catch (e) {
  // Do something when catch error
}
```

clearStorage

> 该 API 使用方法为 wx.clearStorage(Object object)

- **功能说明：** 清理本地数据缓存 。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.clearStorage()
```

```
try {
  wx.clearStorageSync()
} catch(e) {
  // Do something when catch error
}
```

clearStorageSync

> 该 API 使用方法为 wx.clearStorageSync()

- **功能说明：** wx.clearStorage 的同步版本 。
- **示例代码：**

```
wx.clearStorage()
```

```
try {
  wx.clearStorageSync()
} catch(e) {
  // Do something when catch error
}
```

batchSetStorage

> 该 API 使用方法为 wx.batchSetStorage(Object object)

- **功能说明：** 将数据批量存储在本地缓存中指定的 key 中。会覆盖掉原来该 key 对应的内容。除非用户主动删除或因存储空间原因被系统清理，否则数据都一直可用。单个 key 允许存储的最大数据长度为 1MB，所有数据存储上限为 10MB。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
kvList	Array	–	是	[{ key, value }]
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.setStorage({
  key:"key",
  data:"value"
})
```

```
// 开启加密存储
wx.batchSetStorage({
  kvList: [{
    key: 'key',
    value: 'value',
  }],
})
```

batchSetStorageSync

> 该 API 使用方法为 Array.<any> wx.batchGetStorageSync(Array.<string> keyList)

- **功能说明：**将数据批量存储在本地缓存中指定的 key 中。会覆盖掉原来该 key 对应的内容。除非用户主动删除或因存储空间原因被系统清理，否则数据都一直可用。单个 key 允许存储的最大数据长度为 1MB，所有数据存储上限为 10MB。
- **参数及说明：**Array.<Object> kvList。

属性	类型	默认值	必填	说明
key	string	–	是	key 本地缓存中指定的 key。

value	any	–	是	data 需要存储的内容。只支持原生类型、Date、及能够通过 JSON.stringify 序列化的对象。

- **示例代码**

```
try {
  wx.batchSetStorageSync([{key: 'key', value: 'value'}])
} catch (e) { }
```

batchGetStorage

该 API 使用方法为 wx.batchGetStorage(Object object)

- **功能说明：**从本地缓存中异步批量获取指定 key 的内容。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
keyList	Array.<string>	–	是	本地缓存中指定的 keyList
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.batchGetStorage({
  keyList: ['key'],
  success (res) {
    console.log(res)
  }
})
```

batchGetStorageSync

该 API 使用方法为 Array.<any> wx.batchGetStorageSync(Array.<string> keyList)

- **功能说明：**从本地缓存中同步批量获取指定 key 的内容。
- **参数及说明：**Array.<string> keyList，本地缓存中指定的 key 数组。
- **返回值：**Array.<any>，key对应的内容。
- **示例代码**

```
//对于多个key的读取，批量读取在性能上优于多次 getStorageSync 读取
try {
  var valueList = wx.batchGetStorageSync(['key'])
  if (valueList) {
    // Do something with return value
  }
} catch (e) {
  // Do something when catch error
}
```

数据分析

最近更新时间：2023-11-10 15:39:29

reportEvent

> 该 API 使用方法为 wx.reportEvent(string eventId, object data)

- **功能说明：**事件上报。
- **参数及说明：**
  - string eventId：在 mp 实验系统中设置的事件英文名。
  - object data：可被 JSON.stringify 的对象，将一起上报至系统。

画布

画布概要

最近更新时间：2023-11-10 15:39:25

createOffscreenCanvas

> 该 API 使用方法为 OffscreenCanvas wx.createOffscreenCanvas(object object, number width, number height, Object this)

- **功能说明：**创建离屏 canvas 实例。
- **参数及说明：**
  - number width：画布宽度。
  - number height：画布高度。
  - object object

属性	类型	默认值	必填	说明
type	string	webgl	否	创建的离屏 canvas 类型
width	number		否	画布宽度
height	number		否	画布高度
compInst	Component		否	在自定义组件下，当前组件实例的 this

createCanvasContext

> 该 API 使用方法为 CanvasContext wx.createCanvasContext(string canvasId, Object this)

- **功能说明：**创建 canvas 的绘图上下文 canvasContext 对象。
- **参数及说明：**
  - string canvasId：要获取上下文的 `<canvas>` 组件 `canvas-id` 属性。
  - Object this：在自定义组件下，当前组件实例的 this，表示在这自定义组件下查找拥有 canvas-id 的 `<canvas>`，如果省略，则不在任何自定义组件内查找。
- **返回值：**canvasContext。

canvasToTempFilePath

> 该 API 使用方法为 wx.canvasToTempFilePath(Object object, Object this)

- **功能说明：**把当前画布指定区域的内容导出，生成指定大小的图片，在 `draw()` 回调里调用该方法才能保证图片导出成功。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
x	number	0	否	指定的画布区域的左上角横坐标
y	number	0	否	指定的画布区域的左上角纵坐标
width	number	canvas宽度-x	否	指定的画布区域的宽度
height	number	canvas高度-y	否	指定的画布区域的高度
destWidth	number	width*屏幕像素密度	否	输出的图片的宽度
destHeight	number	height*屏幕像素密度	否	输出的图片的高度
canvasId	string	-	是	画布标识，传入 `<canvas>` 组件的 canvas-id
fileType	string	png	否	目标文件的类型
quality	number	-	是	图片的质量，目前仅对 jpg 有效。取值范围为 (0, 1]，不在范围内时当作1.0 处理。
success	function	-	否	接口调用成功的回调函数
fail	fun	-	否	接口调用失败的回调函数

	ctio n			
compl ete	fun ctio n	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- object.fileType 的合法值

值	说明
jpg	jpg 图片
png	png 图片

canvasPutImageData

该 API 使用方法为 wx.canvasPutImageData(Object object, Object this)

- **功能说明：**将像素数据绘制到画布，在自定义组件下，第二个参数传入自定义组件实例 this，以操作组件内 `canvas` 组件。
- **参数1：**Object object。

属性	类型	默认值	必填	说明
canv asId	string	–	是	画布标识，传入 `<canvas>` 组件的 canvas-id 属性
data	Uint8Clam pedArray	–	是	图像像素点数据，一维数组，每四项表示一个像素点的 rgba
x	number	–	是	源图像数据在目标画布中的位置偏移量（x 轴方向的偏移量）
y	number	–	是	源图像数据在目标画布中的位置偏移量（y 轴方向的偏移量）
width	number	–	是	源图像数据矩形区域的宽度
heigh t	number	–	是	源图像数据矩形区域的高度
succ ess	function	–	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **参数2：** Object this，在自定义组件下，当前组件实例的 this，以操作组件内 `<canvas>` 组件。
- **示例代码：**

```
const data = new Uint8ClampedArray([255, 0, 0, 1])
wx.canvasPutImageData({
  canvasId: 'myCanvas',
  x: 0,
  y: 0,
  width: 1,
  data,
  success(res) {}
})
```

canvasGetImageData

> 该 API 使用方法为 wx.canvasGetImageData(Object object, Object this)

- **功能说明：** 获取 canvas 区域隐含的像素数据。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
canvasId	string	–	是	画布标识，传入 `<canvas>` 组件的 `canvas-id` 属性
x	number	–	是	将要被提取的图像数据矩形区域的左上角横坐标
y	number	–	是	将要被提取的图像数据矩形区域的左上角纵坐标
width	number	–	是	将要被提取的图像数据矩形区域的宽度
height	number	–	是	将要被提取的图像数据矩形区域的高度
success	function	–	否	接口调用成功的回调函数

fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
width	number	图像数据矩形的宽度
height	number	图像数据矩形的高度
data	Uint8ClampedArray	图像像素点数据，一维数组，每四项表示一个像素点的 rgba

- **object.success 回调函数参数：**Object this，在自定义组件下，当前组件实例的 this，以操作组件内 `<canvas>` 组件。
- **示例代码：**

```
wx.canvasGetImageData({
  canvasId: 'myCanvas',
  x: 0,
  y: 0,
  width: 100,
```

Image

- **功能说明：**图片对象。
- **参数及说明**

属性	类型	说明
src	string	图片的 URL
width	number	图片的真实宽度
height	number	图片的真实高度
referrerPolicy	string	origin：发送完整的referrer；no-referrer：不发送。格式固定为 https://servicewechat.com/{appid}/{version}/page-frame.html

		，其中 {appid} 为小程序的 appid，{version} 为小程序的版本号，版本号为 0 表示为开发版、体验版以及审核版本，版本号为 devtools 表示为开发者工具，其余为正式版本；
onload	function	图片加载完成后触发的回调函数
onerror	function	图片加载发生错误后触发的回调函数

ImageData

- **功能说明：** ImageData 对象。
- **参数及说明**

属性	类型	说明
width	number	使用像素描述 ImageData 的实际宽度
height	number	使用像素描述 ImageData 的实际高度
data	Uint8ClampedArray	一维数组，包含以 RGBA 顺序的数据，数据使用 0 至 255（包含）的整数表示

canvasGradient

- **功能说明：** 渐变对象。

.addColorStop

> 该 方法 使用方式为 CanvasGradient.addColorStop(number stop, Color color)

- **功能说明：** 添加颜色的渐变点。小于最小 stop 的部分会按最小 stop 的 color 来渲染，大于最大 stop 的部分会按最大 stop 的 color 来渲染，相当于CanvasGradient.addColorStop(number stop, Color color)。
- **参数1：** number stop，表示渐变中开始与结束之间的位置，范围0-1。
- **参数2：** Color color，渐变点的颜色。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

// Create circular gradient
const grd = ctx.createLinearGradient(30, 10, 120, 10)
grd.addColorStop(0, 'red')
```

```
grd.addColorStop(0.16, 'orange')
grd.addColorStop(0.33, 'yellow')
grd.addColorStop(0.5, 'green')
grd.addColorStop(0.66, 'cyan')
grd.addColorStop(0.83, 'blue')
grd.addColorStop(1, 'purple')

// Fill with gradient
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```

RenderingContext

- **功能说明：** Canvas 绘图上下文。
  - 通过 Canvas.getContext('2d') 接口可以获取 CanvasRenderingContext2D 对象，实现了 HTML Canvas 2D Context 定义的属性、方法。
  - 通过 Canvas.getContext('webgl') 或 OffscreenCanvas.getContext('webgl') 接口可以获取 WebGLRenderingContext 对象，实现了 WebGL 1.0 定义的所有属性、方法、常量。
  - CanvasRenderingContext2D 的 drawImage 方法 2.10.0 起支持传入通过 SelectorQuery 获取的 video 对象，2.29.0 起支持传入开启了自定义渲染的 LivePusherContext 对象。

CanvasContext

详情请见 canvasContext。

OffscreenCanvas

详情请见 OffscreenCanvas。

Path2D

详情请见 Path2D。

Canvas

详情请见 Canvas。

Color

详情请见 Color。

canvasContext

最近更新时间：2023-12-06 15:14:27

功能说明：canvas 组件的绘图上下文。

属性说明

属性	类型	说明
fillStyle	String	填充颜色。用法同 CanvasContext.setFillStyle()
strokeStyle	String	边框颜色。用法同 CanvasContext.setFillStyle()
font	String	-
globalCompositeOperation	String	在绘制新形状时，应用的合成操作的类型。目前 Android 版本只适用于 fill 填充块的合成，用于 stroke 线段的合成效果都是 source-over。 目前支持的操作有： • Android：xor, source-over, source-atop, destination-out, lighter, overlay, darken, lighten, hard-light • iOS：xor, source-over, source-atop, destination-over, destination-out, lighter, multiply, overlay, darken, lighten, color-dodge, color-burn, hard-light, soft-light, difference, exclusion, saturation, luminosity
shadowOffsetX	Number	阴影相对于形状在水平方向的偏移
shadowOffsetY	Number	阴影相对于形状在竖直方向的偏移
shadowColor	Number	阴影的颜色
shadowBlur	Number	阴影的模糊级别
lineWidth	Number	线条的宽度。用法同 CanvasContext.setLineWidth()
lineCap	Number	线条的端点样式。用法同 CanvasContext.setLineCap()
lineJoin	Numb	线条的交点样式。用法同 CanvasContext.setLineJoin()

	er	
miterLimit	Number	最大斜接长度。用法同 CanvasContext.setMiterLimit()
lineDashOffset	Number	虚线偏移量，初始值为0
globalAlpha	Number	全局画笔透明度。范围0-1，0表示完全透明，1表示完全不透明

方法集

arc

> 该 方法 使用方式为 CanvasContext.arc(number x, number y, number r, number sAngle, number eAngle, boolean counterclockwise)

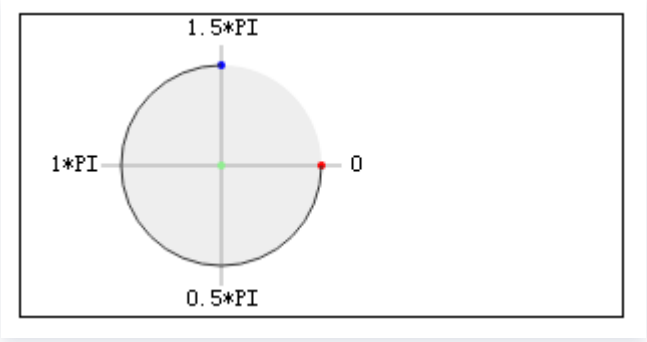
- **功能说明：**创建一条弧线。圆的创建可以指定起始弧度为0，终止弧度为2 * Math.PI。用 stroke 或者 fill 方法来在 canvas 中画弧线。
- **参数及说明：**number x，圆心的 x 坐标；number y，圆心的 y 坐标；number r，圆的半径；number sAngle，起始弧度，单位弧度（在3点钟方向）；number eAngle，终止弧度；number counterclockwise，弧度的方向是否是逆时针。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

// Draw coordinates
ctx.arc(100, 75, 50, 0, 2 * Math.PI)
ctx.setFillStyle('#EEEEEE')
ctx.fill()

ctx.beginPath()
ctx.moveTo(40, 75)
ctx.lineTo(160, 75)
ctx.moveTo(100, 15)
ctx.lineTo(100, 135)
ctx.setStrokeStyle('#AAAAAA')
ctx.stroke()

ctx.setFontSize(12)
ctx.setFillStyle('black')
ctx.fillText('0', 165, 78)
ctx.fillText('0.5*PI', 83, 145)
```

```
ctx.fillText('1*PI', 15, 78)
ctx.fillText('1.5*PI', 83, 10)

// Draw points
ctx.beginPath()
ctx.arc(100, 75, 2, 0, 2 * Math.PI)
ctx.setFillStyle('lightgreen')
ctx.fill()

ctx.beginPath()
ctx.arc(100, 25, 2, 0, 2 * Math.PI)
ctx.setFillStyle('blue')
ctx.fill()

ctx.beginPath()
ctx.arc(150, 75, 2, 0, 2 * Math.PI)
ctx.setFillStyle('red')
ctx.fill()

// Draw arc
ctx.beginPath()
ctx.arc(100, 75, 50, 0, 1.5 * Math.PI)
ctx.setStrokeStyle('#333333')
ctx.stroke()

ctx.draw()
```

针对 arc(100, 75, 50, 0, 1.5 * Math.PI) 的三个关键坐标如下：

- 绿色：圆心 (100, 75)
- 红色：起始弧度 (0)
- 蓝色：终止弧度 (1.5 * Math.PI)



arcTo

> 该 方法 使用方式为 CanvasContext.arcTo(number x1, number y1, number x2, number y2, number radius)

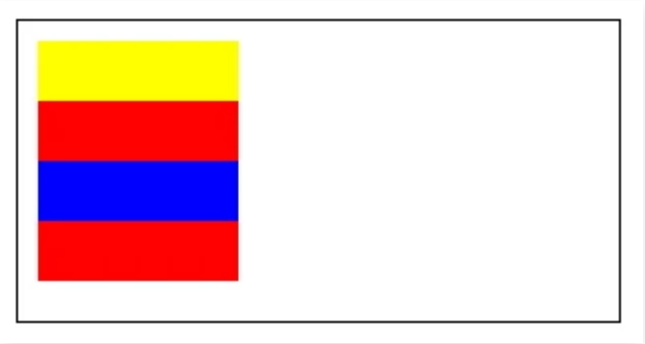
- **功能说明：**根据控制点和半径绘制圆弧路径。
- **参数及说明：**number x1，第一个控制点的 x 轴坐标；number y1，第一个控制点的 y 轴坐标；number x2，第二个控制点的 x 轴坐标；number y2，第二个控制点的 y 轴坐标；number radius，圆弧的半径。

beginPath

> 该 方法 使用方式为 CanvasContext.beginPath()

- **功能说明：**开始创建一个路径。需要调用 fill 或者 stroke 才会使用路径进行填充或描边。
  - 在最开始的时候相当于调用了一次 beginPath。
  - 同一个路径内的多次 setFillStyle、setStrokeStyle、setLineWidth 等设置，以最后一次设置为准。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
// begin path
ctx.rect(10, 10, 100, 30)
ctx.setFillStyle('yellow')
ctx.fill()

// begin another path
ctx.beginPath()
ctx.rect(10, 40, 100, 30)

// only fill this rect, not in current path
ctx.setFillStyle('blue')
ctx.fillRect(10, 70, 100, 30)

ctx.rect(10, 100, 100, 30)

// it will fill current path
ctx.setFillStyle('red')
ctx.fill()
ctx.draw()
```



bezierCurveTo

> 该 方法 使用方式为 CanvasContext.bezierCurveTo(number cp1x, number cp1y, number cp2x, number cp2y, number x, number y)

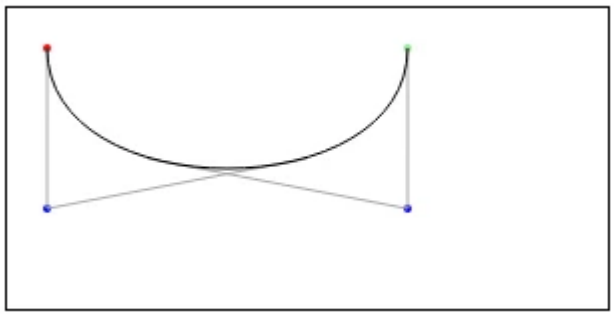
- **功能说明：**创建三次方贝塞尔曲线路径。曲线的起始点为路径中前一个点。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

// Draw points
ctx.beginPath()
ctx.arc(20, 20, 2, 0, 2 * Math.PI)
ctx.setFillStyle('red')
ctx.fill()

ctx.beginPath()
ctx.arc(200, 20, 2, 0, 2 * Math.PI)
ctx.setFillStyle('lightgreen')
ctx.fill()

ctx.beginPath()
ctx.arc(20, 100, 2, 0, 2 * Math.PI)
ctx.arc(200, 100, 2, 0, 2 * Math.PI)
ctx.setFillStyle('blue')
ctx.fill()

ctx.setFillStyle('black')
ctx.setFontSize(12)

// Draw guides
ctx.beginPath()
ctx.moveTo(20, 20)
ctx.lineTo(20, 100)
```

```
ctx.lineTo(150, 75)

ctx.moveTo(200, 20)
ctx.lineTo(200, 100)
ctx.lineTo(70, 75)
ctx.setStrokeStyle('#AAAAAA')
ctx.stroke()

// Draw quadratic curve
ctx.beginPath()
ctx.moveTo(20, 20)
ctx.bezierCurveTo(20, 100, 200, 100, 200, 20)
ctx.setStrokeStyle('black')
ctx.stroke()

ctx.draw()
```

针对 moveTo(20, 20) bezierCurveTo(20, 100, 200, 100, 200, 20) 的三个关键坐标如下：

- 红色：起始点(20, 20)
- 蓝色：两个控制点(20, 100) (200, 100)
- 绿色：终止点(200, 20)



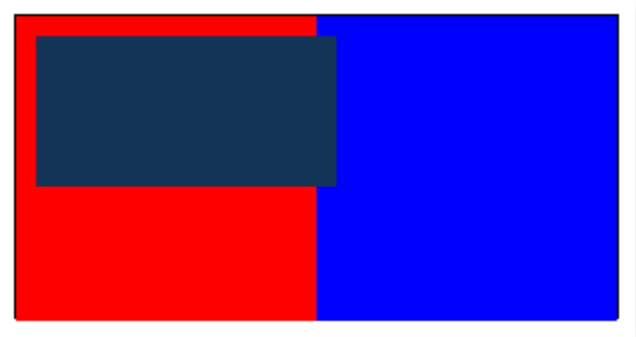
clearRect

> 该 方法 使用方式为 CanvasContext.clearRect(number x, number y, number width, number height)

- **功能说明：**清除画布上在该矩形区域内的内容。
- **参数及说明：**number x，矩形路径左上角的横坐标；number y，矩形路径左上角的纵坐标；number width，矩形路径的宽度；number height，矩形路径的高度。
- **示例代码：**clearRect 并非画一个白色的矩形在地址区域，而是清空，为了有直观感受，对 canvas 加了一层背景色。

```
<canvas canvas-id="myCanvas" style="border: 1px solid; background: #123456;"
/>
```

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setFillStyle('red')
ctx.fillRect(0, 0, 150, 200)
ctx.setFillStyle('blue')
ctx.fillRect(150, 0, 150, 200)
ctx.clearRect(10, 10, 150, 75)
ctx.draw()
```



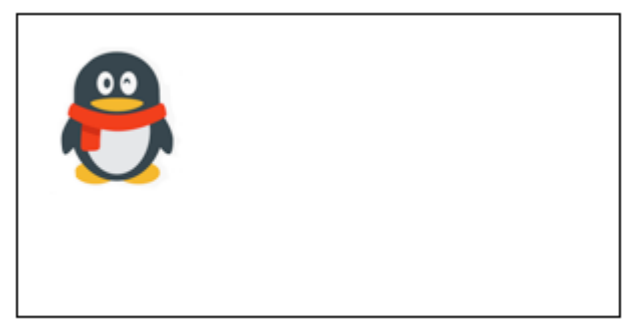
clip

> 该 方法 使用方式为 CanvasContext.clip()

- **功能说明：从原始画布中剪切任意形状和尺寸。一旦剪切了某个区域，则所有之后的绘图都会被限制在被剪切的区域内（不能访问画布上的其他区域）。可以在使用 clip 方法前通过使用 save 方法对当前画布区域进行保存，并在以后的任意时间通过 restore 方法对其进行恢复。**
- **示例代码**

```
const ctx = wx.createCanvasContext('myCanvas')

wx.downloadFile({
  url: 'https://is5.mzstatic.com/image/thumb/Purple128/v4/75/3b/90/753b907c-
b7fb-5877-215a-759bd73691a4/source/50x50bb.jpg',
  success(res) {
    ctx.save()
    ctx.beginPath()
    ctx.arc(50, 50, 25, 0, 2 * Math.PI)
    ctx.clip()
    ctx.drawImage(res.tempFilePath, 25, 25)
```

```
    ctx.restore()
    ctx.draw()
  }
})
```

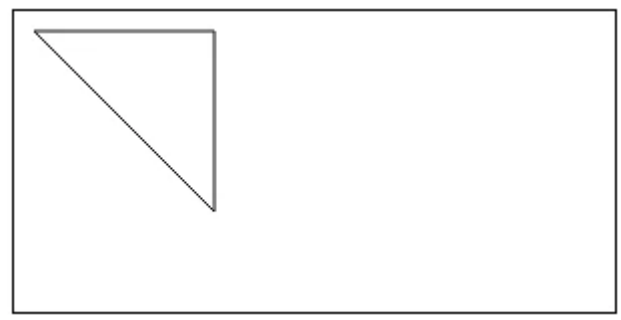


closePath

> 该 方法 使用方式为 CanvasContext.closePath()

- **功能说明：**关闭一个路径。会连接起点和终点。如果关闭路径后没有调用 fill 或者 stroke 并开启了新的路径，那之前的路径将不会被渲染。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.moveTo(10, 10)
ctx.lineTo(100, 10)
ctx.lineTo(100, 100)
ctx.closePath()
ctx.stroke()
ctx.draw()
```



```
const ctx = wx.createCanvasContext('myCanvas')
// begin path
```

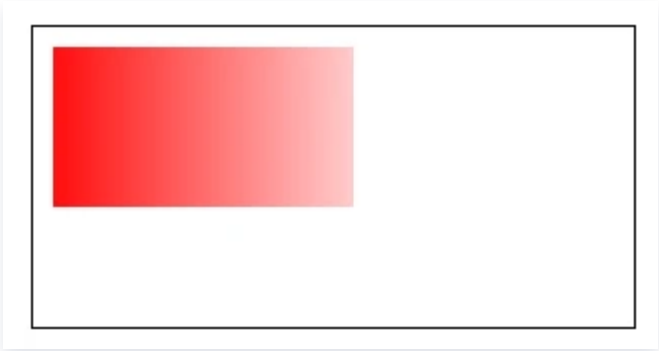
```
ctx.rect(10, 10, 100, 30)
ctx.closePath()

// begin another path
ctx.beginPath()
ctx.rect(10, 40, 100, 30)

// only fill this rect, not in current path
ctx.setFillStyle('blue')
ctx.fillRect(10, 70, 100, 30)

ctx.rect(10, 100, 100, 30)

// it will fill current path
ctx.setFillStyle('red')
ctx.fill()
ctx.draw()
```

createLinearGradient

> 该 方法 使用方式为 CanvasGradient CanvasContext.createLinearGradient(number x0, number y0, number x1, number y1)

- **功能说明：**创建一个线性的渐变颜色。返回的 CanvasGradient 对象需要使用 CanvasGradient.addColorStop() 来指定渐变点，至少要两个。
- **参数及说明：**number x0，起点的 x 坐标；number y0，起点的 y 坐标；number x1，终点的 x 坐标；number y1，终点的 y 坐标。
- **返回值：**CanvasGradient 。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

// Create linear gradient
const grd = ctx.createLinearGradient(0, 0, 200, 0)
grd.addColorStop(0, 'red')
grd.addColorStop(1, 'white')

// Fill with gradient
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```

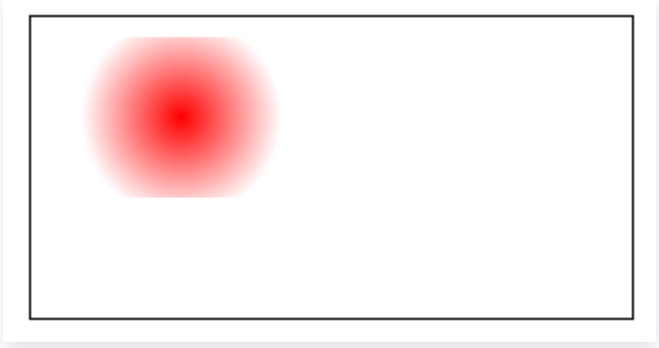


createCircularGradient

> 该 方法 使用方式为 CanvasGradient CanvasContext.createCircularGradient(number x, number y, number r)

- **功能说明：**创建一个圆形的渐变颜色。起点在圆心，终点在圆环。返回的 CanvasGradient 对象需要使用 CanvasGradient.addColorStop() 来指定渐变点，至少要两个。
- **参数及说明：**number x，圆心的 x 坐标；number y，圆心的 y 坐标；number r，圆的半径。
- **返回值：**CanvasGradient 。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

// Create circular gradient
const grd = ctx.createCircularGradient(75, 50, 50)
grd.addColorStop(0, 'red')
grd.addColorStop(1, 'white')

// Fill with gradient
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```



createPattern

- **功能说明：**对指定的图像创建模式的方法，可在指定的方向上重复元图像。**使用方法为** CanvasContext.createPattern(string image, string repetition)。
- **参数及说明：**string image，重复的图像源，支持代码包路径和本地临时路径 (本地路径)。

值	说明
repeat	水平竖直方向都重复
repeat-x	水平方向重复
repeat-y	竖直方向重复
no-repeat	不重复

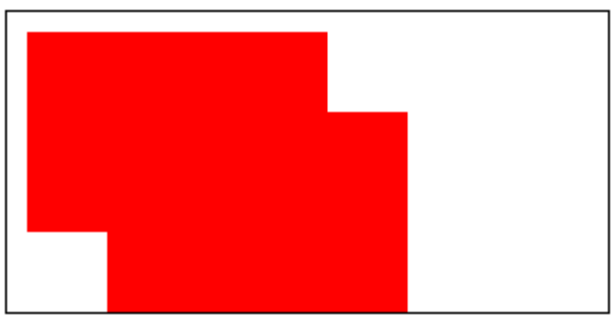
- **示例代码：**

```
constctx=wx.createCanvasContext('myCanvas')
constpattern=ctx.createPattern('/path/to/image','repeat-x')
ctx.fillStyle=pattern
ctx.fillRect(0,0,300,150)
ctx.draw()
```

draw

- **功能说明：**将之前在绘图上下文中的描述（路径、变形、样式）画到 canvas 中，**使用方法为** CanvasContext.draw(boolean reserve, function callback)。
- **参数及说明：**
  - boolean reserve：本次绘制是否接着上一次绘制。即 reserve 参数为 false，则在本次调用绘制之前 native 层会先清空画布再继续绘制；若 reserve 参数为 true，则保留当前画布上的内容，本次调用 drawCanvas 绘制的内容覆盖在上面，默认 false。
  - function callback：绘制完成后执行的回调函数
- **示例代码：**第二次 draw() reserve 为 true。所以保留了上一次的绘制结果，在上下文设置的 fillStyle 'red' 也变成了默认的 'black'。

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 100)
ctx.draw()
ctx.fillRect(50, 50, 150, 100)
ctx.draw(true)
```

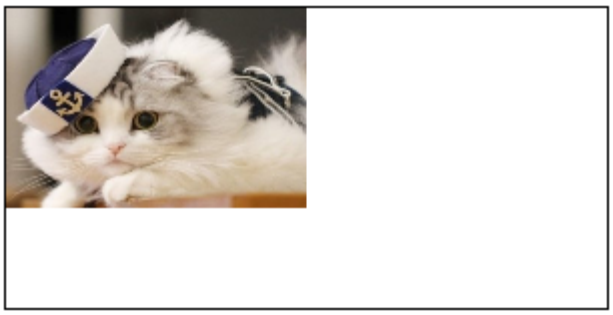


drawImage

> 该 方法 使用方式为 CanvasContext.drawImage(string imageResource, number sx, number sy, number sWidth, number sHeight, number dx, number dy, number dWidth, number dHeight)

- **功能说明：**绘制图像到画布。
- **参数及说明：**string imageResource，所要绘制的图片资源；number sx，源图像的矩形选择框的左上角 x 坐标；number sy，源图像的矩形选择框的左上角 y 坐标；number sWidth，源图像的矩形选择框的宽度；number sHeight，源图像的矩形选择框的高度；number dx，图像的左上角在目标 canvas 上 x 轴的位置；number dy，图像的左上角在目标 canvas 上 y 轴的位置；number dWidth，在目标画布上绘制图像的宽度，允许对绘制的图像进行缩放；number dHeight，在目标画布上绘制图像的高度，允许对绘制的图像进行缩放。
- **示例代码：**有如下三个版本的写法
  - drawImage(imageResource, dx, dy)
  - drawImage(imageResource, dx, dy, dWidth, dHeight)
  - drawImage(imageResource, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)

```
const ctx = wx.createCanvasContext('myCanvas')

wx.chooseImage({
 success(res) {
  ctx.drawImage(res.tempFilePaths[0], 0, 0, 150, 100)
  ctx.draw()
 }
})
```



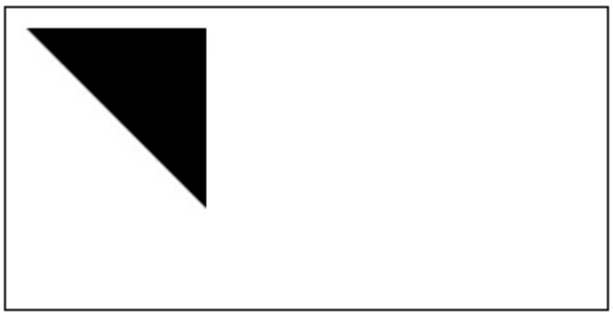
fill

> 该 方法 使用方式为 CanvasContext.fill()

- **功能说明：**对当前路径中的内容进行填充。默认的填充色为黑色。
- **示例代码：**如果当前路径没有闭合，fill() 方法会将起点和终点进行连接，然后填充。

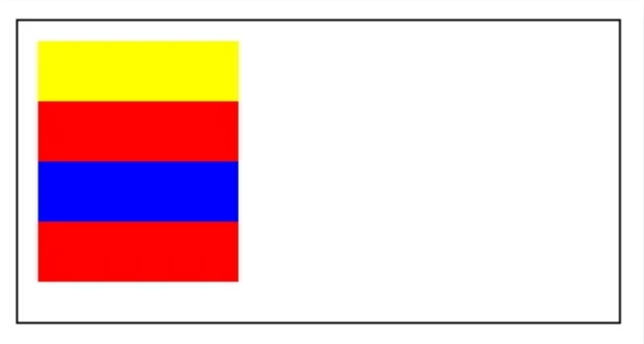
```
const ctx = wx.createCanvasContext('myCanvas')
ctx.moveTo(10, 10)
ctx.lineTo(100, 10)
ctx.lineTo(100, 100)
ctx.fill()
ctx.draw()
```

fill() 填充的路径是从 beginPath() 开始计算，但是不会将 fillRect() 包含进去。



```
const ctx = wx.createCanvasContext('myCanvas')
// begin path
ctx.rect(10, 10, 100, 30)
ctx.setFillStyle('yellow')
ctx.fill()

// begin another path
ctx.beginPath()
ctx.rect(10, 40, 100, 30)
```

```
// only fill this rect, not in current path
ctx.setFillStyle('blue')
ctx.fillRect(10, 70, 100, 30)

ctx.rect(10, 100, 100, 30)

// it will fill current path
ctx.setFillStyle('red')
ctx.fill()
ctx.draw()
```

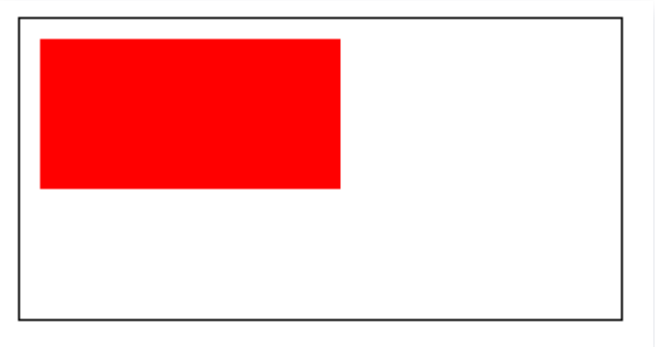


fillRect

> 该 方法 使用方式为 CanvasContext.fillRect(number x, number y, number width, number height)

- **功能说明：**填充一个矩形。用 setFillStyle 设置矩形的填充色，如果没设置默认是黑色。
- **参数及说明：**number x，矩形路径左上角的横坐标；number y，矩形路径左上角的纵坐标；number width，矩形路径的宽度；number height，矩形路径的高度。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 75)
ctx.draw()
```



fillText

> 该 方法 使用方式为 CanvasContext.fillText(string text, number x, number y, number maxWidth)

- **功能说明：**在画布上绘制被填充的文本。
- **参数及说明：**string text，在画布上输出的文本；number x，绘制文本的左上角 x 坐标位置；number y，绘制文本的左上角 y 坐标位置；number maxWidth，需要绘制的最大宽度，可选。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
```

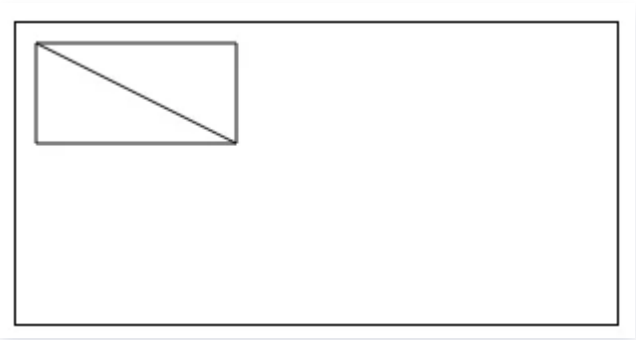
```
const ctx = wx.createCanvasContext('myCanvas')

ctx.setFontSize(20)
ctx.fillText('Hello', 20, 20)
ctx.fillText('MINA', 100, 100)

ctx.draw()
```

lineTo

> 该 方法 使用方式为 CanvasContext.lineTo(number x, number y)

- **功能说明：**增加一个新点，然后创建一条从上次指定点到目标点的线，用 stroke 方法来画线条。
- **参数及说明：**number x，目标位置的 x 坐标；number y，目标位置的 y 坐标。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.moveTo(10, 10)
ctx.rect(10, 10, 100, 50)
ctx.lineTo(110, 60)
ctx.stroke()
ctx.draw()
```



measureText

该 方法 使用方式为 Object CanvasContext.measureText(string text)

- **功能说明：**测量文本尺寸信息。目前仅返回文本宽度，同步接口。
- **参数及说明：**
  - string image：重复的图像源，支持代码包路径和本地临时路径 (本地路径)。
  - string repetition：如何重复图像。
  - repetition 的合法值

属性	类型	说明
width	number	文本的宽度

- **示例代码：**

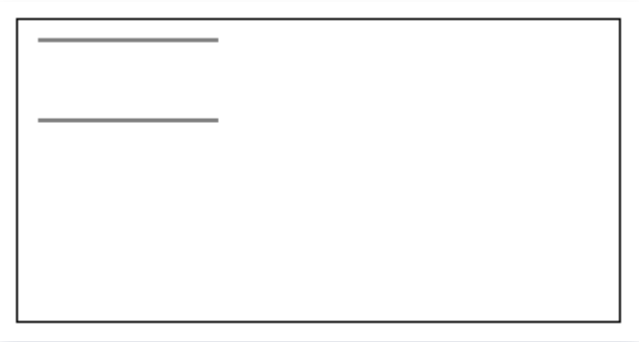
```
const ctx = wx.createCanvasContext('myCanvas')
ctx.font = 'italic bold 20px cursive'
const metrics = ctx.measureText('Hello World')
console.log(metrics.width)
```

moveTo

> 该 方法 使用方式为 CanvasContext.moveTo(number x, number y)

- **功能说明：**把路径移动到画布中的指定点，不用创建线条，利用 stroke 方法来画线条。
- **参数及说明：**number x，目标位置的 x 坐标；number y，目标位置的 y 坐标。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.moveTo(10, 10)
ctx.lineTo(100, 10)

ctx.moveTo(10, 50)
ctx.lineTo(100, 50)
ctx.stroke()
ctx.draw()
```



quadraticCurveTo

> 该 方法 使用方式为 CanvasContext.quadraticCurveTo(number cpx, number cpy, number x, number y)

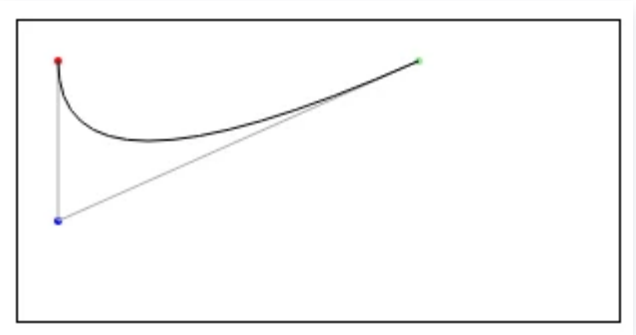
- **功能说明：**创建二次贝塞尔曲线路径。曲线的起始点为路径中前一个点。
- **参数及说明：**number cpx，贝塞尔控制点的 x 坐标；number cpy，贝塞尔控制点的 y 坐标；number x，结束点的 x 坐标；number y，结束点的 y 坐标。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

// Draw points
ctx.beginPath()
ctx.arc(20, 20, 2, 0, 2 * Math.PI)
```

```
ctx.setFillStyle('red')
ctx.fill()

ctx.beginPath()
ctx.arc(200, 20, 2, 0, 2 * Math.PI)
ctx.setFillStyle('lightgreen')
ctx.fill()

ctx.beginPath()
ctx.arc(20, 100, 2, 0, 2 * Math.PI)
ctx.setFillStyle('blue')
ctx.fill()

ctx.setFillStyle('black')
ctx.setFontSize(12)

// Draw guides
ctx.beginPath()
ctx.moveTo(20, 20)
ctx.lineTo(20, 100)
ctx.lineTo(200, 20)
ctx.setStrokeStyle('#AAAAAA')
ctx.stroke()

// Draw quadratic curve
ctx.beginPath()
ctx.moveTo(20, 20)
ctx.quadraticCurveTo(20, 100, 200, 20)
ctx.setStrokeStyle('black')
ctx.stroke()

ctx.draw()
```

针对 moveTo(20, 20) quadraticCurveTo(20, 100, 200, 20) 的三个关键坐标如下：

- 红色：起始点(20, 20)
- 蓝色：控制点(20, 100)
- 绿色：终止点(200, 20)

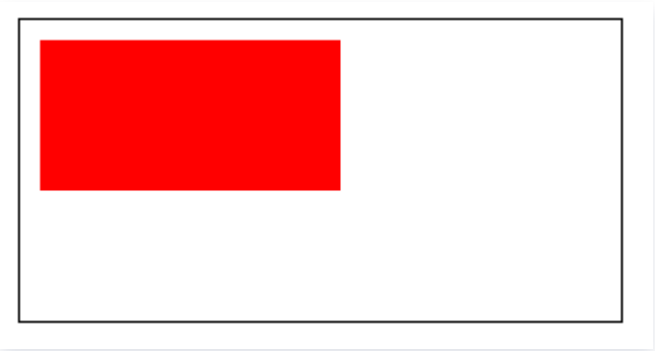


rect

该 方法 使用方式为 CanvasContext.rect(number x, number y, number width, number height)

- **功能说明：**创建一个矩形路径。需要用 fill 或者 stroke 方法将矩形真正的画到 canvas 中。
- **参数及说明：**number x，矩形路径左上角的横坐标；number y，矩形路径左上角的纵坐标；number width，矩形路径的宽度；number height，矩形路径的高度。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.rect(10, 10, 150, 75)
ctx.setFillStyle('red')
ctx.fill()
ctx.draw()
```



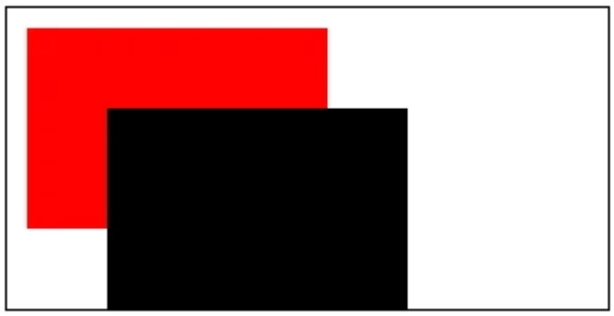
restore

该 方法 使用方式为 CanvasContext.restore()

- **功能说明：**恢复之前保存的绘图上下文。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
```

```
// save the default fill style
ctx.save()
ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 100)

// restore to the previous saved state
ctx.restore()
ctx.fillRect(50, 50, 150, 100)

ctx.draw()
```

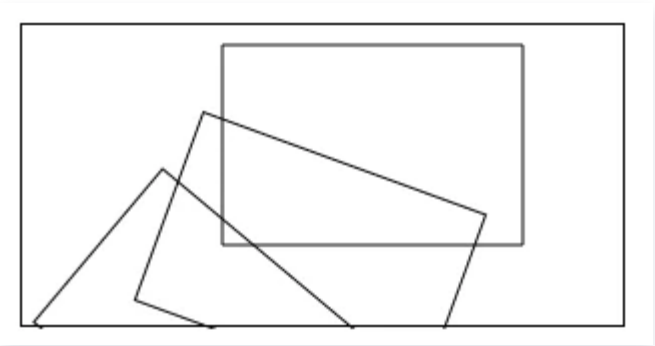


rotate

该 方法 使用方式为 CanvasContext.rotate(number rotate)

- **功能说明：**以原点为中心顺时针旋转当前坐标轴。多次调用旋转的角度会叠加。原点可以用 translate 方法修改。
- **参数及说明：**number rotate，旋转角度，以弧度计 degrees * Math.PI/180；degrees 范围为0-360。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.strokeRect(100, 10, 150, 100)
ctx.rotate(20 * Math.PI / 180)
ctx.strokeRect(100, 10, 150, 100)
ctx.rotate(20 * Math.PI / 180)
ctx.strokeRect(100, 10, 150, 100)

ctx.draw()
```

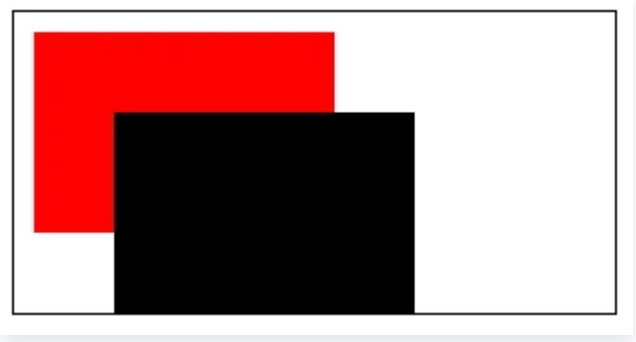


save

> 该 方法 使用方式为 CanvasContext.save()

- **功能说明：**保存绘图上下文。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

// save the default fill style
ctx.save()
ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 100)

// restore to the previous saved state
ctx.restore()
ctx.fillRect(50, 50, 150, 100)

ctx.draw()
```

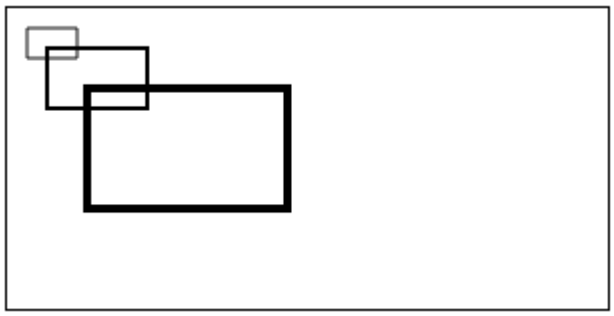


scale

> 该 方法 使用方式为 CanvasContext.scale(number scaleWidth, number scaleHeight)

- **功能说明：**在调用后，之后创建的路径其横纵坐标会被缩放。多次调用倍数会相乘。
- **参数及说明：**number scaleWidth，横坐标缩放的倍数 (1 = 100%，0.5 = 50%，2 = 200%)；number scaleHeight，纵坐标轴缩放的倍数 (1 = 100%，0.5 = 50%，2 = 200%)。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.strokeRect(10, 10, 25, 15)
ctx.scale(2, 2)
ctx.strokeRect(10, 10, 25, 15)
ctx.scale(2, 2)
ctx.strokeRect(10, 10, 25, 15)

ctx.draw()
```



setFillStyle

该 方法 使用方式为 CanvasContext.setFillStyle(string|CanvasGradient color)

- **功能说明：**设置填充色。
- **参数及说明：**string CanvasGradient color，填充的颜色，默认颜色为 black。
- **示例代码：**

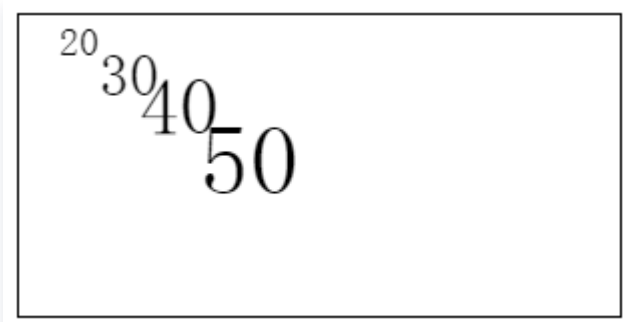
```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 75)
ctx.draw()
```

setFontSize

该 方法 使用方式为 CanvasContext.setFontSize(number fontSize)

- **功能说明：**设置字体的字号。
- **参数及说明：**number fontSize，字体的字号。
- **示例代码**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.setFontSize(20)
ctx.fillText('20', 20, 20)
ctx.setFontSize(30)
ctx.fillText('30', 40, 40)
ctx.setFontSize(40)
ctx.fillText('40', 60, 60)
ctx.setFontSize(50)
ctx.fillText('50', 90, 90)

ctx.draw()
```



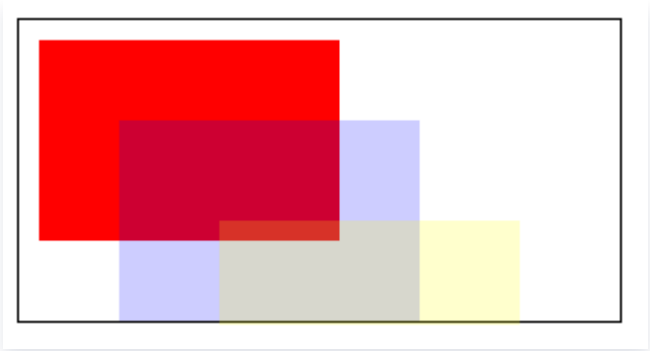
setGlobalAlpha

> 该 方法 使用方式为 CanvasContext.setGlobalAlpha(number alpha)

- **功能说明：**设置全局画笔透明度。
- **参数及说明：**number alpha，透明度。范围0-1，0表示完全透明，1表示完全不透明。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.setFillStyle('red')
ctx.fillRect(10, 10, 150, 100)
ctx.setGlobalAlpha(0.2)
ctx.setFillStyle('blue')
ctx.fillRect(50, 50, 150, 100)
ctx.setFillStyle('yellow')
```

```
ctx.fillRect(100, 100, 150, 100)

ctx.draw()
```



setLineCap

> 该 方法 使用方式为 CanvasContext.setLineCap(string lineCap)

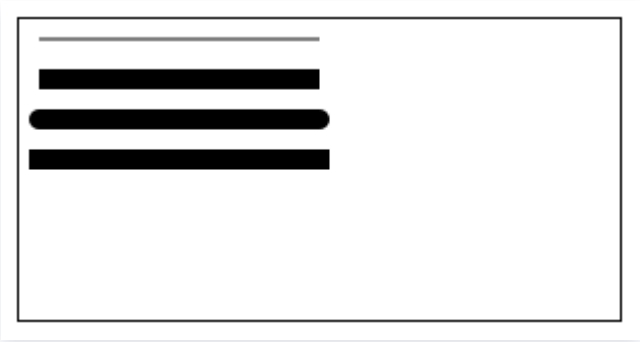
- **功能说明：**设置线条的端点样式。
- **参数及说明：**string lineGap，线条的结束交点样式。

值	说明
butt	向线条的每个末端添加平直的边缘
round	向线条的每个末端添加圆形线帽
square	向线条的每个末端添加正方形线帽

- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.beginPath()
ctx.moveTo(10, 10)
ctx.lineTo(150, 10)
ctx.stroke()

ctx.beginPath()
ctx.setLineCap('butt')
ctx.setLineWidth(10)
ctx.moveTo(10, 30)
ctx.lineTo(150, 30)
ctx.stroke()
```

```
ctx.beginPath()
ctx.setLineCap('round')
ctx.setLineWidth(10)
ctx.moveTo(10, 50)
ctx.lineTo(150, 50)
ctx.stroke()

ctx.beginPath()
ctx.setLineCap('square')
ctx.setLineWidth(10)
ctx.moveTo(10, 70)
ctx.lineTo(150, 70)
ctx.stroke()

ctx.draw()
```



setLineDash

> 该 方法 使用方式为 CanvasContext.setLineDash(Array.<number> pattern, number offset)

- **功能说明：**设置虚线样式。
- **参数及说明：**Array.<number> pattern，一组描述交替绘制线段和间距（坐标空间单位）长度的数字；number offset，虚线偏移量。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.setLineDash([10, 20], 5);

ctx.beginPath();
ctx.moveTo(0,100);
ctx.lineTo(400, 100);
ctx.stroke();

ctx.draw()
```

setLineJoin

> 该 方法 使用方式为 CanvasContext.setLineJoin(string lineJoin)

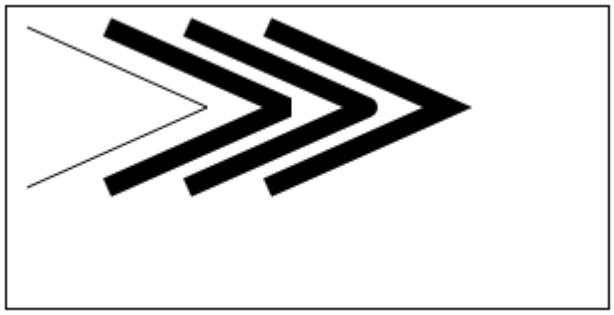
- **功能说明：**设置线条的交点样式。
- **参数及说明：**string lineJoin，线条的结束交点样式。

值	说明
bevel	斜角
round	圆角
miter	尖角

- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.beginPath()
ctx.moveTo(10, 10)
ctx.lineTo(100, 50)
ctx.lineTo(10, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineJoin('bevel')
ctx.setLineWidth(10)
ctx.moveTo(50, 10)
ctx.lineTo(140, 50)
ctx.lineTo(50, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineJoin('round')
```

```
ctx.setLineWidth(10)
ctx.moveTo(90, 10)
ctx.lineTo(180, 50)
ctx.lineTo(90, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineJoin('miter')
ctx.setLineWidth(10)
ctx.moveTo(130, 10)
ctx.lineTo(220, 50)
ctx.lineTo(130, 90)
ctx.stroke()

ctx.draw()
```



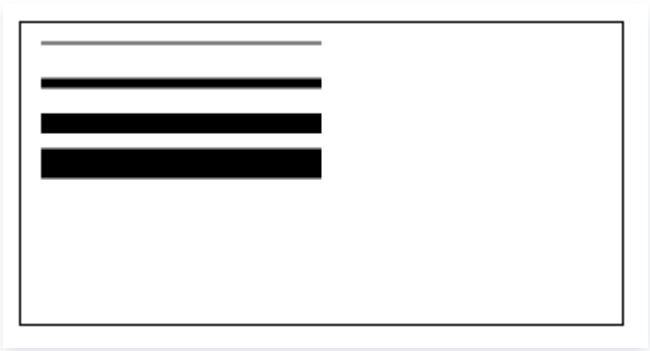
setLineWidth

> 该 方法 使用方式为 CanvasContext.setLineWidth(number lineWidth)

- **功能说明：**设置线条的宽度。
- **参数及说明：**number lineWidth，线条的宽度，单位：px。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.beginPath()
ctx.moveTo(10, 10)
ctx.lineTo(150, 10)
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(5)
ctx.moveTo(10, 30)
ctx.lineTo(150, 30)
```

```
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(10)
ctx.moveTo(10, 50)
ctx.lineTo(150, 50)
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(15)
ctx.moveTo(10, 70)
ctx.lineTo(150, 70)
ctx.stroke()

ctx.draw()
```



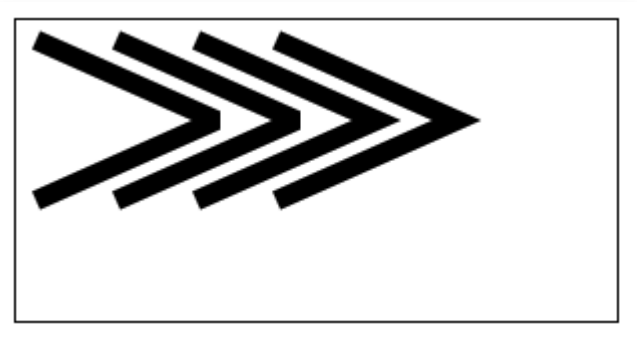
setMiterLimit

> 该 方法 使用方式为 CanvasContext.setMiterLimit(number miterLimit)

- **功能说明：**设置最大斜接长度。斜接长度指的是在两条线交汇处内角和外角之间的距离。当 CanvasContext.setLineJoin() 为 miter 时才有效。超过最大倾斜长度的，连接处将以 lineJoin 为 bevel 来显示。
- **参数及说明：**number miterLimit，最大斜接长度。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.beginPath()
ctx.setLineWidth(10)
ctx.setLineJoin('miter')
ctx.setMiterLimit(1)
ctx.moveTo(10, 10)
ctx.lineTo(100, 50)
ctx.lineTo(10, 90)
```

```
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(10)
ctx.setLineJoin('miter')
ctx.setMiterLimit(2)
ctx.moveTo(50, 10)
ctx.lineTo(140, 50)
ctx.lineTo(50, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(10)
ctx.setLineJoin('miter')
ctx.setMiterLimit(3)
ctx.moveTo(90, 10)
ctx.lineTo(180, 50)
ctx.lineTo(90, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(10)
ctx.setLineJoin('miter')
ctx.setMiterLimit(4)
ctx.moveTo(130, 10)
ctx.lineTo(220, 50)
ctx.lineTo(130, 90)
ctx.stroke()

ctx.draw()
```

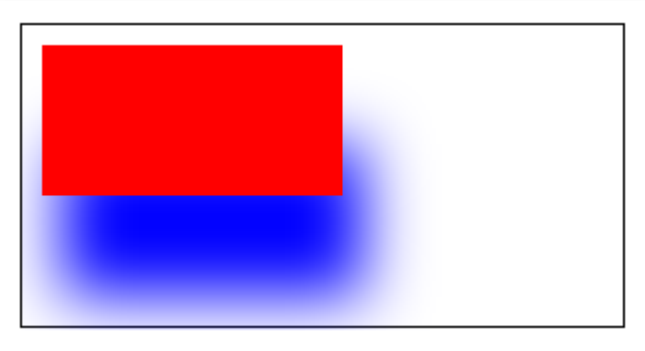


setShadow

该 方法 使用方式为 CanvasContext.setShadow(number offsetX, number offsetY, number blur, string color)

- **功能说明：**设定阴影样式。
- **参数及说明：**number offsetX，阴影相对于形状在水平方向的偏移，默认值为0；number offsetY，阴影相对于形状在竖直方向的偏移，默认值为 0；number blur，阴影的模糊级别，数值越大越模糊。范围 0- 100，默认值为 0；string color，阴影的颜色。默认值为 black。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setFillStyle('red')
ctx.setShadow(10, 50, 50, 'blue')
ctx.fillRect(10, 10, 150, 75)
ctx.draw()
```

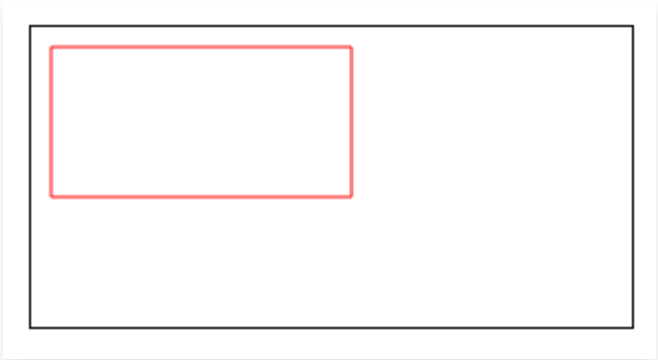


setStrokeStyle

> 该 方法 使用方式为 CanvasContext.setStrokeStyle(string|CanvasGradient color)

- **功能说明：**设置描边颜色。
- **参数及说明：**string CanvasGradient color，填充的颜色，默认颜色为 black。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setStrokeStyle('red')
ctx.strokeRect(10, 10, 150, 75)
ctx.draw()
```



setTextAlign

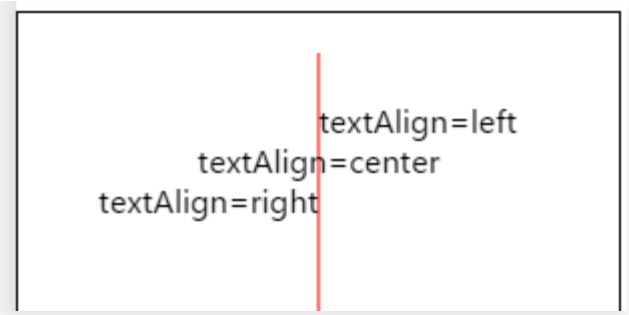
该 方法 使用方式为 CanvasContext.setTextAlign(string align)

- **功能说明：**设置文字的对齐。
- **参数及说明：**string align，文字的对齐方式。

值	说明
left	左对齐
center	居中对齐
right	右对齐

- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.setStrokeStyle('red')
ctx.moveTo(150, 20)
ctx.lineTo(150, 170)
ctx.stroke()

ctx.setFontSize(15)
ctx.setTextAlign('left')
ctx.fillText('textAlign=left', 150, 60)

ctx.setTextAlign('center')
ctx.fillText('textAlign=center', 150, 80)

ctx.setTextAlign('right')
ctx.fillText('textAlign=right', 150, 100)

ctx.draw()
```



setTextBaseline

该 方法 使用方式为 CanvasContext.setTextBaseline(string textBaseline)

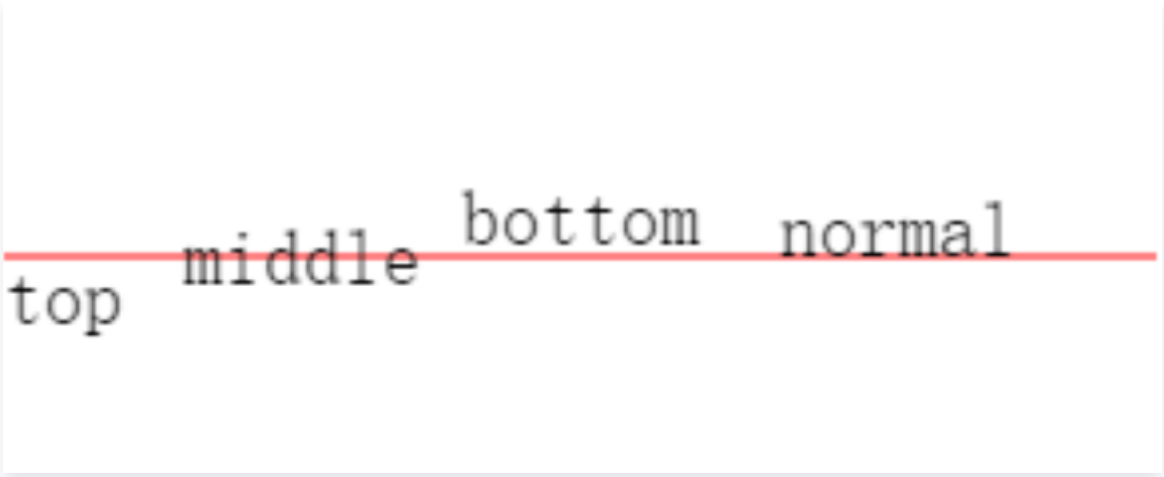
- **功能说明：**设置文字的竖直对齐。
- **参数及说明：**string textBaseline，文字的竖直对齐方式。

值	说明
top	顶部对齐
bottom	底部对齐
middle	居中对齐
normal	默认

- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.setStrokeStyle('red')
ctx.moveTo(5, 75)
ctx.lineTo(295, 75)
ctx.stroke()

ctx.setFontSize(20)

ctx.setTextBaseline('top')
ctx.fillText('top', 5, 75)

ctx.setTextBaseline('middle')
ctx.fillText('middle', 50, 75)

ctx.setTextBaseline('bottom')
```

```
ctx.fillText('bottom', 120, 75)

ctx.setTextBaseline('normal')
ctx.fillText('normal', 200, 75)

ctx.draw()
```



setTransform

> 该 方法 使用方式为 CanvasContext.setTransform(number scaleX, number skewX, number skewY, number scaleY, number translateX, number translateY)

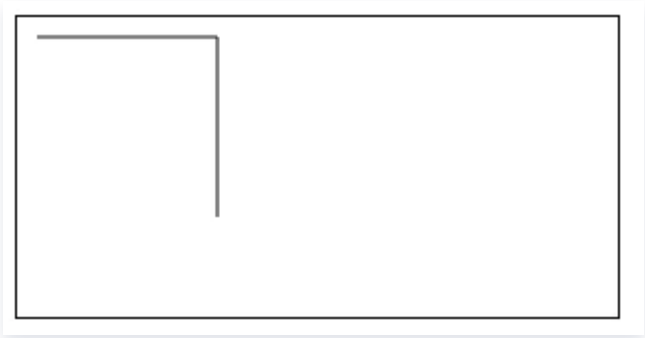
- **功能说明：**使用矩阵重新设置（覆盖）当前变换的方法。
- **参数及说明：**number scaleX，水平缩放；number scaleY，垂直缩放；number skewX，水平倾斜；number skewY，垂直倾斜；number translateX，水平移动；number translateY，垂直移动。

stroke

> 该 方法 使用方式为 CanvasContext.stroke()

- **功能说明：**画出当前路径的边框。默认颜色为黑色。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.moveTo(10, 10)
ctx.lineTo(100, 10)
ctx.lineTo(100, 100)
ctx.stroke()
ctx.draw()
```



stroke() 描绘的路径是从 beginPath() 开始计算，但是不会将 strokeRect() 包含进去。

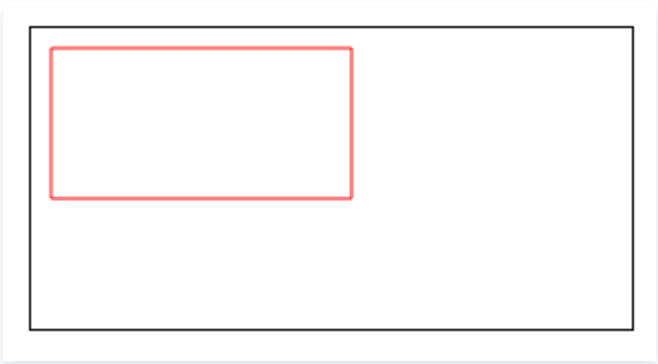
```
const ctx = wx.createCanvasContext('myCanvas')
// begin path
ctx.rect(10, 10, 100, 30)
ctx.setStrokeStyle('yellow')
ctx.stroke()

// begin another path
ctx.beginPath()
ctx.rect(10, 40, 100, 30)

// only stoke this rect, not in current path
ctx.setStrokeStyle('blue')
ctx.strokeRect(10, 70, 100, 30)

ctx.rect(10, 100, 100, 30)

// it will stroke current path
ctx.setStrokeStyle('red')
ctx.stroke()
ctx.draw()
```

strokeRect

> 该 方法 使用方式为 CanvasContext.strokeRect(number x, number y, number width, number height)

- **功能说明：**画一个矩形（非填充）。用 setStrokeStyle 设置矩形线条的颜色，如果没设置默认是黑色。
- **参数及说明：**number x，矩形路径左上角的横坐标；number y，矩形路径左上角的纵坐标；number width，矩形路径的宽度；number height，矩形路径的高度。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')
ctx.setStrokeStyle('red')
ctx.strokeRect(10, 10, 150, 75)
ctx.draw()
```



strokeText

> 该 方法 使用方式为 CanvasContext.strokeText(string text, number x, number y, number maxWidth)

- **功能说明：**给定的 (x, y) 位置绘制文本描边的方法。
- **参数及说明：**string text，要绘制的文本；number x，文本起始点的 x 轴坐标；number y，文本起始点的 y 轴坐标；number maxWidth，需要绘制的最大宽度，可选。

transform

> 该 方法 使用方式为 CanvasContext.transform(number scaleX, number skewX, number skewY, number scaleY, number translateX, number translateY)

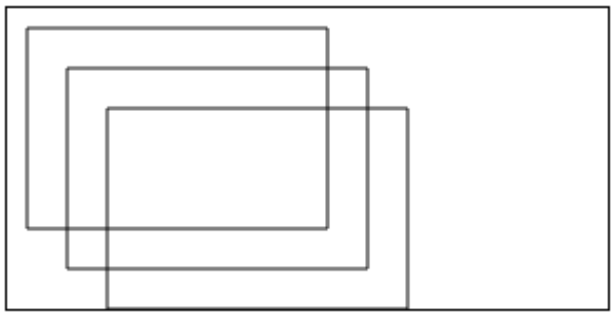
- **功能说明：**使用矩阵多次叠加当前变换的方法。
- **参数及说明：**number scaleX，水平缩放；number scaleY，垂直缩放；number skewX，水平倾斜；number skewY，垂直倾斜；number ranslate X，水平移动；number translate Y，垂直移动。

translate

该 方法 使用方式为 CanvasContext.translate(number x, number y)

- **功能说明：**对当前坐标系的原点 (0, 0) 进行变换。默认的坐标系原点为页面左上角。
- **参数及说明：**number x，水平坐标平移量；number y，竖直坐标平移量。
- **示例代码：**

```
const ctx = wx.createCanvasContext('myCanvas')

ctx.strokeRect(10, 10, 150, 100)
ctx.translate(20, 20)
ctx.strokeRect(10, 10, 150, 100)
ctx.translate(20, 20)
ctx.strokeRect(10, 10, 150, 100)

ctx.draw()
```



OffscreenCanvas

最近更新时间：2023-11-10 15:39:29

功能说明：离屏 canvas 实例，可通过 wx.createOffscreenCanvas 创建。

属性

属性	类型	说明
width	number	画布宽度
height	number	画布高度

方法集

createImage

> 该 方法 使用方式为 Image OffscreenCanvas.createImage()

- **功能说明：**创建一个图片对象。支持在 2D Canvas 和 WebGL Canvas 下使用，但不支持混用 2D 和 WebGL 的方法。
- **返回值：**Image，注意不允许混用 webgl 和 2d 画布创建的图片对象，使用时请注意尽量使用 canvas 自身的 createImage 创建图片对象。

getContext

> 该 方法 使用方式为 RenderingContext OffscreenCanvas.getContext(string contextType)

- **功能说明：**该方法返回 OffscreenCanvas 的绘图上下文。
- **参数及说明：**string contextType，绘图上下文类型，需要与 createOffscreenCanvas 时传入的 type 一致。

值	说明
webgl	webgl 类型上下文
2d	2d 类型

Path2D

最近更新时间：2023-11-10 15:39:29

- **功能说明：**Canvas 2D API 的接口 Path2D 用来声明路径，此路径稍后会被 CanvasRenderingContext2D 对象使用。CanvasRenderingContext2D 接口的 路径方法也存在于 Path2D 这个接口中，允许您在 Canvas 中根据需要创建可以保留并重用的路径。

属性

属性	类型	说明
width	number	使用像素描述 ImageData 的实际宽度
height	number	使用像素描述 ImageData 的实际高度
data	Uint8ClampedArray	一维数组，包含以 RGBA 顺序的数据，数据使用 0 至 255（包含）的整数表示

方法集

addPath

> 该 方法 使用方式为 Path2D.addPath(Path2D path)

- **功能说明：**添加路径到当前路径。
- **参数及说明：**Path2D path，添加的 Path2D 路径。

arc

> 该 方法 使用方式为 Path2D.arc(number x, number y, number radius, number startAngle, number endAngle, boolean counterclockwise)

- **功能说明：**添加一段圆弧路径。
- **参数及说明：**
  - number x：圆心横坐标。
  - number y：圆心纵坐标。
  - number radius：圆形半径，必须为正数。
  - number startAngle：圆弧开始角度。
  - number endAngle：圆弧结束角度。
  - boolean counterclockwise：是否逆时针绘制。如果传 true, 则会从 endAngle 开始绘制到 startAngle。

arcTo

> 该 方法 使用方式为 Path2D.arcTo(number x1, number y1, number x2, number y2, number radius)

- **功能说明：**通过给定控制点添加一段圆弧路径。
- **参数及说明：**
  - number x1：第一个控制点横坐标。
  - number y1：第一个控制点纵坐标。
  - number x2：第二个控制点横坐标。
  - number y2：第二个控制点纵坐标。
  - number radius：圆形半径，必须为非负数。

bezierCurveTo

> 该 方法 使用方式为 Path2D.bezierCurveTo(number cp1x, number cp1y, number cp2x, number cp2y, number x, number y)

- **功能说明：**添加三次贝塞尔曲线路径。
- **参数及说明：**
  - number cp1x：第一个控制点横坐标。
  - number cp1y：第一个控制点纵坐标。
  - number cp2x：第二个控制点横坐标。
  - number cp2y：第二个控制点纵坐标。
  - number x：结束点横坐标。
  - number y：结束点纵坐标。

closePath

> 该 方法 使用方式为 Path2D.closePath()

- **功能说明：**闭合路径到起点。

ellipse

> 该 方法 使用方式为 Path2D.ellipse(number x, number y, number radiusX, number radiusY, number rotation, number startAngle, number endAngle, boolean counterclockwise)

- **功能说明：**添加椭圆弧路径。
- **参数及说明：**

- number x：椭圆圆心横坐标。
- number y：椭圆圆心纵坐标。
- number radiusX：椭圆长轴半径，必须为非负数。
- number radiusY：椭圆短轴半径，必须为非负数。
- number rotation：椭圆旋转角度。
- number startAngle：圆弧开始角度。
- number endAngle：圆弧结束角度。
- boolean counterclockwise：是否逆时针绘制。如果传 true, 则会从 endAngle 开始绘制到 startAngle。

lineTo

> 该 方法 使用方式为 Path2D.lineTo(number x, number y)

- **功能说明：**添加直线路径。
- **参数及说明：**
  - number x：结束点横坐标。
  - number y：结束点纵坐标。

moveTo

> 该 方法 使用方式为 Path2D.moveTo(number x, number y)

- **功能说明：**移动路径开始点。
- **参数及说明：**
  - number x：横坐标。
  - number y：纵坐标。

quadraticCurveTo

> 该 方法 使用方式为 Path2D.quadraticCurveTo(number cpx, number cpy, number x, number y)

- **功能说明：**添加二次贝塞尔曲线路径。
- **参数及说明：**
  - number cpx：控制点横坐标。
  - number cpy：控制点纵坐标。
  - number x：结束点横坐标。
  - number y：结束点纵坐标。

rect

> 该 方法 使用方式为 Path2D.rect(number x, number y, number width, number height)

- **功能说明：**添加方形路径。
- **参数及说明：**
  - number x：开始点横坐标。
  - number y：开始点纵坐标。
  - number width：方形宽度，正数向右，负数向左。
  - number height：方形高度，正数向下，负数向上。

Canvas

最近更新时间：2023-11-10 15:39:29

属性

- number width：画布宽度。
- number height：画布高度。

方法集

cancelAnimationFrame

> 该 方法 使用方式为 Canvas.cancelAnimationFrame(number requestID)

- **功能说明：**取消由 requestAnimationFrame 添加到计划中的动画帧请求。支持在 2D Canvas 和 WebGL Canvas 下使用, 但不支持混用 2D 和 WebGL 的方法。
- **参数及说明：**number requestID。

createImageData

> 该 方法 使用方式为 Image Canvas.createImage()

- **功能说明：**创建一个图片对象。 支持在 2D Canvas 和 WebGL Canvas 下使用, 但不支持混用 2D 和 WebGL 的方法。
- **参数及说明：**Image 。

createImage

> 该 方法 使用方式为 ImageData Canvas.createImageData()

- **功能说明：**创建一个 ImageData 对象。仅支持在 2D Canvas 中使用。
- **参数及说明：**ImageData 。

createPathe2D

> 该 方法 使用方式为 Path2D Canvas.createPath2D(Path2D path)

- **功能说明：**创建 Path2D 对象。
- **参数及说明：**Path2D path。
- **返回值：**Path2D 。

getContext

该 方法 使用方式为 RenderingContext Canvas.getContext(string contextType)

- **功能说明：**该方法返回 Canvas 的绘图上下文。
- **参数及说明：**string contextType，上下文类型。

值	说明
2d	2d 绘图上下文
webgl	webgl 绘图上下文
webgl2	webgl2 绘图上下文

- **返回值：**RenderingContext，支持获取 2D 和 WebGL 绘图上下文。

requestAnimationFrame

该 方法 使用方式为 number Canvas.requestAnimationFrame(function callback)

- **功能说明：**在下次进行重绘时执行。 支持在 2D Canvas 和 WebGL Canvas 下使用, 但不支持混用 2D 和 WebGL 的方法。
- **参数及说明：**function callback，执行的 callback。
- **返回值：**number，请求 ID。

toDataURL

该 方法 使用方式为 string Canvas.toDataURL(string type, number encoderOptions)

- **功能说明：**返回一个包含图片展示的 data URI。可以使用 type 参数其类型，默认为 PNG 格式。
- **参数及说明：**
  - string type：图片格式，默认为 image/png。
  - number encoderOptions：在指定图片格式为 image/jpeg 或 image/webp 的情况下，可以从0到1的区间内选择图片的质量。如果超出取值范围，将会使用默认值0.92。其他参数会被忽略。
- **返回值：**string。

Color

最近更新时间：2023-11-10 15:39:25

Color Name	HEX
AliceBlue	#F0F8FF
AntiqueWhite	#FAEBD7
Aqua	#00FFFF
Aquamarine	#7FFFD4
Azure	#F0FFFF
Beige	#F5F5
Bisque	#FFE4C4
Black	#000000
BlanchedAlmond	#FFEBCD
Blue	#0000FF
BlueViolet	#8A2BE2
Brown	#A52A2A
BurlyWood	#DEB887
CadetBlue	#5F9EA0
Chartreuse	#7FFF00
Chocolate	#D2691E
Coral	#FF7F50
CornflowerBlue	#6495ED
Cornsilk	#FFF8DC
Crimson	#DC143C
Cyan	#00FFFF
DarkBlue	#00008B

DarkCyan	#008B8B
DarkGoldenRod	#B8860B
DarkGray	#A9A9A9
DarkGrey	#A9A9A9
DarkGreen	#006400
DarkKhaki	#BDB76B
DarkMagenta	#8B008B
DarkOliveGreen	#556B2F
DarkOrange	#FF8C00
DarkOrchid	#9932CC
DarkRed	#8B0000
DarkSalmon	#E9967A
DarkSeaGreen	#8FBC8F
DarkSlateBlue	#483D8B
DarkSlateGray	#2F4F4F
DarkSlateGrey	#2F4F4F
DarkTurquoise	#00CED1
DarkViolet	#9400D3
DeepPink	#FF1493
DeepSkyBlue	#00BFFF
DimGray	#696969
DimGrey	#696969
DodgerBlue	#1E90FF
FireBrick	#B22222
FloralWhite	#FFFAF0

ForestGreen	#228B22
Fuchsia	#FF00FF
Gainsboro	#DCDCDC
GhostWhite	#F8F8FF
Gold	#FFD700
GoldenRod	#DAA520
Gray	#808080
Grey	#808080
Green	#008000
GreenYellow	#ADFF2F
HoneyDew	#F0FFF0
HotPink	#FF69B4
IndianRed	#CD5C5C
Indigo	#4B0082
Ivory	#FFFFF0
Khaki	#F0E68C
Lavender	#E6E6FA
LavenderBlush	#FFF0F5
LawnGreen	#7CFC00
LemonChiffon	#FFFACD
LightBlue	#ADD8E6
LightCoral	#F08080
LightCyan	#E0FFFF
LightGoldenRodYellow	#FAFAD2
LightGray	#D3D3D3

LightGrey	#D3D3D3
LightGreen	#90EE90
LightPink	#FFB6C1
LightSalmon	#FFA07A
LightSeaGreen	#20B2AA
LightSkyBlue	#87CEFA
LightSlateGray	#778899
LightSlateGrey	#778899
LightSteelBlue	#B0C4DE
LightYellow	#FFFFE0
Lime	#00FF00
LimeGreen	#32CD32
Linen	#FAF0E6
Magenta	#FF00FF
Maroon	#800000
MediumAquaMarine	#66CDAA
MediumBlue	#0000CD
MediumOrchid	#BA55D3
MediumPurple	#9370DB
MediumSeaGreen	#3CB371
MediumSlateBlue	#7B68EE
MediumSpringGreen	#00FA9A
MediumTurquoise	#48D1CC
MediumVioletRed	#C71585
MidnightBlue	#191970

MintCream	#F5FFFA
MistyRose	#FFE4E1
Moccasin	#FFE4B5
NavajoWhite	#FFDEAD
Navy	#000080
OldLace	#FDF5E6
Olive	#808000
OliveDrab	#6B8E23
Orange	#FFA500
OrangeRed	#FF4500
Orchid	#DA70D6
PaleGoldenRod	#EEE8AA
PaleGreen	#98FB98
PaleTurquoise	#AFEEEE
PaleVioletRed	#DB7093
PapayaWhip	#FFEFD5
PeachPuff	#FFDAB9
Peru	#CD853F
Pink	#FFC0CB
Plum	#DDA0DD
PowderBlue	#B0E0E6
Purple	#800080
RebeccaPurple	#663399
Red	#FF0000
RosyBrown	#BC8F8F

RoyalBlue	#4169E1
SaddleBrown	#8B4513
Salmon	#FA8072
SandyBrown	#F4A460
SeaGreen	#2E8B57
SeaShell	#FFF5EE
Sienna	#A0522D
Silver	#C0C0C0
SkyBlue	#87CEEB
SlateBlue	#6A5ACD
SlateGray	#708090
SlateGrey	#708090
Snow	#FFFAFA
SpringGreen	#00FF7F
SteelBlue	#4682B4
Tan	#D2B48C
Teal	#008080
Thistle	#D8BFD8
Tomato	#FF6347
Turquoise	#40E0D0
Violet	#EE82EE
Wheat	#F5DEB3
White	#FFFFFF
WhiteSmoke	#F5F5F5
Yellow	#FFFF00

YellowGreen	#9ACD32

媒体

地图

最近更新时间：2023-11-10 15:39:25

createMapContext

> 该 API 使用方法为 wx.createMapContext(string mapId, Object this)

- **功能说明：**创建 map 上下文 MapContext 对象。
- **参数及说明：**string mapId，map 组件的 id；Object this，在自定义组件下，当前组件实例的 this，以操作组件内 map 组件
- **返回值：**MapContext 。

MapContext

> ⚠ **注意：**
>
> - 若无特殊说明，MapContext 的方法对手机厂商支持度默认为
> - 系统地图（仅 iOS 支持）：是；
> - 谷歌地图（Android、IDE 支持）：是；
> - 华为地图（仅 Android 支持）：是；
> - 腾讯地图：是；
> - 百度地图：是；
> - 高德地图：是。

- MapContext 实例，可通过 wx.createMapContext 获取。
- MapContext 通过 id 跟一个 <map> 组件绑定，操作对应的 <map> 组件。
- **示例代码：**

```
<!-- map.wxml -->
<map id="myMap" show-location />

<button type="primary" bindtap="getCenterLocation">获取位置</button>
<button type="primary" bindtap="moveToLocation">移动位置</button>
<button type="primary" bindtap="translateMarker">移动标注</button>
<button type="primary" bindtap="includePoints">缩放视野展示所有经纬度</button>
```

```
// map.js
Page({
  onReady: function (e) {
    // 使用 wx.createMapContext 获取 map 上下文
    this.mapCtx = wx.createMapContext('myMap')
  },
  getCenterLocation: function () {
    this.mapCtx.getCenterLocation({
      success: function(res){
        console.log(res.longitude)
        console.log(res.latitude)
      }
    })
  },
  moveToLocation: function () {
    this.mapCtx.moveToLocation()
  },
  translateMarker: function() {
    this.mapCtx.translateMarker({
      markerId: 0,
      autoRotate: true,
      duration: 1000,
      destination: {
        latitude:23.10229,
        longitude:113.3345211,
      },
      animationEnd() {
        console.log('animation end')
      }
    })
  },
  includePoints: function() {
    this.mapCtx.includePoints({
      padding: [10],
      points: [{
        latitude:23.10229,
        longitude:113.3345211,
      }, {
        latitude:23.00229,
        longitude:113.3345211,
      }]
    })
  }
})
```

.addArc

> 该 方法 使用方式为 MapContext.addArc(Object object)

- **功能说明：**添加弧线，途经点与夹角必须设置一个。途经点必须在起终点有效坐标范围内，否则不能生成正确的弧线，同时设置夹角角度时，以夹角角度为准。夹角定义为起点到终点，与起点外切线逆时针旋转的角度。工具侧暂未支持。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
id	number	–	是	圆弧 id
start	Object	–	是	起始点
end	Object	–	是	终点
pass	Object	–	否	途经点
angle	number	0	否	夹角角度
width	number	5	否	线宽
color	number	#000000	否	线的颜色
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

  - start 的结构属性

结构属性	类型	默认值	必填	说明

longitude	number	–	是	经度
latitude	number	–	是	纬度

- end 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

- pass 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

.addCustomLayer

> 该 方法 使用方式为 MapContext.addCustomLayer(Object object)

- **功能说明：**添加个性化图层。图层创建 参考文档 。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
layerId	string	–	是	个性化图层id
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.addGroundOverlay

> 该 方法 使用方式为 MapContext.addGroundOverlay(Object object)

- **功能说明：**创建自定义图片图层，图片会随着地图缩放而缩放。

- **参数及说明：** Object object

属性	类型	默认值	必填	说明
id	String	–	是	图片图层 id
src	String	–	是	图片路径，支持网络图片、临时路径、代码包路径
bounds	Object	–	是	图片覆盖的经纬度范围
visible	Boolean	true	否	是否可见
zIndex	Number	1	否	图层绘制顺序
opacity	Number	1	否	图层透明度
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

  - bounds 的结构属性

结构属性	类型	默认值	必填	说明
southwest	Object	–	是	西南角经纬度
northeast	Object	–	是	东北角经纬度

    - southwest 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

    - northeast 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	-	是	经度
latitude	number	-	是	纬度

.addMarkers

> 该 方法 使用方式为 MapContext.addMarkers(Object object)

- **功能说明：**添加 marker。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
markers	array	-	是	同传入 map 组件的 marker 属性
clear	boolean	false	否	是否先清空地图上所有 marker
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.eraseLines

> 该 方法 使用方式为 MapContext.eraseLines(Object object)

- **功能说明：**擦除或置灰已添加到地图中的线段。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
lines	Array.<Object>	-	是	要擦除的线段数组。详见 polyline 属性 。
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数

complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- lines 的结构属性

结构属性	类型	默认值	必填	说明
id	number	–	是	线段的 id
index	number	–	是	指定线段的某一段，线段起点 index 为0
point	Object	–	是	指定线段某一段中的点
clear	boolean	true	否	为 true 是擦除，false 是置灰

- point 的结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

.fromScreenLocation

> 该 方法 使用方式为 MapContext.fromScreenLocation(Object object)

- **功能说明：**获取屏幕上的点对应的经纬度，坐标原点为地图左上角。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
x	number	–	是	x 坐标值
y	number	–	是	y 坐标值
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	functi	–	否	接口调用结束的回调函数（调用成功、失败都会执

	on			行）

- **object.success 回调函数参数：** Object res。

属性	类型	说明
latitude	number	纬度
longitude	number	经度

.getCenterLocation

> 该 方法 使用方式为 MapContext.getCenterLocation(Object object)

- **功能说明：** 获取当前地图中心的经纬度。返回的是 gcj02 坐标系，可以用于 wx.openLocation() 。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
iconPath	string	–	否	图标路径，支持网络路径、本地路径、代码包路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
longitude	number	经度
latitude	number	纬度

.getRegion

> 该 方法 使用方式为 MapContext.getRegion(Object object)

- **功能说明：** 获取当前地图的视野范围。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
southwest	object	西南角经纬度
northeast	object	东北角经纬度

  - **southwest 结构属性**

结构属性	类型	说明
longitude	number	经度
latitude	number	纬度

  - **northeast 结构属性**

结构属性	类型	说明
longitude	number	经度
latitude	number	纬度

.getRotate

> 该 方法 使用方式为 MapContext.getRotate(Object object)

- **功能说明：** 获取当前地图的旋转角。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
rotate	number	旋转角

.getScale

> 该 方法 使用方式为 MapContext.getScale(Object object)

- **功能说明：** 获取当前地图的缩放级别。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
scale	number	缩放值

.getSkew

> 该 方法 使用方式为 MapContext.getSkew(Object object)

- **功能说明：** 获取当前地图的倾斜角。

- **参数及说明：** Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
skew	number	倾斜角

.includePoints

> 该 方法 使用方式为 MapContext.includePoints(Object object)

- **功能说明：** 缩放视野展示所有经纬度。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
points	array.	–	是	要显示在可视区域内的坐标点列表
padding	array.	–	否	坐标点形成的矩形边缘到地图边缘的距离，单位：像素。格式为[上,右,下,左]，Android 上只能识别数组第一项，上下左右的 padding 一致。开发者工具暂不支持 padding 参数。
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

points 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	-	是	经度
latitude	number	-	是	纬度

.initMarkerCluster

> 该 方法 使用方式为 MapContext.initMarkerCluster(Object object)

- **功能说明：**初始化点聚合的配置，未调用时采用默认配置。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
enableDefaultStyle	boolean	true	否	启用默认的聚合样式
zoomOnClick	boolean	true	否	点击已经聚合的标记点时是否实现聚合分离
gridSize	number	60	否	聚合算法的可聚合距离，即距离小于该值的点会聚合至一起，以像素为单位
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.moveAlong

> 该 方法 使用方式为 MapContext.moveAlong(Object object)

- **功能说明：**沿指定路径移动 marker ，用于轨迹回放等场景。动画完成时触发回调事件，若动画进行中，对同一 marker 再次调用 moveAlong 方法，前一次的动画将被打断。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
markerId	number	-	是	指定 marker
path	array	-	是	移动路径的坐标串，坐标点格式 {longitude, latitude}
autoRotate	boolean	true	否	根据路径方向自动改变 marker 的旋转角度
duration	number	-	是	平滑移动的时间
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.moveToLocation

> 该 方法 使用方式为 MapContext.moveToLocation(Object object)

- **功能说明：**将地图中心移至当前定位点，此时需设置地图组件 show-location 为 true。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
longitude	number	-	否	经度
latitude	number	-	否	纬度
success	function	-	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.on

该 方法 使用方式为 MapContext.on(string event, function callback)

- **功能说明：**监听地图事件。
- **参数1：**visualLayerEvent，可视化图层 visualLayer 统一回调出口，1.5.16 起支持。

参数	类型	说明
layerId	String	图层 id
eventType	String	事件类型
eventInfo	String	事件信息

- **参数2：**markerClusterCreate，缩放或拖动导致新的聚合簇产生时触发，仅返回新创建的聚合簇信息。

参数	类型	说明
clusters	Array<ClusterInfo>	聚合簇数据

- **参数3：**markerClusterClick，聚合簇的点击事件。

参数	类型	说明
cluster	ClusterInfo	聚合簇

  - ClusterInfo 结构

参数	类型	说明
clusterId	Number	聚合簇的 id
center	LatLng	聚合簇的坐标
markerIds	Array<Number>	该聚合簇内的点标记数据数组

- **参数4：**string event，事件名；function callback，事件的回调函数。
  - event 合法值有 markerClusterCreate、markerClusterClick、visualLayerEvent。

- **示例代码**

```
MapContext.on('visualLayerEvent', (res) => {})
MapContext.on('markerClusterCreate', (res) => {})
MapContext.on('markerClusterClick', (res) => {})
```

.openMapApp

> 该 方法 使用方式为 MapContext.openMapApp(Object object)

- **功能说明：** 拉起地图 App 选择导航。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
longitude	number	–	是	目的地经度
latitude	number	–	是	目的地纬度
destination	string	–	是	目的地名称
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.removeArc

> 该 方法 使用方式为 MapContext.removeArc(Object object)

- **功能说明：** 删除弧线。工具侧暂未支持。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
id	numb	–	是	圆弧 id

	er			
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.removeCustomLayer

该 方法 使用方式为 MapContext.removeCustomLayer(Object object)

- **功能说明：**移除个性化图层。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
layerId	string	–	是	个性化图层id
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.removeGroundOverlay

该 方法 使用方式为 MapContext.removeGroundOverlay(Object object)

- **功能说明：**移除自定义图片图层。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
id	String	–	是	图片图层 id
success	function	–	否	接口调用成功的回调函数
fail	functi	–	否	接口调用失败的回调函数

	on			
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.removeMarkers

> 该 方法 使用方式为 MapContext.removeMarkers(Object object)

- **功能说明：**移除 marker。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
markerIds	array	–	是	marker 的 id 集合。
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.setBoundary

> 该 方法 使用方式为 MapContext.setBoundary(Object object)

- **功能说明：**限制地图的显示范围。此接口同时会限制地图的最小缩放整数级别。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
southwest	Object	–	是	西南角经纬度
northeast	Object	–	是	东北角经纬度
success	function	–	否	接口调用成功的回调函数
fail	functi	–	否	接口调用失败的回调函数

	on			
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- southwest 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

- northeast 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

.setCenterOffset

> 该 方法 使用方式为 MapContext.setCenterOffset(Object object)

- **功能说明：**设置地图中心点偏移，向后向下为增长，屏幕比例范围(0.25~0.75)，默认偏移为[0.5, 0.5]。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
offset	Array.<number>	–	是	偏移量，两位数组
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.setLocMarkerIcon

> 该 方法 使用方式为 MapContext.setLocMarkerIcon(Object object)

- **功能说明：**设置定位点图标，支持网络路径、本地路径、代码包路径。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
iconPath	string	–	否	图标路径，支持网络路径、本地路径、代码包路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.toScreenLocation

> 该 方法 使用方式为 MapContext.toScreenLocation(Object object)

- **功能说明：**获取经纬度对应的屏幕坐标，坐标原点为地图左上角。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
latitude	Number	–	是	纬度
longitude	Number	–	是	经度
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
x	number	x 坐标值

y	number	y 坐标值

.translateMarker

该 方法 使用方式为 MapContext.translateMarker(Object object)

- **功能说明：**平移 marker，iOS 以及 IDE 支持动画效果，Android 平台暂不支持。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
markerId	number	–	是	指定 marker
destination	object	–	是	指定 marker 移动到的目标点
animationEnd	function	–	否	动画结束回调函数
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

destination 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

.updateGroundOverlay

该 方法 使用方式为MapContext.updateGroundOverlay(Object object)

- **功能说明：**更新自定义图片图层。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明

id	String	–	是	图片图层 id
src	String	–	是	图片路径，支持网络图片、临时路径、代码包路径
bounds	Object	–	是	图片覆盖的经纬度范围
visible	Boolean	true	否	是否可见
zIndex	Number	1	否	图层绘制顺序
opacity	Number	1	否	图层透明度
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- bounds 的结构属性

结构属性	类型	默认值	必填	说明
southwest	Object	–	是	西南角经纬度
northeast	Object	–	是	东北角经纬度

  - southwest 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

  - northeast 结构属性

结构属性	类型	默认值	必填	说明
longitude	number	–	是	经度
latitude	number	–	是	纬度

图片

最近更新时间：2023-10-20 15:19:18

chooseImage

> 该 API 使用方法为 wx.chooseImage(Object object)

- **功能说明：**从本地相册选择图片或使用相机拍照。
- **参数及说明：**Object object。

属性	类型	合法值及说明	默认值	必填	说明
count	number	–	9	否	多可以选择的图片张数
sizeType	Array.	original：原图 compressed：压缩图	['original', 'compressed']	否	所选的图片的尺寸
sourceType	Array.	album：从相册选图 camera：使用相机	['album', 'camera']	否	选择图片的来源
success	function	–	–	否	接口调用成功的回调函数
fail	function	–	–	否	接口调用失败的回调函数
complete	function	–	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
tempFilePaths	Array.	图片的本地临时文件路径列表 (本地路径)
tempFiles	Array	图片的本地临时文件列表

	.	

- **res.tempFiles 的结构**

属性	类型	说明
path	string	本地临时文件路径
size	number	本地临时文件大小，单位 B

- **示例代码**

```
wx.chooseImage({
  count: 1,
  sizeType: ['original', 'compressed'],
  sourceType: ['album', 'camera'],
  success (res) {
    // tempFilePath可以作为 img 标签的 src 属性显示图片
    const tempFilePaths = res.tempFilePaths
  }
})
```

compressImage

> 该 API 使用方法为 wx.compressImage(Object object)

- **功能说明：**压缩图片接口，可选压缩质量。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
src	string	–	是	图片路径，图片的路径，支持本地路径、代码包路径
quality	number	80	否	压缩质量，范围0～100，数值越小，质量越低，压缩率越高（仅对 jpg 有效）
compressedWidth	number	–	否	压缩后图片的宽度，单位为px，若不填写则默认以 compressedHeight 为准等比缩放
compressedHeight	number	–	否	压缩后图片的高度，单位为px，若不填写则默认以 compressedWidth为准等比缩放

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res。

属性	类型	说明
tempFilePath	string	压缩后图片的临时文件路径 (本地路径)

- **示例代码**

```
wx.compressImage({
  src: '', // 图片路径
  quality: 80 // 压缩质量
})
```

getImageInfo

> 该 API 使用方法为 wx.getImageInfo(Object object)

- **功能说明：** 获取图片信息。网络图片需先配置 download 域名才能生效。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
src	string	–	是	图片的路径，可以是相对路径、临时文件路径、存储文件路径、网络图片路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res。

属性	类型	说明

width	number	图片原始宽度，单位 px。不考虑旋转
height	number	图片原始高度，单位 px。不考虑旋转
path	string	图片的本地路径
orientation	string	拍照时设备方向
type	string	图片格式

- **res.orientation 的合法值**

值	说明
up	默认方向（手机横持拍照），对应 Exif 中的 1，或无 orientation 信息
up-mirrored	同 up，但镜像翻转，对应 Exif 中的2
down	旋转180度，对应 Exif 中的3
down-mirrored	同 down，但镜像翻转，对应 Exif 中的4
left-mirrored	同 left，但镜像翻转，对应 Exif 中的5
right	顺时针旋转90度，对应 Exif 中的6
right-mirrored	同 right，但镜像翻转，对应 Exif 中的7
left	逆时针旋转90度，对应 Exif 中的8

- **示例代码：**

```
wx.getImageInfo({
  src: 'images/a.jpg',
  success(res) {
    console.log(res.width)
    console.log(res.height)
  },
})

wx.chooseImage({
  success(res) {
```

```
    wx.getImageInfo({
      src: res.tempFilePaths[0],
      success(res) {
        console.log(res.width)
        console.log(res.height)
      },
    })
  },
})
```

previewImage

该 API 使用方法为 wx.previewImage(Object object)

- **功能说明：**在新页面中全屏预览图片。预览的过程中用户可以进行保存图片、发送给朋友等操作。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
urls	Array.<string>	-	是	需要预览的图片链接列表。 支持云文件 ID
current	string	urls 的第一张	否	当前显示图片的链接
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.previewImage({
  current: '', // 当前显示图片的http链接
  urls: [], // 需要预览的图片http链接列表
})
```

previewMedia

该 API 使用方法为 wx.previewMedia(Object object)

- **功能说明：**预览图片和视频。

- **参数及说明：**Object object。

类型	默认值	必填	说明
Array.<Object>	-	是	需要预览的资源列表
number	0	否	当前显示的资源序号
boolean	true	否	是否显示长按菜单
string	no-referrer	否	origin：发送完整的 referrer no-referrer：不发送 格式固定为 `https://servicewechat.com/{appid}/{version}/page-frame.html`，其中 • {appid} 为小程序的 appid • {version} 为小程序的版本号，版本号为 0 表示为开发版、体验版以及审核版本，版本号为 devtools 表示为开发者工具，其余为正式版本
function	-	否	接口调用成功的回调函数
function	-	否	接口调用失败的回调函数
function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **支持长按识别的码**

类型	说明
小程序码	-
宿主客户端微信个人码	不支持小游戏
宿主客户端企业微信个人码	不支持小游戏
普通群码	仅包含宿主客户端内微信用户的群，不支持小游戏
互通群码	指宿主客户端内既有微信用户也有企业微信用户的群，不支持小游戏
公众号二维码	不支持小游戏

saveImageToPhotosAlbum

该 API 使用方法为 wx.saveImageToPhotosAlbum(Object object)

说明：
调用前需要“用户授权” scope.writePhotosAlbum。

- **功能说明：**保存图片到系统相册。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
filePath	string	-	是	图片文件路径，可以是临时文件路径或永久文件路径，不支持网络图片路径
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.saveImageToPhotosAlbum({
  success(res) {},
})
```

chooseMessageFile

该 API 使用方法为 wx.chooseMessageFile(Object object)

- **功能说明：**从客户端会话选择文件。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
count	number	-	是	最多可以选择的文件个数，可以 0～100

type	string	all	是	所选的文件的类型，合法值为 • all：从所有文件选择 • video：只能选择视频文件 • image：只能选择图片文件 • file：可以选择除了图片和视频之外的其它的文件
extension	Array.<string>	–	否	根据文件拓展名过滤，仅 type==file 时有效。每一项都不能是空字符串。默认不过滤
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
tempFile	Array.<string>	返回选择的文件的本地临时文件对象数组

  - **tempFile 结构属性**

结构属性	类型	说明
path	string	本地临时文件路径 (本地路径)
size	number	本地临时文件大小，单位 B
name	string	选择的文件名称
type	string	选择的文件类型，合法值 • video：选择了视频文件 • image：选择了图片文件 • file：选择了除图片和视频的文件
time	number	选择的文件的会话发送时间，Unix 时间戳，工具暂不支持此属性

实时音视频

最近更新时间：2023-11-10 15:39:25

createLivePusherContext

该 API 使用方法为 LivePusherContext wx.createLivePusherContext()

说明：
基础库 1.4.96 开始支持，低版本需做兼容处理。

- **功能说明：**创建 live-pusher 上下文 LivePusherContext 对象。
- **返回值：**LivePusherContext 。

LivePusherContext

LivePusherContext 实例，可通过 wx.createLivePusherContext 。
LivePusherContext 与页面内唯一的 live-pusher 组件绑定，操作对应的 live-pusher 组件。

.start

该 方法 使用方式为 LivePusherContext.start(Object object)

- **功能说明：**开始推流，同时开启摄像头预览。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用失败的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.stop

该 方法 使用方式为 LivePusherContext.stop(Object object)

- **功能说明：**停止推流，同时停止摄像头预览。

- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.pause

> 该 方法 使用方式为 LivePlayerContext.pause(Object object)

- **功能说明：** 暂停推流。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.resume

> 该 方法 使用方式为 LivePusherContext.resume(Object object)

- **功能说明：** 恢复推流。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complet	Functio	–	否	接口调用结束的回调函数（调用成功、失败都会执

e	n			行）

.switchCamera

该 方法 使用方式为 LivePusherContext.switchCamera(Object object)

- **功能说明：**切换前后摄像头。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.snapshot

该 方法 使用方式为 LivePusherContext.snapshot(Object object)

- **功能说明：**快照。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.toggleTorch

该 方法 使用方式为 LivePusherContext.toggleTorch(Object object)

- **功能说明：**切换手电筒。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.playBGM

> 该 方法 使用方式为 LivePusherContext.playBGM(Object object)

- **功能说明：**播放背景音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
url	String	–	是	加入背景混音的资源地址
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.stopBGM

> 该 方法 使用方式为 LivePusherContext.stopBGM(Object object)

- **功能说明：**停止背景音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数

complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.pauseBGM

该 方法 使用方式为 LivePusherContext.pauseBGM(Object object)

- **功能说明：**暂停背景音。
- **参数及说明：**Object objec。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.resumeBGM

该 方法 使用方式为 LivePusherContext.resumeBGM(Object object)

- **功能说明：**恢复背景音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.setBGMPosition

该 方法 使用方式为 LivePusherContext.setBGMPosition(Object object)

- **功能说明：**设置背景音进度。

- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
position	Number	–	是	背景音进度，单位：秒
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.setMICVolume

该 方法 使用方式为 LivePusherContext.setMICVolume(Object object)

- **功能说明：**设置麦克风音量。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
volume	String	–	是	音量大小，范围是 0–1
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.setBGMVolume

该 方法 使用方式为 LivePusherContext.setBGMVolume(Object object)

- **功能说明：**设置背景音量。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
volume	String	–	是	音量大小，范围是 0–1

success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.setAudioReverbType

该 方法 使用方式为 LivePusherContext.setAudioReverbType(Object object)

- **功能说明：**设置混音类型。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
audioReverbType	Number	–	是	混音类型，0 ~ 6 分别对应 "关闭混响", "KTV", "小房间", "大会堂", "低沉", "洪亮", "磁性"
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.startPreview

该 方法 使用方式为 LivePusherContext.startPreview(Object object)

- **功能说明：**开启摄像头预览。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complet	function	–	否	接口调用结束的回调函数（调用成功、失败都会执

e				行）

.stopPreview

该 方法 使用方式为 LivePusherContext.stopPreview(Object object)

- **功能说明：**关闭摄像头预览。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用失败的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.startAudioRecord

该 方法 使用方式为 LivePusherContext.startAudioRecord(Object object)

- **功能说明：**开始录音，当主动调用 LivePusherContext.stopAudioRecord 时，或者录音超过1分钟时自动结束录音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用失败的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.stopAudioRecord

该 方法 使用方式为 LivePusherContext.stopAudioRecord(Object object)

- **功能说明：**结束录音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用失败的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res。

属性	类型	说明
tempFilePath	String	录音文件的临时路径 (本地路径)

createLivePlayerContext

> 该 API 使用方法为 LivePlayerContext wx.createLivePlayerContext(string id, Object this)

> **说明：**
> 基础库 1.4.96 开始支持，低版本需做兼容处理。

- **功能说明：** 创建 live-player 上下文 LivePlayerContext 对象。
- **参数及说明：** string id，live-player 组件的 id；Object this，在自定义组件下，当前组件实例的 this，以操作组件内 live-player 组件。
- **返回值：** LivePlayerContext 。

LivePlayerContext

.exitfullscreen

> 该 方法 使用方式为 LivePlayerContext.exitFullScreen(Object object)

- **功能说明：** 退出全屏。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用失败的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.mute

> 该 方法 使用方式为 LivePlayerContext.mute(Object object)

- **功能说明：**静音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.pause

> 该 方法 使用方式为 LivePlayerContext.pause(Object object)

- **功能说明：**暂停。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.play

> 该 方法 使用方式为 LivePlayerContext.play(Object object)

- **功能说明：**播放。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数

complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.requestFullScreen

该 方法 使用方式为 LivePlayerContext.requestFullScreen(Object object)

- **功能说明：** 进入全屏。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
direction	Number	0	否	设置全屏时的方向
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

object.direction 的合法值。

值	说明
0	正常竖向
90	屏幕逆时针 90 度
-90	屏幕顺时针 90 度

.resume

该 方法 使用方式为 LivePlayerContext.resume(Object object)

- **功能说明：** 恢复。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	Function	–	否	接口调用成功的回调函数
fail	Function	–	否	接口调用失败的回调函数
complete	Function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.snapshot

> 该 方法 使用方式为 LivePlayerContext.snapshot(Object object)

- **功能说明：**截图。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用失败的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
tempImagePath	String	图片文件的临时路径
width	String	图片的宽度
height	String	图片的高度

.stop

> 该 方法 使用方式为 LivePlayerContext.stop(Object object)

- **功能说明：**停止。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用失败的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

音频

最近更新时间：2023-12-06 15:14:27

stopVoice

该 API 使用方法为 wx.stopVoice(Object object)

- **功能说明：**结束播放语音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.startRecord({
  success (res) {
    const tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath: tempFilePath,
    })

    setTimeout(() => { wx.stopVoice() }, 5000)
  }
})
```

playVoice

该 API 使用方法为 wx.playVoice(Object object)

- **功能说明：**开始播放语音。同时只允许一个语音文件正在播放，如果前一个语音文件还没播放完，将中断前一个语音播放。

- **参数及说明：**

属性	类型	默认值	必填	说明
filePath	string	–	是	需要播放的语音文件的文件路径 (本地路径)
duration	number	60	否	指定播放时长，到达指定的播放时长后会自动停止播放，单位：秒
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.startRecord({
  success (res) {
    const tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath: tempFilePath,
      complete () { }
    })
  }
})
```

pauseVoice

> 该 API 使用方法为 wx.pauseVoice(Object object)

- **功能说明：**暂停正在播放的语音。再次调用 wx.playVoice 播放同一个文件时，会从暂停处开始播放。如果想从头开始播放，需要先调用 wx.stopVoice 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.startRecord({
  success (res) {
    const tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath: tempFilePath
    })

    setTimeout(() => { wx.pauseVoice() }, 5000)
  }
})
```

setInnerAudioOption

> 该 API 使用方法为 wx.setInnerAudioOption(Object object)

- **功能说明：**设置 InnerAudioContext 的播放选项。设置之后对当前小程序全局生效。
- 参数：Object object。

属性	类型	默认值	必填	说明
mixWithOther	boolean	true	否	是否与其他音频混播，设置为 true 之后，不会终止其他应用的音乐
obeyMuteSwitch	boolean	true	否	（仅在 iOS 生效）是否遵循静音开关，设置为 false 之后，即使是在静音模式下，也能播放声音
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数

complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

getAvailableAudioSources

该 API 使用方法为 wx.getAvailableAudioSources(Object object)

- **功能说明：**获取当前支持的音频输入源。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
audioSources	Array.<string>	支持的音频输入源列表。返回值定义参考 AudioSource

  - **audioSources 合法值**

合法值	说明
auto	自动设置，默认使用手机麦克风，插上耳麦后自动切换使用耳机麦克风，所有平台适用
buildInMic	手机麦克风，仅限 iOS
headsetMic	耳机麦克风，仅限 iOS
mic	麦克风（没插耳麦时是手机麦克风，插耳麦时是耳机麦克风），仅限 Android
camcorder	同 mic，适用于录制音视频内容，仅限 Android
voice_comm	同 mic，适用于实时沟通，仅限 Android

unication	
voice_recognition	同 mic，适用于语音识别，仅限 Android

createAudioContext

该 API 使用方法为 AudioContext wx.createAudioContext(string id, Object this)

- **功能说明：**创建 audio 上下文 AudioContext 对象。
- **参数及说明：**string id，audio 组件的 id；Object this，在自定义组件下，当前组件实例的 this，以操作组件内 audio 组件。
- **返回值：**AudioContext

AudioContext

AudioContext 实例，可通过 wx.createAudioContext 获取。通过 id 跟一个 audio 组件绑定，操作对应的 audio 组件。

pause

该 方法 使用方式为 AudioContext.pause()

功能说明：暂停音频。

play

该 方法 使用方式为 AudioContext.play()

功能说明：播放音频。

seek

该 方法 使用方式为 AudioContext.seek(number position)

- **功能说明：**跳转到指定位置。
- **参数及说明：**number position，跳转位置，单位：s。

setSrc

该 方法 使用方式为 AudioContext.setSrc(string src)

- **功能说明：**设置音频地址。
- **参数及说明：**string src，音频地址。

createInnerAudioContext

> 该 API 使用方法为 InnerAudioContext wx.createInnerAudioContext(Object object)

- **功能说明：**创建内部 `audio` 上下文 InnerAudioContext 对象，使用方法为 wx.createInnerAudioContext()。
- **返回值：**InnerAudioContext 。

InnerAudioContext

属性

- string src：音频资源的地址，用于直接播放。支持云文件 ID。
- number startTime：开始播放的位置（单位：s），默认为0。
- boolean autoplay：是否自动开始播放，默认为 false。
- boolean loop：是否循环播放，默认为 false。
- number volume：音量，范围0~1。默认为1。
- number duration：当前音频的长度（单位：s）。只有在当前有合法的 src 时返回（只读）。
- number currentTime：当前音频的播放位置（单位 s）。只有在当前有合法的 src 时返回，时间保留小数点后 6 位（只读）。
- boolean paused：当前是否是暂停或停止状态（只读）。
- number buffered：音频缓冲的时间点，仅保证当前播放时间点到此时间点内容已缓冲（只读）。

支持格式

格式	iOS	Android
flac	x	√
m4a	√	√
ogg	x	√
ape	x	√
amr	x	√
wma	x	√
wav	√	√
mp3	√	√
mp4	x	√

aac	√	√
aiff	√	x
caf	√	x

示例代码

```
const innerAudioContext = wx.createInnerAudioContext()
innerAudioContext.autoplay = true
innerAudioContext.src =
'https://ws.stream.qqmusic.qq.com/M500001VfvsJ21xFqb.mp3?
guid=ffffffff82def4af4b12b3cd9337d5e7&uin=346897220&vkey=6292F51E1E384E06
1FF02C31F716658E5C81F5594D561F2E88B854E81CAAB7806D5E4F103E55D33C16F
3FAC506D1AB172DE8600B37E43FAD&fromtag=46'
innerAudioContext.onPlay(() => {
  console.log('开始播放')
})
innerAudioContext.onError((res) => {
  console.log(res.errMsg)
  console.log(res.errCode)
})
```

方法集

play

> 该 方法 使用方式为 InnerAudioContext.play()

功能说明： 播放。

pause

> 该 方法 使用方式为 InnerAudioContext.pause()

功能说明： 暂停，暂停后的音频再播放会从暂停处开始播放。

stop

> 该 方法 使用方式为 InnerAudioContext.stop()

功能说明： 停止，停止后的音频再播放会从头开始播放。

seek

该 方法 使用方式为 InnerAudioContext.seek(number position)

功能说明：跳转到指定位置。

destroy

该 方法 使用方式为 InnerAudioContext.destroy()

功能说明：销毁当前实例。

onCanplay

该 方法 使用方式为 InnerAudioContext.onCanplay(function listener)

- **功能说明：**监听音频进入可以播放状态的事件。但不保证后面可以流畅播放。
- **参数及说明：**function listener，音频进入可以播放状态的事件的监听函数。

offCanplay

该 方法 使用方式为 InnerAudioContext.offCanplay(function listener)

- **功能说明：**移除音频进入可以播放状态的事件的监听函数。
- **参数及说明：**function listener，onCanplay 传入的监听函数。不传此参数则移除所有监听函数。

onPlay

该 方法 使用方式为 InnerAudioContext.onPlay(function listener)

- **功能说明：**监听音频播放事件。
- **参数及说明：**function listener，音频播放事件的监听函数。

offPlay

该 方法 使用方式为 InnerAudioContext.offPlay(function listener)

- **功能说明：**移除音频播放事件的监听函数。
- **参数及说明：**function listener，onPlay 传入的监听函数。不传此参数则移除所有监听函数。

onPuase

该 方法 使用方式为 InnerAudioContext.onPause(function listener)

- **功能说明：**监听音频暂停事件。
- **参数及说明：**function listener，音频暂停事件的监听函数。

offPause

该 方法 使用方式为 InnerAudioContext.offPause(function listener)

- **功能说明：**移除音频暂停事件的监听函数。
- **参数及说明：**function listener，onPause 传入的监听函数。不传此参数则移除所有监听函数。

onStop

该 方法 使用方式为 InnerAudioContext.onStop(function listener)

- **功能说明：**监听音频停止事件。
- **参数及说明：**function listener，音频停止事件的监听函数。

offStop

该 方法 使用方式为 InnerAudioContext.offStop(function listener)

- **功能说明：**移除音频停止事件的监听函数。
- **参数及说明：**function listener，onStop 传入的监听函数。不传此参数则移除所有监听函数。

onEnded

该 方法 使用方式为 InnerAudioContext.onEnded(function listener)

- **功能说明：**监听音频自然播放至结束的事件。
- **参数及说明：**function listener，音频自然播放至结束的事件的监听函数。

offEnded

该 方法 使用方式为 InnerAudioContext.offEnded(function listener)

- **功能说明：**移除音频自然播放至结束的事件的监听函数。
- **参数及说明：**function listener，onEnded 传入的监听函数。不传此参数则移除所有监听函数。

onTimeUpdate

该 方法 使用方式为 InnerAudioContext.onTimeUpdate(function listener)

- **功能说明：**监听音频播放进度更新事件。
- **参数及说明：**function listener，音频播放进度更新事件的监听函数。

offTimeUpdate

该 方法 使用方式为 InnerAudioContext.offTimeUpdate(function listener)

- **功能说明：**移除音频播放进度更新事件的监听函数。

- **参数及说明：**function listener，onTimeUpdate 传入的监听函数。不传此参数则移除所有监听函数。

onError

> 该 方法 使用方式为 InnerAudioContext.onError(function listener)

- **功能说明：**监听音频播放错误事件。
- **参数及说明：**function listener，音频播放错误事件的监听函数。
- **Object res**

属性	类型	说明
errCode	number	

- **errCode 的合法值**

值	说明
10001	系统错误
10002	网络错误
10003	文件错误
10004	格式错误
-1	未知错误

offError

> 该 方法 使用方式为 InnerAudioContext.offError(function listener)

- **功能说明：**移除音频播放错误事件的监听函数。
- **参数及说明：**function listener，onError 传入的监听函数。不传此参数则移除所有监听函数。

onWaitting

> 该 方法 使用方式为 InnerAudioContext.onWaiting(function listener)

- **功能说明：**监听音频加载中事件。当音频因为数据不足，需要停下来加载时会触发。
- **参数及说明：**function listener，onWaiting 传入的监听函数。不传此参数则移除所有监听函数。

offWaitting

该 方法 使用方式为 InnerAudioContext.offWaiting(function listener)

- **功能说明：**移除音频加载中事件的监听函数。
- **参数及说明：**function listener，音频加载中事件的监听函数。

onSeeking

该 方法 使用方式为 InnerAudioContext.onSeeking(function listener)

- **功能说明：**监听音频进行跳转操作的事件。
- **参数及说明：**function listener，音频进行跳转操作的事件的监听函数。

offSeeking

该 方法 使用方式为 InnerAudioContext.offSeeking(function listener)

- **功能说明：**移除音频进行跳转操作的事件的监听函数。
- **参数及说明：**function listener，onSeeking 传入的监听函数。不传此参数则移除所有监听函数。

onSeeked

该 方法 使用方式为 InnerAudioContext.onSeeked(function listener)

- **功能说明：**监听音频完成跳转操作的事件。
- **参数及说明：**function listener，音频完成跳转操作的事件的监听函数。

offSeeked

该 方法 使用方式为 InnerAudioContext.offSeeked(function listener)

- **功能说明：**移除音频完成跳转操作的事件的监听函数。
- **参数及说明：**function listener，onSeeked 传入的监听函数。不传此参数则移除所有监听函数。

视频

最近更新时间：2023-12-06 15:14:27

saveVideoToPhotosAlbum

该 API 使用方法为 wx.saveVideoToPhotosAlbum(Object object)

- **功能说明：**保存视频系统相册。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
filePath	string	-	是	视频文件路径，可以是临时文件路径也可以是永久文件路径
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.saveVideoToPhotosAlbum({
  filePath: 'qqfile://xxx',
  success(res) {
    console.log(res.errMsg)
  },
})
```

createVideoContext

该 API 使用方法为 VideoContext wx.createVideoContext(string id, Object this)

- **功能说明：**创建 video 上下文 VideoContext 对象。
- **参数及说明：**
  - **string id：**<video> 组件的 id。
  - 在自定义组件下，当前组件实例的 this，以操作组件内 <video> 组件。
- **返回值：**VideoContext

chooseVideo

> 该 API 使用方法为 wx.chooseVideo(Object object)

- **功能说明：**拍摄视频或从手机相册中选视频。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
sourceType	Array.\	['album', 'camera']	否	视频选择来源
compressed	boolean	ture	否	是否压缩所选择的视频文件
maxDuration	number	60	否	拍摄视频，最长拍摄时间，单位：秒
camera	string	‘back’	否	默认拉起的是前置或者后置摄像头。部分 Android 手机下由于系统 ROM 不支持无法生效
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.sourceType 的合法值**

值	说明
album	从相册选择视频
camera	使用相机拍摄视频

- **object.camera 的合法值**

值	说明
back	默认拉起后置摄像头
front	默认拉起前置摄像头

- **object.success 回调函数参数：**Object res。

属性	类型	说明
tempFilePath	string	选定视频的临时文件路径
duration	number	选定视频的时间长度
size	number	选定视频的数据量大小
height	number	返回选定视频的高度
width	number	返回选定视频的宽度
thumbTempFilePath	string	视频缩略图临时文件路径

- **示例代码**

```
wx.chooseVideo({
  sourceType: ['album', 'camera'],
  maxDuration: 60,
  camera: 'back',
  success(res) {
    console.log(res.tempFilePath)
  },
})
```

compressVideo

> 该 API 使用方法为 wx.compressVideo(Object object)

- **功能说明：**压缩视频接口。开发者可指定压缩质量 quality 进行压缩。当需要更精细的控制时，可指定 bitrate、fps、和 resolution，当 quality 传入时，这三个参数将被忽略。
- **参数及说明：**Object object。

属性	类型	必填	说明
src	string	是	视频文件路径，可以是临时文件路径也可以是永久文件路径
quality	string	是	压缩质量，合法值为： • low：低 • medium：中

			• high：高
bitrate	number	是	码率，单位 kbps
fps	number	是	帧率
resolution	number	是	相对于原视频的分辨率比例，取值范围(0, 1]
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
tempFilePath	string	压缩后的临时文件地址
size	string	压缩后的大小，单位 kB

chooseMedia

> 该 方法 使用方式为 wx.chooseMedia(Object object)

- **功能说明：** 拍摄或从手机相册中选择图片或视频。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
count	number	9	否	最多可以选择的文件个数，基础库2.25.0前，最多可支持9个文件，2.25.0及以后最多可支持20个文

				件
mediaType	Array.<string>	['image'，'video']	否	文件类型，合法值为： • image：只能拍摄图片或从相册选择图片 • video：只能拍摄或从相册选择视频 • mix：可同事选择图片和视频
sourceType	Array.<string>	['album', 'camera']	否	图片和视频选择的来源，合法值为： • album：从相册选择 • camera：使用相机拍摄
maxDuratuib	number	10	否	是否压缩所选文件，基础库2.25.0前仅对 mediaType 为 image 时有效，2.25.0及以后对全量 mediaType 有效
camera	string	‘back’	否	仅在 sourceType 为 camera 时生效，使用前置或后置摄像头，合法值为： • back：使用后置摄像头 • front：使用前置摄像头
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
tempFiles	Array.<Object>	本地临时文件列表
type	string	文件类型，有效值有 image 、video、mix

tempFiles 结构属性

结构属性	类型	说明
tempFilePath	string	本地临时文件路径（本地目录）
size	number	本地临时文件大小，单位 B
duration	number	视频的时间长度

height	number	视频的高度
width	number	视频的宽度
thumbTempFilePath	string	视频缩略图临时文件路径
fileType	string	文件类型，合法值有： • image：图片 • video：视频

- **示例代码**

```
wx.chooseMedia({
  count: 9,
  mediaType: ['image','video'],
  sourceType: ['album', 'camera'],
  maxDuration: 30,
  camera: 'back',
  success(res) {
    console.log(res.tempFiles.tempFilePath)
    console.log(res.tempFiles.size)
  }
})
```

VideoContext

- **功能说明：**VideoContext 实例，可通过 wx.createVideoContext 获取。videoContext 通过 id 跟一个 <video> 组件绑定，操作对应的 <video> 组件。
- **示例代码**

  对应的 WXML 文件：

```
<view class="section tc">
  <video
    id="myVideo"
    src="https://qzonestyle.gtimg.cn/qzone/qzact/act/external/qq-video/qq-
video.mp4"
    enable-danmu
    danmu-btn
    controls
  ></video>
  <view class="btn-area">
    <input bindblur="bindInputBlur" />
    <button bindtap="bindSendDanmu">发送弹幕</button>
```

```
  </view>
</view>
```

对应的 js文件：

```
function getRandomColor() {
  const rgb = []
  for (let i = 0; i < 3; ++i) {
    let color = Math.floor(Math.random() * 256).toString(16)
    color = color.length == 1 ? '0' + color : color
    rgb.push(color)
  }
  return '#' + rgb.join('')
}

Page({
  onReady(res) {
    this.videoContext = wx.createVideoContext('myVideo')
  },
  inputValue: '',
  bindInputBlur(e) {
    this.inputValue = e.detail.value
  },
  bindSendDanmu() {
    this.videoContext.sendDanmu({
      text: this.inputValue,
      color: getRandomColor(),
    })
  },
})
```

方法集

exitFullScreen

该 方法 使用方式为 VideoContext.exitFullScreen()

功能说明： 退出全屏。

exitBackgroundPlayback

该 方法 使用方式为 VideoContext.exitBackgroundPlayback()

功能说明： 退出后台音频播放模式。

exitPictureInPicture

> 该 方法 使用方式为 VideoContext.exitPictureInPicture(Object object)

- **功能说明：**退出小窗，该方法可在任意页面调用。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

hideStatusBar

> 该 方法 使用方式为 VideoContext.hideStatusBar()

功能说明：隐藏状态栏，仅在 iOS 全屏下有效。

pause

> 该 方法 使用方式为 VideoContext.pause()

功能说明：暂停视频。

play

> 该 方法 使用方式为 VideoContext.play()

功能说明：播放视频。

requestFullScreen

> 该 方法 使用方式为 VideoContext.requestFullScreen(Object object)

- **功能说明：**进入全屏。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明

direction	number	-	否	设置全屏时视频的方向，不指定则根据宽高比自动判断

- **object.direction 的合法值**

值	说明
0	正常竖向
90	屏幕逆时针 90 度
-90	屏幕顺时针 90 度

requestBackgroundPlayback

> 该 方法 使用方式为 VideoContext.requestBackgroundPlayback()

功能说明：进入后台音频播放模式。

seek

> 该 方法 使用方式为 VideoContext.seek(number position)

- **功能说明：**跳转到指定位置。
- **参数及说明：**number position，跳转到的位置，单位：s。

sendDanmu

> 该 方法 使用方式为 VideoContext.sendDanmu(Object data)

- **功能说明：**发送弹幕。(iOS SDK 暂不支持弹幕功能)
- **参数：**Object data，弹幕内容。

属性	类型	默认值	必填	说明
text	string	-	是	弹幕文字
color	string	-	否	弹幕颜色

showStatusBar

> 该 方法 使用方式为 VideoContext.showStatusBar()

功能说明：显示状态栏，仅在 iOS 全屏下有效。

stop

> 该 方法 使用方式为 VideoContext.stop()

功能说明：停止视频。

透明视频

最近更新时间：2023-10-20 15:19:19

createAnimationVideo

该 API 使用方法为 AnimationVideoContext wx.createAnimationVideo(string id, Object this)

- **功能说明：**创建 animation-video 上下文 AnimationVideoContext 对象。
- **参数及说明：**
  - string id：<animation-video> 组件的 id。
  - Object this：在自定义组件下，当前组件实例的 this，以组件内 <animation-video> 组件。
- **返回值：**AnimationVideoContext 。

AnimationVideoContext

- **功能说明：**AnimationVideoContext 实例，可通过 wx.createAnimationVideo 获取。

AnimationVideoContext 通过 id 跟一个 <animation-video> 组件绑定，操作对应的 <animation-video> 组件。

- **示例代码**

```
<view class="wrap">
  <view class="card-area">
    <view class="top-description border-bottom">
      <view>功能</view>
      <view>play pause seek</view>
    </view>
    <view class="video-area">
      <animation-video
        id="myAnimationVideo"
        path="{{leftAlphaSrcPath}}"
        loop="{{loop}}"
        resource-width="800"
        resource-height="400"
        canvas-style="width:200px;height:200px"
        autoplay="{{autoplay}}"
        bindstarted="onStarted"
        bindended="onEnded"
      ></animation-video>
    </view>
    <button bindtap="play">
      播放
```

```
    </button>
    <button bindtap="pause">
      暂停
    </button>
    <button bindtap="seek">
      跳转到动画2S处
    </button>
  </view>
</view>
```

```
Page({
  onLoad() {
    this.createCtx();
  },
  createCtx() {
    this.myAnimationVideo = wx.createAnimationVideoContext('my-video');
  },
  play() {
    this.myAnimationVideo?.play();
  },
  pause() {
    this.myAnimationVideo?.pause();
  },
  seek() {
    this.myAnimationVideo?.seek(2);
  },
})
```

.play

> 该 方法 使用方式为 AnimationVideoContext.play()

- **功能说明：**播放视频。

.pause

> 该 方法 使用方式为 AnimationVideoContext.pause()

- **功能说明：**暂停视频。

.seek

> 该 方法 使用方式为 AnimationVideoContext.seek(number position)

- **功能说明：**跳转到指定位置。
- **参数及说明：**number position，跳转到的位置，单位 s。

相机

最近更新时间：2023-10-20 15:19:19

createCameraContext

该 API 使用方法为 CameraContext wx.createCameraContext()

说明：
使用该接口需同时在 camera 组件属性中指定 frame-size。

- **功能说明：**创建 camera 上下文 CameraContext 对象。
- **返回值：**CameraContext 。

CameraContext

- **功能说明：**CameraContext 实例，可通过 wx.createCameraContext 取。cameraContext 与页面内唯一的 <camera> 组件绑定，操作对应的 <camera> 组件。

.onCameraFrame

该 方法 使用方式为 CameraFrameListener CameraContext.onCameraFrame(onCameraFrameCallback callback)

- **功能说明：**获取 Camera 实时帧数据。
- **参数及说明：**function callback。
- **回调函数参数：**Object res。

属性	类型	说明
width	number	图像数据矩形的宽度
height	number	图像数据矩形的高度
data	ArrayBuffer	图像像素点数据，一维数组，每四项表示一个像素点的 rgba

- **返回值：**CameraFrameListener 。
- **示例代码**

```
const context = wx.createCameraContext()
const listener = context.onCameraFrame((frame) => {
  console.log(frame.data instanceof ArrayBuffer, frame.width, frame.height)
```

```
})
listener.start()
```

.setZoom

该 方法 使用方式为 CameraContext.setZoom(Object object)

- **功能说明：**设置缩放级别。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
zoom	number	–	是	缩放级别，范围[1, maxZoom]。zoom 可取小数，精确到小数后一位。maxZoom 可在 bindinitdone 返回值中获取
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
zoom	number	实际设置的缩放级别。由于系统限制，某些机型可能无法设置成指定值，会改用最接近的可设值。

.startRecord

该 方法 使用方式为 CameraContext.startRecord(Object object)

- **功能说明：**开始录像。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
timeoutCallback	function	–	否	超过30s或页面 `onHide` 时会结束录像

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.timeoutCallback 回调函数参数：** Object res。

属性	类型	说明
tempThumbPath	string	封面图片文件的临时路径
tempVideoPath	string	视频的文件的临时路径

.stopRecord

> 该 方法 使用方式为 CameraContext.startRecord(Object object)

- **功能说明：** 结束录像。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res。

属性	类型	说明
tempThumbPath	string	封面图片文件的临时路径
tempVideoPath	string	视频的文件的临时路径

.takePhoto

> 该 方法 使用方式为 CameraFrameListener.stop(Object object)

- **功能说明：**拍摄照片。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
quality	string	normal	否	成像质量
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.quality 的合法值**

值	说明
high	高质量
normal	普通质量
low	低质量

- **object.success 回调函数参数：**Object res。

属性	类型	说明
tempImagePath	string	照片文件的临时路径

CameraFrameListener

说明：

- CameraContext.onCameraFrame() 返回的监听器。
- 相关文档，可参见 camera 。

.start

该 方法 使用方式为 CameraFrameListener.start(Object object)

- **功能说明：**开始监听帧数据。

- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.stop

> 该 方法 使用方式为 CameraFrameListener.stop(Object object)

- **功能说明：** 停止监听帧数据。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

录音

最近更新时间：2023-11-10 15:39:26

startRecord

> 该 API 使用方法为 wx.startRecord(Object object)

- **功能说明：**开始录音。当主动调用 wx.stopRecord，或者录音超过1分钟时自动结束录音。当用户离开小程序时，此接口无法调用。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
tempFilePath	string	录音文件的临时路径 (本地路径)

- **示例代码**

```
wx.startRecord({
  success (res) {
    const tempFilePath = res.tempFilePath
  }
})
setTimeout(function () {
  wx.stopRecord() // 结束录音
}, 10000)
```

stopRecord

> 该 API 使用方法为 wx.stopRecord(Object object)

- **功能说明：**停止录音。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.startRecord({
  success (res) {
    const tempFilePath = res.tempFilePath
  }
})
setTimeout(function () {
  wx.stopRecord() // 结束录音
}, 10000)
```

getRecorderManager

> 该 API 使用方法为 RecorderManager wx.getRecorderManager()

- **功能说明：**获取全局唯一的录音管理器 RecorderManager。
- **返回值：**RecorderManager 。

RecorderManager

.onError

> 该 方法 使用方式为 RecorderManager.onError(function listener)

- **功能说明：**监听录音错误事件。
- **参数及说明：**function listener，录音错误事件的监听函数。Object res 参数。

属性	类型	说明
errMsg	string	错误信息

.onFrameRecorded

该 方法 使用方式为 RecorderManager.onFrameRecorded(function listener)

- **功能说明：**监听已录制完指定帧大小的文件事件。如果设置了 frameSize，则会回调此事件。
- **参数及说明：**function listener，已录制完指定帧大小的文件事件的监听函数。Object res 参数。

属性	类型	说明
frameBuffer	ArrayBuffer	录音分片数据
isLastFrame	boolean	当前帧是否正常录音结束前的最后一帧

.onInterruptionBegin

该 方法 使用方式为 RecorderManager.onInterruptionBegin(function listener)

- **功能说明：**监听录音因为受到系统占用而被中断开始事件。以下场景会触发此事件:微信语音聊天、微信视频聊天。此事件触发后，录音会被暂停。pause 事件在此事件后触发。
- **参数及说明：**function listener，录音因为受到系统占用而被中断开始事件的监听函数。

.onInterruptionEnd

该 方法 使用方式为 RecorderManager.onInterruptionEnd(function listener)

- **功能说明：**监听录音中断结束事件。在收到 interruptionBegin 事件之后，小程序内所有录音会暂停，收到此事件之后才可再次录音成功。
- **参数及说明：**function listener，录音中断结束事件的监听函数。

.onPause

该 方法 使用方式为 RecorderManager.onPause(function listener)

- **功能说明：**监听录音暂停事件。
- **参数及说明：**function listener，录音暂停事件的监听函数。

.onResume

该 方法 使用方式为 RecorderManager.onResume(function listener)

- **功能说明：**监听录音继续事件。
- **参数及说明：**function listener，录音继续事件的监听函数。

.onStart

> 该 方法 使用方式为 RecorderManager.start(Object object)

- **功能说明：**监听录音开始事件。
- **参数及说明：**function listener，录音开始事件的监听函数。

.onStop

> 该 方法 使用方式为 RecorderManager.onStop(function listener)

- **功能说明：**监听录音结束事件。
- **参数及说明：**function listener，录音结束事件的监听函数。Object res 参数。

属性	类型	说明
tempFilePath	string	录音文件的临时路径 (本地路径)
duration	number	录音总时长，单位：ms
fileSize	number	录音文件大小，单位：Byte

.pause

> 该 方法 使用方式为 RecorderManager.pause()

功能说明：暂停录音。

.resume

> 该 方法 使用方式为 RecorderManager.resume()

功能说明：继续录音。

.start

> 该 方法 使用方式为 RecorderManager.start(Object object)

- **功能说明：**开始录音。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明

duration	number	60000	否	录音的时长，单位 ms，最大值 600000（10 分钟）
sampleRate	number	80000	否	采样率（pc不支持）
numberOfChannels	number	2	否	录音通道数
encodeBitRate	number	48000	否	编码码率，有效值见下表格
format	string	aac	否	音频格式
frameSize	number	-	否	指定帧大小，单位 KB。传入 frameSize 后，每录制指定帧大小的内容后，会回调录制的文件内容，不指定则不会回调。暂仅支持 mp3、pcm 格式
audioSource	string	auto	否	指定录音的音频输入源

- **采样率与编码码率限制**

每种采样率有对应的编码码率范围有效值，设置不合法的采样率或编码码率会导致录音失败，具体对应关系如下表。

采样率	编码码率
8000	16000 ~ 48000
11025	16000 ~ 48000
12000	24000 ~ 64000
16000	24000 ~ 96000
22050	32000 ~ 128000
24000	32000 ~ 128000
32000	48000 ~ 192000
44100	64000 ~ 320000
48000	64000 ~ 320000

.stop

> 该 方法 使用方式为 RecorderManager.stop()

- **功能说明：**暂停录音。

背景音频

最近更新时间：2023-11-10 15:39:26

stopBackgroundAudio

> 该 API 使用方法为 wx.stopBackgroundAudio(Object object)

- **功能说明：**停止播放音乐。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

seekBackgroundAudio

> 该 API 使用方法为 wx.seekBackgroundAudio(Object object)

- **功能说明：**控制音乐播放进度。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
position	number	–	是	音乐位置，单位：秒
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.seekBackgroundAudio({
  position: 30
})
```

playBackgroundAudio

> 该 API 使用方法为 wx.playBackgroundAudio(Object object)

- **功能说明：**使用后台播放器播放音乐。对于宿主客户端来说，只能同时有一个后台音乐在播放。当用户离开小程序后，音乐将暂停播放；当用户在其他小程序占用了音乐播放器，原有小程序内的音乐将停止播放。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
dataUrl	string	–	是	音乐链接，目前支持的格式有 m4a, aac, mp3, wav
title	string	–	否	音乐标题
coverImgUrl	string	–	否	封面URL
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.playBackgroundAudio({
  dataUrl: '',
  title: '',
  coverImgUrl: ''
})
```

pauseBackgroundAudio

> 该 API 使用方法为 wx.pauseBackgroundAudio(Object object)

- **功能说明：**暂停播放音乐。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

onBackgroundAudioStop

> 该 API 使用方法为 wx.onBackgroundAudioStop(function listener)

- **功能说明：**监听音乐停止事件。
- **参数及说明：**function listener，音乐停止事件的监听函数。

onBackgroundAudioPlay

> 该 API 使用方法为 wx.onBackgroundAudioPlay(function listener)

- **功能说明：**监听音乐播放事件。
- **参数及说明：**function listener，音乐播放事件的监听函数。

onBackgroundAudioPause

> 该 API 使用方法为 wx.onBackgroundAudioPause(function listener)

- **功能说明：**监听音乐暂停事件。
- **参数及说明：**function listener，音乐暂停事件的监听函数。

getBackgroundAudioPlayerState

> 该 API 使用方法为 wx.getBackgroundAudioPlayerState(Object object)

- **功能说明：**获取后台音乐播放状态。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
duration	number	选定音频的长度（单位：s），只有在音乐播放中时返回
currentPosition	number	选定音频的播放位置（单位：s），只有在音乐播放中时返回 getBackgroundAudioManager
status	number	播放状态，合法值： • 0：暂停中 • 1：播放中 • 2：没有音乐播放
downloadPercent	number	音频的下载进度百分比，只有在音乐播放中时返回
dataUrl	string	歌曲数据链接，只有在音乐播放中时返回

- **示例代码**

```
wx.getBackgroundAudioPlayerState({
  success (res) {
    const status = res.status
    const dataUrl = res.dataUrl
    const currentPosition = res.currentPosition
    const duration = res.duration
    const downloadPercent = res.downloadPercent
  }
})
```

getBackgroundAudioManager

- **功能说明：**获取全局唯一的背景音频管理器。 小程序切入后台，如果音频处于播放状态，可以继续播放。但是后台状态不能通过调用API操纵音频的播放状态。

BackgroundAudioManager

BackgroundAudioManager 实例，可通过 wx.getBackgroundAudioManager 获取。

属性

类型	说明
string src	音频的数据源。默认为空字符串，当设置了新的 src 时，会自动开始播放，目前支持的格式有 m4a, aac, mp3, wav
number startTime	音频开始播放的位置，单位：s
string title	音频标题，用于原生音频播放器音频标题（必填）。原生音频播放器中的分享功能，分享出去的卡片标题，也将使用该值
string epname	专辑名，原生音频播放器中的分享功能，分享出去的卡片简介，也将使用该值
string singer	歌手名，原生音频播放器中的分享功能，分享出去的卡片简介，也将使用该值
string coverImgUrl	封面图 URL，用于做原生音频播放器背景图。原生音频播放器中的分享功能，分享出去的卡片配图及背景也将使用该图
string webUrl	页面链接，原生音频播放器中的分享功能，分享出去的卡片简介，也将使用该值
string protocol	音频协议。默认值为 'http'，设置 'hls' 可以支持播放 HLS 协议的直播音频
number playbackRate	播放速度。范围 0.5-2.0，默认为 1（Android 需要 6 及以上版本）
number duration	当前音频的长度（单位：s），只有在有合法 src 时返回（只读）
number currentTime	当前音频的播放位置（单位：s），只有在有合法 src 时返回（只读）
boolean paused	当前是否暂停或停止（只读）
number	音频已缓冲的时间，仅保证当前播放时间点到此时间点内容已缓冲（只读）

buffered	
string referrerPolicy	origin：发送完整的referrer；no-referrer：不发送。格式固定为 https://servicewechat.com/{appid}/{version}/page-frame.html，其中 {appid} 为小程序的 appid，{version} 为小程序的版本号，版本号为 0 表示为开发版、体验版以及审核版本，版本号为 devtools 表示为开发者工具，其余为正式版本；

方法集

onCanplay

该 方法 使用方式为 BackgroundAudioManager.onCanplay(function listener)

- **功能说明：**监听背景音频进入可播放状态事件。 但不保证后面可以流畅播放。
- **参数及说明：**function listener，背景音频进入可播放状态事件的监听函数。

onEnded

该 方法 使用方式为 BackgroundAudioManager.onEnded(function listener)

- **功能说明：**监听背景音频自然播放结束事件。
- **参数及说明：**function listener，背景音频自然播放结束事件的监听函数。

onError

该 方法 使用方式为 BackgroundAudioManager.onError(function listener)

- **功能说明：**监听背景音频播放错误事件。
- **参数及说明：**function listener，背景音频播放错误事件的监听函数。

onNext

该 方法 使用方式为 BackgroundAudioManager.onNext(function listener)

- **功能说明：**监听用户在系统音乐播放面板点击下一曲事件（仅 iOS）。
- **参数及说明：**function listener，用户在系统音乐播放面板点击下一曲事件的监听函数。

onPause

该 方法 使用方式为 BackgroundAudioManager.onPause(function listener)

- **功能说明：**监听背景音频暂停事件。
- **参数及说明：**function listener，背景音频暂停事件的监听函数。

onPlay

该 方法 使用方式为 BackgroundAudioManager.onPlay(function listener)

- **功能说明：**监听背景音频播放事件。
- **参数及说明：**function listener，背景音频播放事件的监听函数。

onPrev

该 方法 使用方式为 BackgroundAudioManager.onPrev(function listener)

- **功能说明：**监听用户在系统音乐播放面板点击上一曲事件（仅 iOS）。
- **参数及说明：**function listener，用户在系统音乐播放面板点击上一曲事件的监听函数。

onSeeked

该 方法 使用方式为 BackgroundAudioManager.onSeeked(function listener)

- **功能说明：**监听背景音频完成跳转操作事件。
- **参数及说明：**function listener，背景音频完成跳转操作事件的监听函数。

onSeeking

该 方法 使用方式为 BackgroundAudioManager.onSeeking(function listener)

- **功能说明：**监听背景音频开始跳转操作事件。
- **参数及说明：**function listener，背景音频开始跳转操作事件的监听函数。

onStop

该 方法 使用方式为 BackgroundAudioManager.onStop(function listener)

- **功能说明：**监听背景音频停止事件。
- **参数及说明：**function listener，背景音频停止事件的监听函数。

onTimeUpdate

该 方法 使用方式为 BackgroundAudioManager.onTimeUpdate(function listener)

- **功能说明：**监听背景音频播放进度更新事件，只有小程序在前台时会回调。
- **参数及说明：**function listener，背景音频播放进度更新事件的监听函数。

onWaiting

该 方法 使用方式为 BackgroundAudioManager.onWaiting(function listener)

- **功能说明：**监听音频加载中事件。当音频因为数据不足，需要停下来加载时会触发。

- **参数及说明：**function listener，音频加载中事件的监听函数。

pause

> 该 方法 使用方式为 BackgroundAudioManager.pause()

- **功能说明：**暂停音乐
- **错误码**

错误码	错误信息	说明
10001	-	系统错误
10002	-	网络错误
10003	-	文件错误，请检查是否responseheader是否缺少Content-Length
10004	-	格式错误
-1	-	未知错误

play

> 该 方法 使用方式为 BackgroundAudioManager.play()

功能说明：播放音乐。

seek

> 该 方法 使用方式为 BackgroundAudioManager.seek(number currentTime)

- **功能说明：**跳转到指定位置
- **参数及说明：**number currentTime，跳转的位置，单位 s。精确到小数点后3位，即支持 ms 级别精确度。
- **错误码**

错误码	错误信息	说明
10001	-	系统错误
10002	-	网络错误

10003	–	文件错误，请检查是否 responseheader 是否缺少 Content-Length
10004	–	格式错误
-1	–	未知错误

stop

> 该 方法 使用方式为 BackgroundAudioManager.stop()

- **功能说明：** 停止音乐
- **错误码**

错误码	错误信息	说明
10001	–	系统错误
10002	–	网络错误
10003	–	文件错误，请检查是否 responseheader 是否缺少 Content-Length
10004	–	格式错误
-1	–	未知错误

示例代码

```
const backgroundAudioManager = wx.getBackgroundAudioManager()

backgroundAudioManager.title = '此时此刻'
backgroundAudioManager.epname = '此时此刻'
backgroundAudioManager.singer = '许巍'
backgroundAudioManager.coverImgUrl =
'http://y.gtimg.cn/music/photo_new/T002R300x300M000003rsKF44GyaSk.jpg?
max_age=2592000'
// 设置了 src 之后会自动播放
```

```
backgroundAudioManager.src =
'http://ws.stream.qqmusic.qq.com/M500001VfvsJ21xFqb.mp3?
guid=ffffffff82def4af4b12b3cd9337d5e7&uin=346897220&vkey=6292F51E1E384E06
1FF02C31F716658E5C81F5594D561F2E88B854E81CAAB7806D5E4F103E55D33C16F
3FAC506D1AB172DE8600B37E43FAD&fromtag=46'
```

富文本

最近更新时间：2023-11-10 15:39:29

EditorContext

功能说明：EditorContext 实例，可通过 wx.createSelectorQuery 获取。EditorContext 通过 id 跟一个 editor 组件绑定，操作对应的 editor 组件。

.blur

该 方法 使用方式为 EditorContext.blur()

- **功能说明：**编辑器失焦，同时收起键盘。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.clear

该 方法 使用方式为 EditorContext.clear()

- **功能说明：**清空编辑器内容。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
comple	functi	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

te	on			

.format

该 方法 使用方式为 EditorContext.format(string name, string value)

- **功能说明：**修改样式。
- **参数及说明：**Object object。

.getContents

该 方法 使用方式为 EditorContext.getContents()

- **功能说明：**获取编辑器内容。
- **参数及说明：**Object object。

.getSelectionText

该 方法 使用方式为 EditorContext.getSelectionText(Object object)

- **功能说明：**获取编辑器已选区域内的纯文本内容。当编辑器失焦或未选中一段区间时，返回内容为空。
- **参数及说明：**Object object。

.insertDivider

该 方法 使用方式为 EditorContext.insertDivider()

- **功能说明：**插入分割线。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.insertImage

该 方法 使用方式为 EditorContext.insertImage(Object object)

- **功能说明：**插入图片。
  - 地址为临时文件时，获取的编辑器 html 格式内容中标签增加属性 data-local，deta 格式内容中图片 attributes 属性增加 data-local 字段，该值为传入的临时文件地址。
  - 开发者可选择在提交阶段上传图片到服务器，获取到网络地址后进行替换。替换时对于 html 内容应替换掉的 src 值，对于 delta 内容应替换掉 insert { image: abc } 值。
- **参数及说明：**Object object。

.insertText

该 方法 使用方式为 EditorContext.insertText(Object object)

- **功能说明：**覆盖当前选区，设置一段文本。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
text	string	–	否	文本内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.redo

该 方法 使用方式为 EditorContext.redo(Object object)

- **功能说明：**恢复。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	functi	–	否	接口调用失败的回调函数

	on			
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.removeFormat

> 该 方法 使用方式为 EditorContext.removeFormat(Object object)

- **功能说明：**清除当前选区的样式。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.scrollIntoView

> 该 方法 使用方式为 EditorContext.scrollIntoView()

- **功能说明：**使得编辑器光标处滚动到窗口可视区域内。

.setContents

> 该 方法 使用方式为 EditorContext.setContents(Object object)

- **功能说明：**初始化编辑器内容，html 和 delta 同时存在时仅 delta 生效。
- **参数及说明：**Object object。

.undo

> 该 方法 使用方式为 EditorContext.undo()

- **功能说明：**撤销。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

画面录制器

最近更新时间：2023-11-10 15:39:29

createMediaRecorder

该 API 使用方法为 MediaRecorder wx.createMediaRecorder(Object canvas, Object options)

- **功能说明：**创建 WebGL 画面录制器，可逐帧录制在 WebGL 上渲染的画面并导出视频文件。
- **参数及说明：**
  - Object canvas，WebGL 对象，通过 SelectorQuery 获取到的 node 对象或通过 wx.createOffscreenCanvas 创建的离屏 WebGL Canvas 对象。
  - Object options。

属性	类型	默认值	必填	说明
duration	number	600	否	指定录制的时长（s），到达自动停止。最大7200，最小5
videoBitsPerSecond	number	1000	否	视频比特率（kbps），最小值600，最大值3000
gop	number	12	否	视频关键帧间隔
fps	number	24	否	视频 fps
width	number	canvas.width	否	画布录制宽度
height	number	canvas.height	否	画布录制高度

- **返回值：**MediaRecorder 。
- **示例代码**

```
// 准备 canvas 对象，可以是 wxml 声明的 node 对象
const canvas = await new Promise(resolve => {
  wx.createSelectQuery().select('#canvas').node(res =>
resolve(res.node)).exec()
})
```

```
// 也可以是 wx.createOffscreenCanvas 创建的离屏 canvas
const canvas = wx.createOffscreenCanvas()
canvas.width = 300
canvas.height = 150

// 准备一个 canvas 绘制函数，这里使用 three.js
const THREE = require('threejs-
miniprogram').createScopedThreejs(canvas)
const camera = new THREE.PerspectiveCamera(70, canvas.width /
canvas.height, 1, 1000)
const scene = new THREE.Scene()
const texture = await new Promise(resolve => new
THREE.TextureLoader().load('./test.png', resolve)) // 准备一个图片加载为贴图
const geometry = new THREE.BoxBufferGeometry(200, 200, 200)
const material = new THREE.MeshBasicMaterial({ map: texture })
const mesh = new THREE.Mesh(geometry, material)
camera.position.z = 400;
scene.add(mesh)
const renderer = new THREE.WebGLRenderer({ antialias: false })
renderer.setPixelRatio(1)
renderer.setSize(canvas.width, canvas.height)

// canvas 绘制函数
const render = () => {
  mesh.rotation.x += 0.005
  mesh.rotation.y += 0.1
  renderer.render(scene, camera)
}

// 创建 mediaRecorder
const fps = 30
const recorder = wx.createMediaRecorder(canvas, {
  fps,
})

// 启动 mediaRecorder
await recorder.start()

// 录制 5s 的视频
let frames = fps * 5
// 逐帧绘制
while (frames--) {
  await recorder.requestFrame()
  render()
}
```

```
// 绘制完成，生成视频
const {tempFilePath} = await recorder.stop()
// 销毁 mediaRecorder
recorder.destroy()
```

- **低版本异步接口兼容：**对基础库2.16.1版本前的 mediaRecorder，所有的接口都没有返回 Promise 对象，若需要兼容低版本，则可采用如下方式的写法：

```
// 启动 mediaRecorder
await new Promise(resolve => {
  recorder.on('start', resolve)
  recorder.start()
})

// 逐帧绘制
while (frames--) {
  await new Promise(resolve => recorder.requestFrame(resolve))
  render()
}

// 绘制完成，生成视频
const {tempFilePath} = await new Promise(resolve => {
  recorder.on('stop', resolve)
  recorder.stop()
})
```

MediaRecorder

功能说明：可通过 wx.createMediaRecorder 创建。MediaRecorder WebGL 画面录制器，可以进行录制相关操作，在结束录制时导出视频文件。

.destroy

> 该 方法 使用方式为 Promise MediaRecorder.destroy()

- **功能说明：**销毁录制器。
- **返回值：**Promise。

.off

> 该 方法 使用方式为 MediaRecorder.off(string eventName, function callback)

- **功能说明：**取消监听录制事件。当对应事件触发时，该回调函数不再执行。
- **参数及说明：**
  - string eventName，事件名。
  - function callback，事件触发时执行的回调函数。

.on

> 该 方法 使用方式为 MediaRecorder.on(string eventName, function callback)

- **功能说明：**注册监听录制事件的回调函数。当对应事件触发时，回调函数会被执行。
- **参数及说明：**string eventName，事件名，其合法值为如下。

值	说明
start	录制开始事件
stop	录制结束事件
pause	录制暂停事件
resume	录制继续事件
timeupdate	录制事件更新事件

.pause

> 该 方法 使用方式为 Promise MediaRecorder.pause()

- **功能说明：**暂停录制。
- **返回值：**Promise。

.requestFrame

> 该 方法 使用方式为 Promise MediaRecorder.requestFrame(function callback)

- **功能说明：**请求下一帧录制，在 callback 里完成一帧渲染后开始录制当前帧。
- **参数及说明：**function callback。
- **返回值：**Promise。

.resume

> 该 方法 使用方式为 Promise MediaRecorder.resume()

- **功能说明：**恢复录制。
- **返回值：**Promise。

.start

> 该 方法 使用方式为 Promise MediaRecorder.start()

- **功能说明：**开始录制。
- **返回值：**Promise。

.stop

> 该 方法 使用方式为 Promise MediaRecorder.stop()

- **功能说明：**暂停录制。
- **返回值：**Promise。

文件

最近更新时间：2023-11-10 15:39:26

saveFile

> 该 API 使用方法为 wx.saveFile(Object object)

> ⚠ **注意：**
> 本地文件存储的大小限制为 10M。

- **功能说明：**保存文件到本地。

  > ⚠ **注意：**
  > saveFile 会把临时文件移动，因此调用成功后传入的 tempFilePath 将不可用。

- **参数及说明：Object object**

属性	类型	默认值	必填	说明
tempFilePath	string	-	是	需要保存的文件的临时路径
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
savedFilePath	number	存储后的文件路径

- **示例代码：**

```
wx.chooseImage({
success(res) {
  const tempFilePaths = res.tempFilePaths
```

```
    wx.saveFile({
      tempFilePath: tempFilePaths[0],
      success(res) {
        const savedFilePath = res.savedFilePath
      }
    })
  }
})
```

removeSavedFile

> 该 API 使用方法为 wx.removeSavedFile(Object object)

- **功能说明：**删除本地缓存文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	需要删除的文件路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.getSavedFileList({
  success(res) {
    if (res.fileList.length > 0) {
      wx.removeSavedFile({
        filePath: res.fileList[0].filePath,
        complete(res) {
          console.log(res)
        }
      })
    }
  }
})
```

openDocument

> 该 API 使用方法为 wx.openDocument(Object object)

- **功能说明：**新开页面打开文档。
- **参数及说明：Object object**

属性	类型	默认值	必填	说明
filePath	string	–	是	文件路径，可通过 downloadFile 获得
fileType	string	–	否	文件类型，指定文件类型打开文件
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fileType 的合法值**

值	说明
doc	doc 格式
docx	docx 格式
xls	xls 格式
xlsx	xlsx 格式
ppt	ppt 格式
pptx	pptx 格式
pdf	pdf 格式

- **示例代码：**

```
wx.downloadFile({
  // 示例 url，并非真实存在
```

```
  url: 'https://example.com/somefile.pdf',
  success(res) {
    const filePath = res.tempFilePath
    wx.openDocument({
      filePath,
      success(res) {
        console.log('打开文档成功')
      }
    })
  }
})
```

getSavedFileList

> 该 API 使用方法为 wx.getSavedFileList(Object object)

- **功能说明：**获取该小程序下已保存的本地缓存文件列表。
- **参数及说明：Object object**

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：Object res**

属性	类型	说明
fileList	Array.\	文件数组，每一项是一个 FileItem

- **res.fileList 的结构**

属性	类型	说明
filePath	string	本地路径
size	number	本地文件大小，以字节为单位

createTime	number	文件保存时的时间戳，从1970/01/01 08:00:00 到当前时间的秒数

- **示例代码：**

```
wx.getSavedFileList({
  success(res) {
    console.log(res.fileList)
  }
})
```

getSavedFileInfo

> 该 API 使用方法为 wx.getSavedFileInfo(Object object)

- **功能说明：**获取本地文件的文件信息。此接口只能用于获取已保存到本地的文件，若需要获取临时文件信息，请使用 wx.getFileInfo() 接口。
- **参数及说明：Object object**

属性	类型	默认值	必填	说明
filePath	string	–	是	文件路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
size	number	文件大小，单位 B
createTime	number	文件保存时的时间戳，从1970/01/01 08:00:00 到该时刻的秒数

- **示例代码：**

```
wx.getSavedFileList({
  success(res) {
    console.log(res.fileList)
  }
})
```

getFileSystemManager

> 该 API 使用方法为 FileSystemManager wx.getFileSystemManager()

- **功能说明：**获取全局唯一的文件管理器，返回值 FileSystemManager 文件管理器。

getFileInfo

> 该 API 使用方法为 wx.getFileInfo(Object object)

- **功能说明：**获取文件信息。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	本地文件路径
digestAlgorithm	string	'md5'	否	计算文件摘要的算法
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.digestAlgorithm 的合法值**

值	说明
md5	md5 算法
sha1	sha1 算法

- **object.success 回调函数参数：** Object res

属性	类型	说明
size	number	文件大小，以字节为单位
digest	string	按照传入的 digestAlgorithm 计算得出的文件摘要

- **示例代码：**

```
wx.getFileInfo({
  success(res) {
    console.log(res.size)
    console.log(res.digest)
  }
})
```

FileSystemManager

.access

> 该 方法 使用方式为 FileSystemManager.access(Object object)

- **功能说明：** 判断文件/目录是否存在。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
path	string	–	是	要判断是否存在的文件/目录路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail no such file or directory ${path}	文件/目录不存在

.accessSync

该 方法 使用方式为 FileSystemManager.accessSync(string path)

- **功能说明：**FileSystemManager.access 的同步版本。
- **参数及说明：**string path，要判断是否存在的文件/目录路径

.appendFile

该 方法 使用方式为 FileSystemManager.appendFile(Object object)

- **功能说明：**在文件结尾追加内容。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	-	是	要追加内容的文件路径
data	string/ArrayBuffer	-	是	要追加的文本或二进制数据
encoding	string	utf-8	否	指定写入文件的字符编码
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.encoding 的合法值**

值	说明
ascii	-

base64	–
binary	–
hex	–
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	–
latin1	–

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail no such file or directory, open ${filePath}	指定的 filePath 文件不存在
fail illegal operation on a directory, open "${filePath}"	指定的 filePath 是一个已经存在的目录
fail permission denied, open ${dirPath}	指定的 filePath 路径没有写权限
fail sdcard not mounted	指定的 filePath 是一个已经存在的目录

.appendFileSync

> 该 方法 使用方式为 FileSystemManager.appendFileSync(string filePath, string|ArrayBuffer data, string encoding)

- **功能说明：** FileSystemManager.appendFile 的同步版本。
- **参数及说明**
  - string filePath：要追加内容的文件路径。
  - string|ArrayBuffer data：要追加的文本或二进制数据。
  - string encoding：指定写入文件的字符编码。
- **encoding 的合法值**

值	说明

ascii	–
base64	–
binary	–
hex	–
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	–
latin1	–

.close

> 该 方法 使用方式为 FileSystemManager.close(Object object)

- **功能说明：**关闭文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
fd	string	–	是	需要被关闭的文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
const fs = wx.getFileSystemManager()
// 打开文件
fs.open({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+',
  success(res) {
    // 关闭文件
```

```
    fs.close({
      fd: res.fd
    })
  }
})
```

.closeSync

该 方法 使用方式为 undefined FileSystemManager.closeSync(Object object)

- **功能说明：**同步关闭文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
fd	string	–	是	需要被关闭的文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得

- **返回值：**undefined。
- **示例代码**

```
const fs = wx.getFileSystemManager()
const fd = fs.openSync({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+'
})

// 关闭文件
fs.closeSync({fd: fd})
```

.copyFile

该 方法 使用方式为 FileSystemManager.copyFile(Object object)

- **功能说明：**复制文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
srcPath	string	–	是	源文件路径，只可以是普通文件

destPath	string	–	是	目标文件路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail permission denied, copyFile ${srcPath} -> ${destPath}	指定目标文件路径没有写权限
fail no such file or directory, copyFile ${srcPath} -> ${destPath}	源文件不存在，或目标文件路径的上层目录不存在

.copyFileSync

> 该 方法 使用方式为 FileSystemManager.copyFileSync(string srcPath, string destPath)

- **功能说明：** FileSystemManager.copyFile 的同步版本。
- **参数及说明**
  - string srcPath：源文件路径，只可以是普通文件。
  - string destPath：目标文件路径。

.getFileInfo

> 该 方法 使用方式为 FileSystemManager.getFileInfo(Object object)

- **功能说明：** 获取该小程序下的本地临时文件或本地缓存文件信息。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	要读取的文件路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
size	number	文件大小，以字节为单位

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail file not exist	指定的 filePath 找不到文件

.fstat

> 该 方法 使用方式为 FileSystemManager.fstat(Object object)

- **功能说明：** 获取文件的状态信息。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
fd	string	–	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
success	function	–	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
stats	Stats	Stats 对象，包含了文件的状态信息

- **示例代码**

```
const fs = wx.getFileSystemManager()
// 打开文件
fs.open({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+',
  success(res) {
    // 获取文件的状态信息
    fs.fstat({
      fd: res.fd,
      success(res) {
        console.log(res.stats)
      }
    })
  }
})
```

.fstatSync

> 该 方法 使用方式为 Stats FileSystemManager.fstatSync(Object object)

- **功能说明：** 同步获取文件的状态信息。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
fd	string	–	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得

- **返回值：** Stats，Stats 对象，包含了文件的状态信息。

- **示例代码**

```
const fs = wx.getFileSystemManager()
const fd = fs.openSync({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+'
})
const stats = fs.fstatSync({fd: fd})
console.log(stats)
```

.ftruncate

> 该 方法 使用方式为 FileSystemManager.ftruncate(Object object)

- **功能说明：**对文件内容进行截断操作。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
fd	string	–	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
length	number	–	是	截断位置，默认0。如果 length 小于文件长度（单位：字节），则只有前面 length 个字节会保留在文件中，其余内容会被删除；如果 length 大于文件长度，则会对其进行扩展，并且扩展部分将填充空字节（'\0'）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
const fs = wx.getFileSystemManager()
// 打开文件
fs.open({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+',
```

```
  success(res) {
    // 对文件内容进行截断操作
    fs.ftruncate({
      fd: res.fd,
      length: 10, // 从第10个字节开始截断文件
      success(res) {
        console.log(res)
      }
    })
  }
})
```

.ftruncateSync

该 方法 使用方式为 undefined FileSystemManager.ftruncateSync(Object object)

- **功能说明：**对文件内容进行截断操作。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
fd	string	–	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
length	number	–	是	截断位置，默认0。如果 length 小于文件长度（单位：字节），则只有前面 length 个字节会保留在文件中，其余内容会被删除；如果 length 大于文件长度，则会对其进行扩展，并且扩展部分将填充空字节（'\0'）

- **返回值：**undefined。
- **示例代码**

```
const fs = wx.getFileSystemManager()
const fd = fs.openSync({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+'
})
fs.ftruncateSync({
  fd: fd,
  length: 10 // 从第10个字节开始截断文件
})
```

.getSavedFileList

> 该 方法 使用方式为 FileSystemManager.getSavedFileList(Object object)

- **功能说明：**获取该小程序下已保存的本地缓存文件列表。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
fileList	Array.\	文件数组

- **res.fileList 的结构**

属性	类型	说明
filePath	string	本地路径
size	number	本地文件大小，以字节为单位
createTime	number	文件保存时的时间戳，从1970/01/01 08:00:00 到当前时间的秒数

.mkdir

> 该 方法 使用方式为 FileSystemManager.mkdir(Object object)

- **功能说明：**创建目录。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明

dirPath	string	–	是	创建的目录路径
recursive	boolean	false	否	是否在递归创建该目录的上级目录后再创建该目录。如果对应的上级目录已经存在，则不创建该上级目录。如 dirPath 为 a/b/c/d 且 recursive 为 true，将创建 a 目录，再在 a 目录下创建 b 目录，以此类推直至创建 a/b/c 目录下的 d 目录。
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail no such file or directory ${dirPath}	上级目录不存在
fail permission denied, open ${dirPath}	指定的 filePath 路径没有写权限
fail file already exists ${dirPath}	有同名文件或目录

.mkdirSync

> 该 方法 使用方式为 FileSystemManager.mkdirSync(string dirPath, boolean recursive)

- **功能说明：** FileSystemManager.mkdir 的同步版本。
- **参数及说明：**
  - string dirPath：创建的目录路径。
  - boolean recursive：是否在递归创建该目录的上级目录后再创建该目录。如果对应的上级目录已经存在，则不创建该上级目录。如 dirPath 为 a/b/c/d 且 recursive 为 true，将创建 a 目录，再在 a 目录下

创建 b 目录，以此类推直至创建 a/b/c 目录下的 d 目录。

.open

该 方法 使用方式为 FileSystemManager.open(Object object)

- **功能说明：**打开文件，返回文件描述符。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	-	是	文件路径（本地路径）
flag	string	r	否	文件系统标志，默认值：r
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- flag 合法值

合法值	说明
a	打开文件用于追加。如果文件不存在，则创建该文件
ax	类似于 'a'，但如果路径存在，则失败
a+	打开文件用于读取和追加。如果文件不存在，则创建该文件
ax+	类似于 'a+'，但如果路径存在，则失败
as	打开文件用于追加（在同步模式中）。如果文件不存在，则创建该文件
as+	打开文件用于读取和追加（在同步模式中）。如果文件不存在，则创建该文件
r	打开文件用于读取。如果文件不存在，则会发生异常
r+	打开文件用于读取和写入。如果文件不存在，则会发生异常
w	打开文件用于写入。如果文件不存在则创建文件，如果文件存在则截断文件
wx	类似于 'w'，但如果路径存在，则失败

w+	打开文件用于读取和写入。 如果文件不存在则创建文件，如果文件存在则截断文件
wx+	类似于 'w+'，但如果路径存在，则失败

- **object.success 回调函数参数：** Object res

属性	类型	说明
fd	string	文件描述符

- **示例代码**

```
const fs = wx.getFileSystemManager()
fs.open({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+',
  success(res) {
    console.log(res.fd)
  }
})
```

.openSync

该 方法 使用方式为 string FileSystemManager.openSync(Object object)

- **功能说明：** 同步打开文件，返回文件描述符。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
filePath	string	-	是	文件路径（本地路径）
flag	string	r	否	文件系统标志，默认值：r

  - flag 合法值

合法值	说明
a	打开文件用于追加。 如果文件不存在，则创建该文件
ax	类似于 'a'，但如果路径存在，则失败
a+	打开文件用于读取和追加。 如果文件不存在，则创建该文件

ax+	类似于 'a+'，但如果路径存在，则失败
as	打开文件用于追加（在同步模式中）。如果文件不存在，则创建该文件
as+	打开文件用于读取和追加（在同步模式中）。如果文件不存在，则创建该文件
r	打开文件用于读取。如果文件不存在，则会发生异常
r+	打开文件用于读取和写入。如果文件不存在，则会发生异常
w	打开文件用于写入。如果文件不存在则创建文件，如果文件存在则截断文件
wx	类似于 'w'，但如果路径存在，则失败
w+	打开文件用于读取和写入。如果文件不存在则创建文件，如果文件存在则截断文件
wx+	类似于 'w+'，但如果路径存在，则失败

- **返回值：**string，文件描述符。
- **示例代码**

```
const fs = wx.getFileSystemManager()
const fd = fs.openSync({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+'
})
console.log(fd)
```

.read

> 该 方法 使用方式为 FileSystemManager.read(Object object)

- **功能说明：**读文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
fd	string	–	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
arrayBuffer	ArrayBuffer	–	是	数据写入的缓冲区，必须是 ArrayBuffer 实例
offset	number	0	否	缓冲区中的写入偏移量，默认0

length	number	0	否	要从文件中读取的字节数，默认0
position	number	–	否	文件读取的起始位置，如不传或传 null，则会从当前文件指针的位置读取。如果 position 是正整数，则文件指针位置会保持不变并从 position 读取文件
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
bytesRead	number	实际读取的字节数
arrayBuffer	ArrayBuffer	被写入的缓存区的对象，即接口入参的 arrayBuffer

- **示例代码**

```
const fs = wx.getFileSystemManager()
const ab = new ArrayBuffer(1024)
// 打开文件
fs.open({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+',
  success(res) {
    // 读取文件到 ArrayBuffer 中
    fs.read({
      fd: res.fd,
      arrayBuffer: ab,
      length: 10,
      success(res) {
        console.log(res)
      }
    })
  }
})
```

.readSync

该 方法 使用方式为 ReadResult FileSystemManager.readSync(Object object)

- **功能说明：**读文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
fd	string	-	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
arrayBuffer	ArrayBuffer	-	是	数据写入的缓冲区，必须是 ArrayBuffer 实例
offset	number	0	否	缓冲区中的写入偏移量，默认0
length	number	0	否	要从文件中读取的字节数，默认0
position	number	-	否	文件读取的起始位置，如不传或传 null，则会从当前文件指针的位置读取。如果 position 是正整数，则文件指针位置会保持不变并从 position 读取文件

- **返回值：**ReadResult，文件读取结果。
- **示例代码**

```
const fs = wx.getFileSystemManager()
const ab = new ArrayBuffer(1024)
const fd = fs.openSync({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+'
})
const res = fs.readSync({
  fd: fd,
  arrayBuffer: ab,
  length: 10
})
console.log(res)
```

.readCompressedFile

该 方法 使用方式为 FileSystemManager.readCompressedFile(Object object)

- **功能说明：**读取指定压缩类型的本地文件内容。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	要读取的文件的路径 (本地用户文件或代码包文件)
compressionAlgorithm	string	–	是	文件压缩类型，目前仅支持 br：brotli压缩文件
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
data	ArrayBuffer	文件内容

- **示例代码**

```
const fs = wx.getFileSystemManager()

// 异步接口
fs.readCompressedFile({
  filePath: '${wx.env.USER_DATA_PATH}/hello.br',
  compressionAlgorithm: 'br',
  success(res) {
    console.log(res.data)
  },
  fail(res) {
    console.log('readCompressedFile fail', res)
  }
})

// 同步接口
const data = fs.readCompressedFileSync({
```

```
  filePath: '${wx.env.USER_DATA_PATH}/hello.br',
  compressionAlgorithm: 'br',
})
console.log(data)
```

.readCompressedFileSync

> 该 方法 使用方式为 ArrayBuffer FileSystemManager.readCompressedFileSync(Object object)

- **功能说明：**同步读取指定压缩类型的本地文件内容。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	要读取的文件的路径 (本地用户文件或代码包文件)
compressionAlgorithm	string	–	是	文件压缩类型，目前仅支持 br：brotli 压缩文件

- **返回值：**ArrayBuffer，文件读取结果。
- **示例代码**

```
const fs = wx.getFileSystemManager()

// 异步接口
fs.readCompressedFile({
  filePath: '${wx.env.USER_DATA_PATH}/hello.br',
  compressionAlgorithm: 'br',
  success(res) {
    console.log(res.data)
  },
  fail(res) {
    console.log('readCompressedFile fail', res)
  }
})

// 同步接口
try {
  const data = fs.readCompressedFileSync({
    filePath: '${wx.env.USER_DATA_PATH}/hello.br',
    compressionAlgorithm: 'br',
```

```
  })
  console.log(data)
} catch (err) {
  console.log(err)
}
```

.readdir

该 方法 使用方式为 FileSystemManager.readdir(Object object)

- **功能说明：**读取目录内文件列表。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
dirPath	string	–	是	要读取的目录路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
files	Array.\	指定目录下的文件名数组。

- **object.fail 回调函数参数：**Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail no such file or directory ${dirPath}	目录不存在
fail not a directory ${dirPath}	dirPath 不是目录

fail permission denied, open ${dirPath}	指定的 filePath 路径没有读权限

.readdirSync

该 方法 使用方式为 Array.<string> FileSystemManager.readdirSync(string dirPath)

- **功能说明：**FileSystemManager.readdir 的同步版本。
- **参数及说明：**string dirPath，要读取的目录路径。
- **返回值：**Array.<string> files，指定目录下的文件名数组。

.readFile

该 方法 使用方式为 FileSystemManager.readFile(Object object)

- **功能说明：**读取本地文件内容。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	要读取的文件的路径
encoding	string	–	否	指定读取文件的字符编码，如果不传 encoding，则以 ArrayBuffer 格式读取文件的二进制内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.encoding 的合法值**

值	说明
ascii	–
base64	–

binary	–
hex	–
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	–
latin1	–

- **object.success 回调函数参数：** Object res

属性	类型	说明
data	string/ArrayBuffer	文件内容

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail no such file or directory, open ${filePath}	指定的 filePath 所在目录不存在
fail permission denied, open ${dirPath}	指定的 filePath 路径没有读权限

.readFileSync

> 该 方法 使用方式为 string|ArrayBuffer FileSystemManager.readFileSync(string filePath, string encoding)

- **功能说明：** FileSystemManager.readFile 的同步版本。
- **参数及说明：**
  - string filePath：要读取的文件的路径。
  - string encoding：指定读取文件的字符编码，如果不传 encoding，则以 ArrayBuffer 格式读取文件的二进制内容。
- **encoding 的合法值**

值	说明

ascii	-
base64	-
binary	-
hex	-
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	-
latin1	-

- **返回值：** string|ArrayBuffer data，文件内容。

.readZipEntry

> 该 方法 使用方式为 FileSystemManager.readZipEntry(Object object)

- **功能说明：** 读取压缩包内的文件。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
filePath	string	-	是	要读取的压缩包的路径 (本地路径)
encoding	string	-	否	统一指定读取文件的字符编码，只在 entries 值为"all"时有效。如果 entries 值为"all"且不传 encoding，则以 ArrayBuffer 格式读取文件的二进制内容
entries	Array.<Object>/'all'	-	是	要读取的压缩包内的文件列表（当传入"all" 时表示读取压缩包内所有文件）
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

  - **encoding 合法值**

值	说明
ascii	-
base64	-
binary	-
hex	-
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	-
latin1	-

- **entries 结构属性**

结构属性	类型	默认值	必填	说明
path	string	-	是	压缩包内文件路径
encoding	string	-	否	指定读取文件的字符编码，如果不传 encoding，则以 ArrayBuffer 格式读取文件的二进制内容
position	number	-	否	从文件指定位置开始读，如果不指定，则从文件头开始读。读取的范围应该是左闭右开区间 [position, position+length)。有效范围：[0, fileLength - 1]。单位：byte
length	number	-	否	指定文件的长度，如果不指定，则读到文件末尾。有效范围：[1, fileLength]。单位：byte

- **object.success 回调函数参数：** Object res

属性	类型	说明
entries	Object	文件读取结果。res.entries 是一个对象，key是文件路径，value是一个对象 FileItem ，表示该文件的读取结果。每个 FileItem 包含 data（文件内容）和 errMsg（错误信息）属性

- **entries 结构属性**

结构属性	类型	说明

path	string	文件路径

- **path 结构属性**

结构属性	类型	说明
data	string/ArrayBuffer	文件内容
errMsg	string	错误信息

- **示例代码**

```
const fs = wx.getFileSystemManager()
// 读取zip内某个或多个文件
fs.readZipEntry({
  filePath: 'wxfile://from/to.zip',
  entries: [{
    path: 'some_folder/my_file.txt', // zip内文件路径
    encoding: 'utf-8', // 指定读取文件的字符编码，如果不传 encoding，则以 ArrayBuffer 格式读取文件的二进制内容
    position: 0, // 从文件指定位置开始读，如果不指定，则从文件头开始读。读取的范围应该是左闭右开区间 [position, position+length)。有效范围：[0, fileLength - 1]。单位：byte
    length: 10000, // 指定文件的长度，如果不指定，则读到文件末尾。有效范围：[1, fileLength]。单位：byte
  }, {
    path: 'other_folder/orther_file.txt', // zip内文件路径
  }],
  success(res) {
    console.log(res.entries)
    // res.entries === {
    //     'some_folder/my_file.txt': {
    //         errMsg: 'readZipEntry:ok',
    //         data: 'xxxxxx'
    //     },
    //     'other_folder/orther_file.txt': {
    //         data: (ArrayBuffer)
    //     }
    // }
  },
  fail(res) {
```

```
    console.log(res.errMsg)
  },
})

// 读取zip内所有文件。允许指定统一的encoding。position、length则不再允许指定，
分别默认为0和文件长度
fs.readZipEntry({
  filePath: 'wxfile://from/to.zip',
  entries: 'all'
  encoding: 'utf-8', // 统一指定读取文件的字符编码，如果不传 encoding，则以
ArrayBuffer 格式读取文件的二进制内容
  success(res) {
    console.log(res.entries)
    // res.entries === {
    //     'some_folder/my_file.txt': {
    //         errMsg: 'readZipEntry:ok',
    //         data: 'xxxxxx'
    //     },
    //     'other_folder/orther_file.txt': {
    //         errMsg: 'readZipEntry:ok',
    //         data: 'xxxxxx'
    //     }
    // }
  },
  fail(res) {
    console.log(res.errMsg)
  },
})
```

.removeSavedFile

> 该 方法 使用方式为 FileSystemManager.removeSavedFile(Object object)

- **功能说明：** 删除该小程序下已保存的本地缓存文件。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	需要删除的文件路径
success	function	–	否	接口调用成功的回调函数
fail	functi	–	否	接口调用失败的回调函数

	on			
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail file not exist	指定的 tempFilePath 找不到文件

.rename

> 该 方法 使用方式为 FileSystemManager.rename(Object object)

- **功能说明：** 重命名文件。可以把文件从 oldPath 移动到 newPath。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
oldPath	string	–	是	源文件路径，可以是普通文件或目录
newPath	string	–	是	新文件路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail permission denied, rename ${oldPath} -> ${newPath}	指定源文件或目标文件没有写权限
fail no such file or directory, rename ${oldPath} -> ${newPath}	源文件不存在，或目标文件路径的上层目录不存在

.renameSync

该 方法 使用方式为 FileSystemManager.renameSync(string oldPath, string newPath)

- **功能说明：**FileSystemManager.rename 的同步版本。
- **参数及说明**
  - string oldPath：源文件路径，可以是普通文件或目录。
  - string newPath：新文件路径。

.rmdir

该 方法 使用方式为 FileSystemManager.rmdir(Object object)

- **功能说明：**删除目录。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
dirPath	string	-	是	要删除的目录路径
recursive	boolean	false	否	是否递归删除目录。如果为 true，则删除该目录和该目录下的所有子目录以及文件。
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：**Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail no such file or directory ${dirPath}	目录不存在
fail directory not empty	目录不为空
fail permission denied, open ${dirPath}	指定的 dirPath 路径没有写权限

.rmdirSync

> 该 方法 使用方式为 FileSystemManager.rmdirSync(string dirPath, boolean recursive)

- **功能说明：**FileSystemManager.rmdir 的同步版本。
- **参数及说明：**
  - string dirPath：要删除的目录路径。
  - boolean recursive：是否递归删除目录。如果为 true，则删除该目录和该目录下的所有子目录以及文件。

.saveFile

> 该 方法 使用方式为 FileSystemManager.saveFile(Object object)

- **功能说明：**保存临时文件到本地。此接口会移动临时文件，因此调用成功后，tempFilePath 将不可用。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
tempFilePath	string	-	是	临时存储文件路径
filePath	string	-	否	要存储的文件路径
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数

complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
savedFilePath	number	存储后的文件路径

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail tempFilePath file not exist	指定的 tempFilePath 找不到文件
fail permission denied, open "${filePath}"	指定的 filePath 路径没有写权限
fail no such file or directory "${dirPath}"	上级目录不存在

.saveFileSync

> 该 方法 使用方式为 number FileSystemManager.saveFileSync(string tempFilePath, string filePath)

- **功能说明：** FileSystemManager.saveFile 的同步版本。
- **参数及说明：**
  - string tempFilePath：临时存储文件路径。
  - string filePath：要存储的文件路径。
- **返回值：** number savedFilePath，存储后的文件路径。

.stat

> 该 方法 使用方式为 FileSystemManager.stat(Object object)

- **功能说明：** 获取文件 Stats 对象。
- **参数及说明：** Object object

属性	类型	默认	必	说明

		值	填	
path	string	–	是	文件/目录路径
recursive	boolean	false	否	是否递归获取目录下的每个文件的 Stats 信息
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res

属性	类型	说明
stats	Stats/Object	当 recursive 为 false 时，res.stats 是一个 Stats 对象。当 recursive 为 true 且 path 是一个目录的路径时，res.stats 是一个 Object，key 以 path 为根路径的相对路径，value 是该路径对应的 Stats 对象。

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail permission denied, open ${path}	指定的 path 路径没有读权限
fail no such file or directory ${path}	文件不存在

- **示例代码**

recursive 为 false 时

```
let fs = wx.getFileSystemManager()
fs.stat({
  path: `${wx.env.USER_DATA_PATH}/testDir`,
  success: res => {
```

```
    console.log(res.stats.isDirectory())
  }
})
```

recursive 为 true 时

```
fs.stat({
  path: `${wx.env.USER_DATA_PATH}/testDir`,
  recursive: true,
  success: res => {
    Object.keys(res.stats).forEach(path => {
      let stats = res.stats[path]
      console.log(path, stats.isDirectory())
    })
  }
})
```

.statSync

该 方法 使用方式为 Stats|Object FileSystemManager.statSync(string path, boolean recursive)

- **功能说明：**FileSystemManager.stat 的同步版本。
- **参数及说明：**
  - string path：文件/目录路径。
  - boolean recursive：是否递归获取目录下的每个文件的 Stats 信息。
- **返回值：**Stats|Object stats，当 recursive 为 false 时，res.stats 是一个 Stats 对象。当 recursive 为 true 且 path 是一个目录的路径时，res.stats 是一个 Object，key 以 path 为根路径的相对路径，value 是该路径对应的 Stats 对象。
- **示例代码**

  recursive 为 false 时

```
let fs = wx.getFileSystemManager()
fs.stat({
  path: `${wx.env.USER_DATA_PATH}/testDir`,
  success: res => {
    console.log(res.stats.isDirectory())
  }
})
```

recursive 为 true 时

```
fs.stat({
  path: `${wx.env.USER_DATA_PATH}/testDir`,
  recursive: true,
  success: res => {
    Object.keys(res.stats).forEach(path => {
      let stats = res.stats[path]
      console.log(path, stats.isDirectory())
    })
  }
})
```

.truncate

该 方法 使用方式为 FileSystemManager.truncate(Object object)

- **功能说明：**对文件内容进行截断操作。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	–	是	要截断的文件路径 (本地路径)
length	number	0	否	截断位置，默认0。如果 length 小于文件长度（字节），则只有前面 length 个字节会保留在文件中，其余内容会被删除；如果 length 大于文件长度，则会对其进行扩展，并且扩展部分将填充空字节（'\0'）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
const fs = wx.getFileSystemManager()
fs.truncate({
```

```
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  length: 10, // 从第10个字节开始截断
  success(res) {
    console.log(res)
  }
})
```

.truncateSync

该 方法 使用方式为 undefined FileSystemManager.truncateSync(Object object)

- **功能说明：**对文件内容进行截断操作 (truncate 的同步版本)
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string		是	要截断的文件路径 (本地路径)
length	number	0	否	截断位置，默认0。如果 length 小于文件长度（字节），则只有前面 length 个字节会保留在文件中，其余内容会被删除；如果 length 大于文件长度，则会对其进行扩展，并且扩展部分将填充空字节（'\0'）

- **返回值：**undefined。
- **示例代码**

```
const fs = wx.getFileSystemManager()
fs.truncateSync({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  length: 10, // 从第10个字节开始截断
})
```

.unlink

该 方法 使用方式为 FileSystemManager.unlink(Object object)

- **功能说明：**删除文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明

filePath	string	–	是	要删除的文件路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail permission denied, open ${path}	指定的 path 路径没有读权限
fail no such file or directory ${path}	文件不存在
fail operation not permitted, unlink ${filePath}	传入的 filePath 是一个目录

.unlinkSync

> 该 方法 使用方式为 FileSystemManager.unlinkSync(string filePath)

- **功能说明：** FileSystemManager.unlink 的同步版本。
- **参数及说明：** string filePath，要删除的文件路径。

.unzip

> 该 方法 使用方式为 FileSystemManager.unzip(Object object)

- **功能说明：** 解压文件。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
zipFileP	string	–	是	源文件路径，只可以是 zip 压缩文件

ath				
targetPath	string	–	是	目标目录路径
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail permission denied, unzip ${zipFilePath} -> ${destPath}	指定目标文件路径没有写权限
fail no such file or directory, unzip ${zipFilePath} -> "${destPath}	源文件不存在，或目标文件路径的上层目录不存在

.write

> 该 方法 使用方式为 FileSystemManager.write(Object object)

- **功能说明：** 写入文件。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
fd	string	–	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
data	string/ArrayBuffer	–	是	要写入的文本或二进制数据

offset	number	–	否	只在 data 类型是 ArrayBuffer 时有效，决定 arrayBuffe 中要被写入的部位，即 arrayBuffer 中的索引，默认0
length	number	–	否	只在 data 类型是 ArrayBuffer 时有效，指定要写入的字节数，默认为 arrayBuffer 从0开始偏移 offset 个字节后剩余的字节数
encoding	string	utf8	否	指定写入文件的字符编码
position	number	–	否	指定文件开头的偏移量，即数据要被写入的位置。当 position 不传或者传入非 Number 类型的值时，数据会被写入当前指针所在位置
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **encoding 合法值**

值	说明
ascii	–
base64	–
binary	–
hex	–
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	–
latin1	–

- **object.success 回调函数参数：** Object res

属性	类型	说明
bytesWritten	number	实际被写入到文件中的字节数（注意，被写入的字节数不一定与被写入的字符串字符数相同）

- **示例代码**

```
const fs = wx.getFileSystemManager()
// 打开文件
fs.open({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+',
  success(res) {
    // 写入文件
    fs.write({
      fd: res.fd,
      data: 'some text',
      success(res) {
        console.log(res.bytesWritten)
      }
    })
  }
})
```

.writeSync

> 该 方法 使用方式为 FileSystemManager.writeFile(Object object)

- **功能说明：** 同步写入文件。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
fd	string	–	是	文件描述符。fd 通过 FileSystemManager.open 或 FileSystemManager.openSync 接口获得
data	string/ArrayBuffer	–	是	要写入的文本或二进制数据
offset	number	–	否	只在 data 类型是 ArrayBuffer 时有效，决定 arrayBuffe 中要被写入的部位，即 arrayBuffer 中的索引，默认0
length	number	–	否	只在 data 类型是 ArrayBuffer 时有效，指定要写入的字节数，默认为 arrayBuffer 从0开始偏移 offset 个字节后剩余的字节数
encod	string	utf8	否	指定写入文件的字符编码

ing				
position	number	–	否	指定文件开头的偏移量，即数据要被写入的位置。当 position 不传或者传入非 Number 类型的值时，数据会被写入当前指针所在位置
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **encoding 合法值**

值	说明
ascii	–
base64	–
binary	–
hex	–
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	–
latin1	–

- **object.success 回调函数参数：** Object res

属性	类型	说明
bytesWritten	number	实际被写入到文件中的字节数（注意，被写入的字节数不一定与被写入的字符串字符数相同）

- **示例代码**

```
const fs = wx.getFileSystemManager()
const fd = fs.openSync({
  filePath: `${wx.env.USER_DATA_PATH}/hello.txt`,
  flag: 'a+'
})
const res = fs.writeSync({
```

```
  fd: fd,
  data: 'some text'
})
console.log(res.bytesWritten)
```

.writeFile

该 方法 使用方式为 FileSystemManager.writeFile(Object object)

- **功能说明：**写文件。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
filePath	string	-	是	要写入的文件路径
data	string/ArrayBuffer	-	是	要写入的文本或二进制数据
encoding	string	utf8	否	指定写入文件的字符编码
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.encoding 的合法值**

值	说明
ascii	-
base64	-
binary	-
hex	-
ucs2/ucs-2/utf16le/utf-16le	以小端序读取

utf-8/utf8	-
latin1	-

- **object.fail 回调函数参数：** Object res

属性	类型	说明
errMsg	string	错误信息

- **res.errMsg 的合法值**

值	说明
fail no such file or directory, open ${filePath}	指定的 filePath 所在目录不存在
fail permission denied, open ${dirPath}	指定的 filePath 路径没有写权限

.writeFileSync

> 该 方法 使用方式为 FileSystemManager.writeFileSync(string filePath, string|ArrayBuffer data, string encoding)

- **功能说明：** FileSystemManager.writeFile 的同步版本。
- **参数及说明：**
  - string filePath：要写入的文件路径。
  - string|ArrayBuffer data：要写入的文本或二进制数据。
  - string encoding：指定写入文件的字符编码。
- **encoding 的合法值**

值	说明
ascii	-
base64	-
binary	-
hex	-
ucs2/ucs-2/utf16le/utf-16le	以小端序读取
utf-8/utf8	-
latin1	-

错误码

> **说明：**
> 若无特殊说明，错误码均以下表为准。

错误码	错误信息	说明
1300001	operation not permitted	操作不被允许（例如，filePath 预期传入一个文件而实际传入一个目录）
1300002	no such file or directory ${path}	文件/目录不存在，或者目标文件路径的上层目录不存在
1300005	Input/output error	输入输出流不可用
1300009	bad file descriptor	无效的文件描述符
1300013	permission denied	权限错误，文件是只读或只写
1300014	Path permission denied	传入的路径没有权限
1300020	not a directory	dirPath 指定路径不是目录，常见于指定的写入路径的上级路径为一个文件的情况
1300021	Is a directory	指定路径是一个目录
1300022	Invalid argument	无效参数，可以检查length或offset是否越界
1300036	File name too long	文件名过长
1300066	directory not empty	目录不为空
1300201	system error	系统接口调用失败
1300202	the maximum size of the file storage limit is exceeded	存储空间不足，或文件大小超出上限（上限100M）

1300203	base64 encode error	字符编码转换失败（例如 base64 格式错误）
1300300	sdcard not mounted	android sdcard 挂载失败
1300301	unable to open as fileType	无法以fileType打开文件
1301000	permission denied, cannot access file path	目标路径无访问权限（usr目录）
1301002	data to write is empty	写入数据为空
1301003	illegal operation on a directory	不可对目录进行此操作（例如，指定的 filePath 是一个已经存在的目录）
1301004	illegal operation on a package directory	不可对代码包目录进行此操作
1301005	file already exists ${dirPath}	已有同名文件或目录
1301006	value of length is out of range	传入的 length 不合法
1301007	value of offset is out of range	传入的 offset 不合法
1301009	value of position is out of range	position值越界
1301100	store directory is empty	store目录为空
1301102	unzip open file fail	压缩文件打开失败
1301103	unzip entry fail	解压单个文件失败
1301104	unzip fail	解压失败
1301111	brotli decompress fail	brotli解压失败（例如，指定的 compressionAlgorithm 与文件实际压缩格式不符）

1301112	tempFilePath file not exist	指定的 tempFilePath 找不到文件
1302001	fail permission denied	指定的 fd 路径没有读权限/没有写权限
1302002	excced max concurrent fd limit	fd数量已达上限
1302003	invalid flag	无效的flag
1302004	permission denied when open using flag	无法使用flag标志打开文件
1302005	array buffer does not exist	未传入arrayBuffer
1302100	array buffer is readonly	arrayBuffer只读

Stats

描述文件状态的对象。

属性

- number mode：文件的类型和存取的权限，对应 POSIX stat.st_mode。
- number size：文件大小，单位：B，对应 POSIX stat.st_size。
- number lastAccessedTime：文件最近一次被存取或被执行的时间，UNIX 时间戳，对应 POSIX stat.st_atime。
- number lastModifiedTime：文件最后一次被修改的时间，UNIX 时间戳，对应 POSIX stat.st_mtime。

方法集

.isDirectory

> 该 方法 使用方式为 boolean Stats.isDirectory()

- **功能说明：**判断当前文件是否一个目录。
- **返回值：**boolean，表示当前文件是否一个目录。

.isFile

> 该 方法 使用方式为 boolean Stats.isFile()

- **功能说明：**判断当前文件是否一个普通文件。
- **返回值：**boolean，表示当前文件是否一个普通文件。

ReadResult

- **功能说明：**文件读取结果。 通过 FileSystemManager.readSync 接口返回。
- **属性：**
  - number bytesRead，实际读取的字节数；
  - ArrayBuffer arrayBuffer，被写入的缓存区的对象，即接口入参的 arrayBuffer。

WriteResult

- **功能说明：**文件写入结果。 通过 FileSystemManager.writeSync 接口返回。
- **属性：**number bytesWritten，实际被写入到文件中的字节数（注意，被写入的字节数不一定与被写入的字符串字符数相同）。

开放接口

最近更新时间：2024-05-17 14:18:01

登录

login

> 该 API 使用方法为 wx.login(Object object)

- **功能说明：IDE 目前暂不支持，该 API 需要和宿主客户端联调，在真机上返回的内容是由宿主客户端提供，宿主的返回值可以遵循微信标准，也可以自定义返回内容。**
- **额外说明：在 IDE 中可以使用 mock 面板进行返回值的 mock。**
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	functioen	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.login({
  success(res) {
    console.log(res ,"--------------info, host app return");
  }
})
```

checkSession

> 该 API 使用方法为 wx.checkSession(Object object)

- **功能说明：IDE 目前暂不支持，该 API 需要和宿主客户端联调，在真机上返回的内容是由宿主客户端提供，宿主的返回值可以遵循微信标准，也可以自定义返回内容。**
- **额外说明：在 IDE 中可以使用 mock 面板进行返回值的 mock。**

- 检查登录态是否过期。通过 wx.login 接口获得的用户登录态拥有一定的时效性。用户越久未使用小程序，用户登录态越有可能失效。反之如果用户一直在使用小程序，则用户登录态一直保持有效。具体时效逻辑由官方维护，对开发者透明。开发者只需要调用 wx.checkSession 接口检测当前用户登录态是否有效。
- 登录态过期后开发者可以再调用 wx.login 获取新的用户登录态。调用成功说明当前 session_key 未过期，调用失败说明 session_key 已过期。

- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.checkSession({
  success () {
    //session_key 未过期，并且在本生命周期一直有效
  },
  fail () {
    // session_key 已经失效，需要重新执行登录流程
    wx.login() //重新登录
  }
})
```

账号信息

getAccountInfoSync

> 该 API 使用方法为 Object wx.getAccountInfoSync()

- **功能说明：**获取当前账号信息。线上小程序版本号仅支持在正式版小程序中获取，开发版和体验版中无法获取。
- **返回值：**Object，账号信息。

属性	类型	说明
miniProgram	Object	小程序账号信息
plugin	Object	插件账号信息（仅在插件中调用时包含

		这一项）

- **miniProgram 结构属性**

结构属性	类型	说明
appId	string	小程序 appId
envVersion	string	小程序版本，合法值为： • develop：开发版 • trial：体验版 • release：正式版
version	string	线上小程序版本号

- **plugin 结构属性**

结构属性	类型	说明
appId	string	插件 appId
version	string	插件版本号

用户信息

getUserProfile

> 该 API 使用方法为 wx.getUserProfile(Object object)

- **功能说明：IDE 目前暂不支持，该 API 需要和宿主客户端联调，在真机上返回的内容是由宿主客户端提供，宿主的返回值可以遵循微信标准，也可以自定义返回内容。**
- **额外说明：在 IDE 中可以使用 mock 面板进行返回值的 mock。**
- 获取用户信息。页面产生单击事件（例如 button 上 bindtap 的回调中）后才可调用，每次请求都会弹出授权窗口，用户同意后返回 userInfo。该接口用于替换 wx.getUserInfo，详见 用户信息接口调整说明 。
- **返回值：**Object object。

属性	类型	默认值	必填	说明
lang	string	en	否	显示用户信息的语言，合法值为： • en：英文 • zh_CN：简体中文 • zh_TW：繁体中文

desc	string	–	是	声明获取用户个人信息后的用途，不超过30个字符
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

getUserInfo

> 该 API 使用方法为 wx.getUserInfo(Object object)

- **功能说明：IDE 目前暂不支持，该 API 需要和宿主客户端联调，在真机上返回的内容是由宿主客户端提供，宿主的返回值可以遵循微信标准，也可以自定义返回内容。**
- **额外说明：在 IDE 中可以使用 mock 面板进行返回值的 mock。**
- 获取用户信息，在使用过程中需要用户授权 scope.userInfo。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
withCredentials	boolean		否	是否带上登录态信息。当 withCredentials 为 true 时，要求此前有调用过 wx.login 且登录态尚未过期，此时返回的数据会包含 encryptedData，iv 等敏感信息；当 withCredentials 为 false 时，不要求有登录态，返回的数据不包含 encryptedData，iv 等敏感信息
lang	string	en	否	显示用户信息的语言，合法值为： • en：英文 • zh_CN：简体中文 • zh_TW：繁体中文
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：Object res**

属性	类型	说明
userInfo	UserInfo	用户信息对象，不包含 openid 等敏感信息
rawData	string	不包括敏感信息的原始数据字符串，用于计算签名
signature	string	使用 sha1(rawData + sessionkey) 得到字符串，用于校验用户信息
encryptedData	string	包括敏感数据在内的完整用户信息的加密数据
iv	string	加密算法的初始向量
cloudID	string	敏感数据对应的云 ID，开通云开发的小程序才会返回，可通过云调用直接获取开放数据，详细见云调用直接获取开放数据

- **示例代码**

```
// 必须是在用户已经授权的情况下调用
wx.getUserInfo({
  success: function(res) {
    var userInfo = res.userInfo
    var nickName = userInfo.nickName
    var avatarUrl = userInfo.avatarUrl
    var gender = userInfo.gender //性别 0：未知、1：男、2：女
    var province = userInfo.province
    var city = userInfo.city
    var country = userInfo.country
  }
})
```

敏感数据有两种获取方式：

1. **使用加密数据解密算法。**
2. **使用云调用直接获取开放数据，获取得到的开放数据为以下 json 结构：**

```
{
  "openId": "OPENID",
  "nickName": "NICKNAME",
  "gender": GENDER,
```

```
  "city": "CITY",
  "province": "PROVINCE",
  "country": "COUNTRY",
  "avatarUrl": "AVATARURL",
  "unionId": "UNIONID",
  "watermark": {
    "appid":"APPID",
    "timestamp":TIMESTAMP
  }
}
```

- **小程序用户信息组件示例代码**

```
<!-- 如果只是展示用户头像昵称，可以使用 <open-data /> 组件 -->
<open-data type="userAvatarUrl"></open-data>
<open-data type="userNickName"></open-data>
<!-- 需要使用 button 来授权登录 -->
<button wx:if="{{canIUse}}" open-type="getUserInfo"
bindgetuserinfo="bindGetUserInfo">授权登录</button>
<view wx:else>请升级宿主客户端版本</view>
```

```
Page({
  data: {
    canIUse: wx.canIUse('button.open-type.getUserInfo')
  },
  onLoad: function() {
    // 查看是否授权
    wx.getSetting({
      success (res){
        if (res.authSetting['scope.userInfo']) {
          // 已经授权，可以直接调用 getUserInfo 获取头像昵称
          wx.getUserInfo({
            success: function(res) {
              console.log(res.userInfo)
            }
          })
        }
      }
    })
  },
  bindGetUserInfo (e) {
    console.log(e.detail.userInfo)
```

```
  }
})
```

userInfo

- **功能说明：**用户信息。
- **参数及说明：**

属性	类型	说明
nickName	string	用户昵称
avatarUrl	string	用户头像图片的 URL。URL 最后一个数值代表正方形头像大小（有 0、46、64、96、132 数值可选，0 代表 640x640 的正方形头像，46 表示 46x46 的正方形头像，剩余数值以此类推。默认132），用户没有头像时该项为空。若用户更换头像，原有头像 URL 将失效
gender	number	用户性别。不再返回，合法值为： • 0：未知 • 1：男性 • 2：女性
country	string	用户所在国家。不再返回
province	string	用户所在省份。不再返回
city	string	用户所在城市。不再返回
language	string	显示 country，province，city 所用的语言。强制返回 “zh_CN”，合法值为： • en：英文 • zh_CN：简体中文 • zh_TW：繁体中文

设置

AuthSetting

- **功能说明：**用户授权设置信息。
- **参数及说明：**

属性	说明
boolean scope.userLocation	是否授权地理位置，对应接口 wx.getLocation 是否授权地理位置，对应接口 wx.chooseLocation
boolean scope.writePhotosAlbum	是否授权保存到相册 wx.saveImageToPhotosAlbum
boolean scope.camera	是否授权摄像头，对应 <camera /> 组件
boolean scope.addFriend	允许被添加好友，主动调用 wx.authorize 接口进行授权

getSetting

> 该 API 使用方法为 wx.getSetting(Object object)

- **功能说明：**获取用户的当前设置。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
withSubscriptions	Boolean	false	否	是否同时获取用户订阅消息的订阅状态，默认不获取
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

> ⚠ **注意：**
> withSubscriptions 只返回用户勾选过订阅面板中的“总是保持以上选择，不再询问”的订阅消息。

- **object.success 回调函数参数：**Object res。

属性	类型	说明
authSetting	AuthSetting	用户授权结果

- **示例代码：**

```
wx.getSetting({
  success(res) {
    console.log(res.authSetting)
    // res.authSetting = {
    //   "scope.userInfo": true,
    //   "scope.userLocation": true
    // }
  }
})
```

openSetting

> 该 API 使用方法为 wx.openSetting(Object object)

- **功能说明：**调起客户端小程序设置界面，返回用户设置的操作结果，用户发生单击行为后，可以跳转打开设置页，管理授权信息。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
authSetting	AuthSetting	用户授权结果

- **示例代码：**

```
wx.openSetting({
  success(res) {
    console.log(res.authSetting)
    // res.authSetting = {
```

```
    //   "scope.userInfo": true,
    //   "scope.userLocation": true
    // }
  }
})
```

```
wx.openSetting({
  success (res) {
    console.log(res.authSetting)
    // res.authSetting = {
    //   "scope.userInfo": true,
    //   "scope.userLocation": true
    // }
  }
})
```

生物认证

checkIsSoterEnrolledInDevice

> 该 API 使用方法为 wx.checkIsSoterEnrolledInDevice(Object object)

- **功能说明：**校验设备内是否录入生物信息。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
isEnrolled	boolean	是否已录入信息
errMsg	string	错误信息

- **示例代码：**

```
wx.checkIsSoterEnrolledInDevice({
  success(res) {
    console.log(res.isEnrolled)
  }
})
```

checkIsSupportSoterAuthentication

> 该 API 使用方法为 wx.checkIsSupportSoterAuthentication(Object object)

- **功能说明：**校验本机是否支持生物认证，支持时回调 success，不支持时回调 fail。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.checkIsSupportSoterAuthentication({
  success() {
    // 支持生物认证
  }
})
```

startSoterAuthentication

> 该 API 使用方法为 wx.startSoterAuthentication(Object object)

- **功能说明：**开始生物认证。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

authContent	string	''	否	验证描述，即识别过程中显示在界面上的对话框提示内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.startSoterAuthentication({
  authContent: '生物认证解锁',
  success() {
    // 认证成功
  }
})
```

授权

authorize

> 该 API 使用方法为 wx.authorize(Object object)

- **功能说明：**提前向用户发起授权请求。调用后会立刻弹窗询问用户是否同意授权小程序使用某项功能或获取用户的某些数据，但不会实际调用对应接口。如果用户之前已经同意授权，则不会出现弹窗，直接返回成功。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
scope	string	–	是	需要获取权限的 scope
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
// 可以通过 wx.getSetting 先查询一下用户是否授权了 "scope.record" 这个 scope
wx.getSetting({
  success(res) {
    if (!res.authSetting['scope.record']) {
      wx.authorize({
        scope: 'scope.record',
        success() {
          // 用户已经同意小程序使用录音功能，后续调用 wx.startRecord 接口不会弹窗询问
          wx.startRecord()
        }
      })
    }
  }
})
```

位置

最近更新时间：2023-10-20 15:19:19

getLocation

该 API 使用方法为 wx.getLocation(Object object)

说明：
调用前需要“用户授权” scope.userLocation。

- **功能说明：**获取当前的地理位置、速度。当用户离开小程序后，此接口无法调用。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
type	string	wgs84	否	wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标
altitude	string	false	否	传入 true 会返回高度信息，由于获取高度需要较高精确度，会减慢接口返回速度
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
latitude	number	纬度，范围为 -90~90，负数表示南纬
longitude	number	经度，范围为 -180~180，负数表示西经
speed	number	速度，单位 m/s

accuracy	number	位置的精确度
altitude	number	高度，单位 m
verticalAccuracy	number	垂直精度，单位 m（Android 无法获取，返回 0）
horizontal Accuracy	number	水平精度，单位 m

- **示例代码：**

```
wx.getLocation({
  type: 'gcj02',
  success(res) {
    const latitude = res.latitude
    const longitude = res.longitude
    const speed = res.speed
    const accuracy = res.accuracy
  }
})
```

> **说明：**
> - 工具中定位模拟使用 IP 定位，可能会有一定误差。且工具目前仅支持 gcj02 坐标。
> - 使用第三方服务进行逆地址解析时，请确认第三方服务默认的坐标系，正确进行坐标转换。

choosePoi

> 该 API 使用方法为 wx.choosePoi(Object object)

- **功能描述：**打开 POI 列表选择位置，支持模糊定位（精确到市）和精确定位混选。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数

complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
type	number	选择城市时，值为1，选择精确位置时，值为2
city	number	城市名称
name	string	位置名称
address	string	详细地址
latitude	number	纬度，浮点数，范围为-90~90，负数表示南纬。使用 gcj02 国测局坐标系（即将废弃）
longitude	number	经度，浮点数，范围为-180~180，负数表示西经。使用 gcj02 国测局坐标系（即将废弃）

- **示例：**

chooseLocation

> 该 API 使用方法为 wx.chooseLocation(Object object)

- **功能描述：**打开地图选择位置。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
type	string	tencent	否	腾讯地图 tencent，谷歌地图 google，仅 IDE 支持

latitude	number	–	否	目标地纬度
longitude	number	–	否	目标地经度
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
name	string	位置名称
address	string	详细地址
latitude	number	纬度，浮点数，范围为-90~90，负数表示南纬。使用 gcj02 国测局坐标系
longitude	number	经度，浮点数，范围为-180~180，负数表示西经。使用 gcj02 国测局坐标系

- **示例：**

stopLocationUpdate

> 该 API 使用方法为 wx.stopLocationUpdate(Object object)

- **功能说明：**关闭监听实时位置变化，前后台都停止消息接收。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

startLocationUpdateBackground

> 该 API 使用方法为 wx.startLocationUpdateBackground(Object object)

- **功能说明：**开启小程序进入前后台时均接收位置消息，需引导用户开启'授权'。授权以后，小程序在运行中或进入后台均可接受位置消息变化。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
type	string	gcj02	否	wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

startLocationUpdate

> 该 API 使用方法为 wx.startLocationUpdate(Object object)

- **功能说明：**开启小程序进入前台时接收位置消息。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
type	string	坐标，gcj02	否	wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标
succes	func	–	否	接口调用成功的回调函数

s	tion			
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

openLocation

> 该 API 使用方法为 wx.openLocation(Object object)

- **功能说明：**使用内置地图查看位置。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
type	string	tencent	否	腾讯地图 tencent，谷歌地图 google，仅 IDE 支持
latitude	number	–	是	纬度，范围为-90~90，负数表示南纬。使用 gcj02 国测局坐标系
longitude	number	–	是	经度，范围为-180~180，负数表示西经。使用 gcj02 国测局坐标系
scale	number	18	否	缩放比例，范围5~18
name	string	–	否	位置名
address	string	–	否	地址的详细说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.getLocation({
 type: 'gcj02', //返回可以用于 wx.openLocation 的经纬度
 success (res) {
  const latitude = res.latitude
  const longitude = res.longitude
  wx.openLocation({
   latitude,
   longitude,
   scale: 18
  })
 }
})
```

onLocationChangeError

该 API 使用方法为 wx.onLocationChangeError(function listener)

- **功能说明：**取消监听持续定位接口返回失败时触发。
- **参数及说明：**function callback 的回调函数。

onLocationChange

该 API 使用方法为 wx.onLocationChange(function listener)

- **功能说明：**监听实时地理位置变化事件，需结合 wx.startLocationUpdateBackground 、wx.startLocationUpdate 使用。
- **参数及说明：**function callback。
- **实时地理位置变化事件的回调函数参数：**Object res。

属性	类型	说明
latitude	number	纬度，范围为 -90~90，负数表示南纬。使用 gcj02 国测局坐标系
longitude	number	经度，范围为 -180~180，负数表示西经。使用 gcj02 国测局坐标系
speed	number	速度，单位 m/s
accuracy	number	位置的精确度

altitude	number	高度，单位m
verticalAccuracy	number	垂直精度，单位m（Android 无法获取，返回0）
horizontalAccuracy	number	水平精度，单位m

- **示例代码**

```
const _locationChangeFn = function(res) {
 console.log('location change', res)
}
wx.onLocationChange(_locationChangeFn)
wx.offLocationChange(_locationChangeFn)
```

offLocationChangeError

> 该 API 使用方法为 wx.offLocationChangeError(function listener)

- **功能说明：**取消监听持续定位接口返回失败时触发。
- **参数及说明：**function callback 的回调函数。

offLocationChange

> 该 API 使用方法为 wx.offLocationChange(function listener)

- **功能说明：**取消监听实时地理位置变化事件。
- **参数及说明：**function callback 实时地理位置变化事件的回调函数。

getFuzzyLocation

> 该 API 使用方法为 wx.getFuzzyLocation(Object object)

- **功能说明：**获取当前的模糊地理位置。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
type	string	wgs84	否	wgs84 返回 gps 坐标，gcj02 返回可用于 wx.openLocation 的坐标

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
latitude	number	纬度，范围为 -90~90，负数表示南纬
longitude	number	经度，范围为 -180~180，负数表示西经

- **示例代码**

```
wx.getFuzzyLocation({
 type: 'wgs84',
 success (res) {
  const latitude = res.latitude
  const longitude = res.longitude
 }
})
```

设备

蓝牙-通用

最近更新时间：2023-10-20 15:19:19

stopBluetoothDevicesDiscovery

> 该 API 使用方法为 wx.stopBluetoothDevicesDiscovery(Object object)

- **功能说明：**停止搜寻附近的蓝牙外围设备。若已经找到需要的蓝牙设备并不需要继续搜索时，建议调用该接口停止蓝牙搜索。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征

10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码**

```
wx.stopBluetoothDevicesDiscovery({
  success (res) {
    console.log(res)
  }
})
```

startBluetoothDevicesDiscovery

该 API 使用方法为 wx.startBluetoothDevicesDiscovery(Object object)

> ⚠ **注意：**
> 考虑到蓝牙功能可以间接进行定位，Android 6.0 及以上版本，无定位权限或定位开关未打开时，无法进行设备搜索。这种情况下，Android 8.0.16 前，接口调用成功但无法扫描设备；Android 8.0.16 及以上版本，会返回错误。

- **功能说明：**开始搜寻附近的蓝牙外围设备。此操作比较耗费系统资源，请在搜索到需要的设备后及时调用 wx.stopBluetoothDevicesDiscovery 停止搜索。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
services	Array.<string>	_	否	要搜索的蓝牙设备主服务的 UUID 列表（支持 16/32/128 位 UUID）。某些蓝牙设备会广播自己的主 service 的 UUID。如果设置此参数，则只搜索广播包有对应 UUID 的主服务的蓝牙设备。建议通过该参数过滤掉周边不需要处理的其他蓝牙设备

allowDuplicatesKey	boolean	false	否	是否允许重复上报同一设备。如果允许重复上报
interval	number	0	否	上报设备的间隔，单位 ms。0表示找到新设备立即上报，其他数值根据传入的间隔上报
powerLevel	string	medium	否	扫描模式，越高扫描越快，也越耗电。仅 Android 宿主客户端7.0.12及以上支持，合法值为： • low：低 • medium：中 • high：高
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常

10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码**

```
// 以宿主客户端硬件平台的蓝牙智能灯为例，主服务的 UUID 是 FEE7。传入这个参数，只搜索主服务 UUID 为 FEE7 的设备
wx.startBluetoothDevicesDiscovery({
  services: ['FEE7'],
  success (res) {
    console.log(res)
  }
})
```

openBluetoothAdapter

> 该 API 使用方法为 wx.openBluetoothAdapter(Object object)

说明：

- 其他蓝牙相关 API 必须在 wx.openBluetoothAdapter 调用之后使用。否则 API 会返回错误（errCode=10000）。
- 在用户蓝牙开关未开启或者手机不支持蓝牙功能的情况下，调用 wx.openBluetoothAdapter 会返回错误（errCode=10001），表示手机蓝牙功能不可用。

- **功能说明：**初始化蓝牙模块。iOS 上开启主机/从机（外围设备）模式时需分别调用一次，并指定对应的 `mode` 。
- **参数及说明：**Object object。

属性	类型	合法值及说明	默认值	必填	说明
mode	string	central：主机模式 peripheral：从机（外围设备）模式	central	否	蓝牙模式，可作为主/从设备，仅 iOS 需要
success	function	-	-	否	接口调用成功的回调函数
fail	func	-	-	否	接口调用失败的回调函数

	tion				
complete	function	–	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误返回值：**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **object.fail 回调函数返回的 state 参数（仅 iOS）**

状态码	说明
0	未知
1	重置中
2	不支持
3	未授权

4	未开启

- **示例代码：**

```
wx.openBluetoothAdapter({
  success (res) {
    console.log(res)
  }
})
```

getConnectedBluetoothDevices

> 该 API 使用方法为 wx.getConnectedBluetoothDevices(Object object)

- **功能说明：**根据主服务 UUID 获取已连接的蓝牙设备。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
services	Array.	–	是	蓝牙设备主服务的 UUID 列表（支持 16/32/128 位 UUID）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object res。

属性	类型	说明
devices	Array.	搜索到的设备列表

结构属性	类型	说明
name	string	蓝牙设备名称，某些设备可能没有
deviceId	string	用于区分设备的 id

- **错误返回值**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于 4.3 不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码**

```
wx.getConnectedBluetoothDevices({
  services: ['FEE7'],
  success (res) {
    console.log(res)
  }
})
```

getBluetoothDevices

> 该 API 使用方法为 wx.getBluetoothDevices(Object object)

- **功能说明：**获取在蓝牙模块生效期间所有搜索到的蓝牙设备。包括已经和本机处于连接状态的设备。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object res。

属性	类型	说明
devices	Array.<Object>	UUID 对应的已连接设备列表

结构属性	类型	说明
name	string	蓝牙设备名称，某些设备可能没有
deviceId	string	蓝牙设备 id
RSSI	number	当前蓝牙设备的信号强度，单位：dBm
advertisData	ArrayBuffer	当前蓝牙设备的广播数据段中的 ManufacturerData 数据段
advertisServiceUUIDs	Array.<string>	当前蓝牙设备的广播数据段中的 ServiceUUIDs 数据段
localName	string	当前蓝牙设备的广播数据段中的 LocalName 数据段
serviceData	Object	当前蓝牙设备的广播数据段中的 ServiceData 数据段
connectable	boolean	当前蓝牙设备是否可连接（ Android 8.0以下不支持返回该值 ）

getBluetoothAdapterState

该 API 使用方法为 wx.getBluetoothAdapterState(Object object)

- **功能说明：**获取本机蓝牙适配器状态。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object res。

属性	类型	说明
discovering	boolean	是否正在搜索设备
available	boolean	蓝牙适配器是否可用

- **错误返回值**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开

10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
wx.getBluetoothAdapterState({
  success (res) {
    console.log(res)
  }
})
```

closeBluetoothAdapter

> 该 API 使用方法为 wx.closeBluetoothAdapter(Object object)

- **功能说明：**关闭蓝牙模块。调用该方法将断开所有已建立的连接并释放系统资源。建议在使用蓝牙流程后，与 wx.openBluetoothAdapter 成对调用。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误返回值**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接

10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码**

```
wx.closeBluetoothAdapter({
  success (res) {
    console.log(res)
  }
})
```

onBluetoothDeviceFound

> 该 API 使用方法为 wx.onBluetoothDeviceFound(function listener)

⚠ 注意：

- 若在 wx.onBluetoothDeviceFound 回调了某个设备，则此设备会添加到 wx.getBluetoothDevices 接口获取到的数组中。
- 蓝牙设备在被搜索到时，系统返回的 name 字段一般为广播包中的 LocalName 字段中的设备名称，而如果与蓝牙设备建立连接，系统返回的 name 字段会改为从蓝牙设备上获取到的 GattName。若需要动态改变设备名称并展示，建议使用 localName 字段。
- Android 下部分机型需要有位置权限才能搜索到设备，需留意是否开启了位置权限。

- **功能说明：**监听搜索到新设备的事件。
- **参数及说明：**object res 参数，function listener，搜索到新设备的事件的监听函数。

属性	类型	说明
devices	Array.<Object>	新搜索到的设备列表

  - **devices 结构属性**

结构属性	类型	说明
name	string	蓝牙设备名称，某些设备可能没有
deviceId	string	蓝牙设备 id
RSSI	number	当前蓝牙设备的信号强度，单位 dBm
advertisData	ArrayBuffer	当前蓝牙设备的广播数据段中的 ManufacturerData 数据段
advertisServiceUUIDs	Array.<string>	当前蓝牙设备的广播数据段中的 ServiceUUIDs 数据段
localName	string	当前蓝牙设备的广播数据段中的 LocalName 数据段
serviceData	Object	当前蓝牙设备的广播数据段中的 ServiceData 数据段
connectable	boolean	当前蓝牙设备是否可连接（ Android 8.0 以下不支持返回该值 ）

- **示例代码**

```
// ArrayBuffer转16进度字符串示例
function ab2hex(buffer) {
  var hexArr = Array.prototype.map.call(
    new Uint8Array(buffer),
    function(bit) {
      return ('00' + bit.toString(16)).slice(-2)
    }
  )
  return hexArr.join('');
}
wx.onBluetoothDeviceFound(function(res) {
  var devices = res.devices;
  console.log('new device list has founded')
```

```
  console.dir(devices)
  console.log(ab2hex(devices[0].advertisData))
})
```

offBluetoothDeviceFound

> 该 API 使用方法为 wx.offBluetoothDeviceFound()

- **功能说明：**移除搜索到新设备的事件的全部监听函数。

onBluetoothAdapterStateChange

> 该 API 使用方法为 wx.onBluetoothAdapterStateChange(function listener)

- **功能说明：**监听蓝牙适配器状态变化事件。
- **参数及说明：**Object res 参数，function listener，蓝牙适配器状态变化事件的监听函数。

属性	类型	说明
available	boolean	蓝牙适配器是否可用
discovering	boolean	蓝牙适配器是否处于搜索状态

- **示例代码**

```
wx.onBluetoothAdapterStateChange(function (res) {
  console.log('adapterState changed, now is', res)
})
```

offBluetoothAdapterStateChange

> 该 API 使用方法为 wx.offBluetoothAdapterStateChange()

- **功能说明：**移除蓝牙适配器状态变化事件的全部监听函数。

makeBluetoothPair

> 该 API 使用方法为 wx.makeBluetoothPair(Object object)

- **功能说明：**蓝牙配对接口，仅 Android 支持。通常情况下（需要指定 `pin` 码或者密码时）系统会接管配对流程。该接口只应当在开发者不想让用户手动输入 `pin` 码且真机验证确认可以正常生效情况下用。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
deviceId	string	-	是	蓝牙设备 id
pin	string	-	是	pin 码，Base64 格式。
timeout	number	20000	否	超时时间，单位：ms
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

isBluetoothDevicePaired

> 该 API 使用方法为 wx.isBluetoothDevicePaired(Object object)

- **功能说明：**查询蓝牙设备是否配对，仅 Android 支持。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	-	是	蓝牙设备 id
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

蓝牙-低功耗中心设备

最近更新时间：2023-10-20 15:19:20

writeBLECharacteristicValue

该 API 使用方法为 wx.writeBLECharacteristicValue(Object object)

注意：

- 并行调用多次会存在写失败的可能性。
- 小程序不会对写入数据包大小做限制，但系统与蓝牙设备会限制蓝牙4.0单次传输的数据大小，超过最大字节数后会发生写入错误，建议每次写入不超过20字节。
- 若单次写入数据过长，iOS 上存在系统不会有任何回调的情况（包括错误回调）。
- Android 平台上，在调用 wx.notifyBLECharacteristicValueChange 成功后立即调用本接口，在部分机型上会发生 10008 系统错误。

- **功能说明：**向蓝牙低功耗设备特征值中写入二进制数据。**必须设备的特征支持 write 才可以成功调用。**
- **参数及说明：**Object object。

属性	类型	必填	说明
deviceId	string	是	蓝牙设备 id
serviceId	string	是	蓝牙特征对应服务的 UUID
characteristicId	string	是	蓝牙特征的 UUID
value	ArrayBuffer	是	蓝牙设备特征对应的二进制值
writeType	string	否	蓝牙特征值的写模式设置，有两种模式，iOS 优先 write，Android 优先 writeNoResponse。（基础库2.22.0开始支持），合法值为： • write：强制回复写，不支持时报错 • writeNoResponse：强制无回复写，不支持时报错
success	function	否	接口调用成功的回调函数
fail	functio	否	接口调用成功的回调函数

	n		
complete	function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
// 向蓝牙设备发送一个0x00的16进制数据
let buffer = new ArrayBuffer(1)
let dataView = new DataView(buffer)
dataView.setUint8(0, 0)

wx.writeBLECharacteristicValue({
  // 这里的 deviceId 需要在 getBluetoothDevices 或 onBluetoothDeviceFound 接口中获取
  deviceId,
```

```
  // 这里的 serviceId 需要在 getBLEDeviceServices 接口中获取
  serviceId,
  // 这里的 characteristicId 需要在 getBLEDeviceCharacteristics 接口中获取
  characteristicId,
  // 这里的value是ArrayBuffer类型
  value: buffer,
  success (res) {
    console.log('writeBLECharacteristicValue success', res.errMsg)
  }
})
```

readBLECharacteristicValue

该 API 使用方法为 wx.readBLECharacteristicValue(Object object)

注意：

- 并行调用多次会存在读失败的可能性。
- 接口读取到的信息需要在 wx.onBLECharacteristicValueChange 方法注册的回调中获取。

- **功能说明：**读取蓝牙低功耗设备特征值的二进制数据。**必须设备的特征支持 read 才可以成功调用** 。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	–	是	蓝牙设备 id
serviceId	string	–	是	蓝牙特征对应服务的 UUID
characteristicId	string	–	是	蓝牙特征的 UUID
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于 4.3 不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
// 必须在这里的回调才能获取
wx.onBLECharacteristicValueChange(function(characteristic) {
  console.log('characteristic value comed:', characteristic)
})

wx.readBLECharacteristicValue({
  // 这里的 deviceId 需要已经通过 createBLEConnection 与对应设备建立链接
  deviceId,
  // 这里的 serviceId 需要在 getBLEDeviceServices 接口中获取
  serviceId,
  // 这里的 characteristicId 需要在 getBLEDeviceCharacteristics 接口中获取
  characteristicId,
  success (res) {
```

```
    console.log('readBLECharacteristicValue:', res.errCode)
  }
})
```

setBLEMTU

> 该 API 使用方法为 wx.setBLEMTU(Object object)

- **功能说明：**协商设置蓝牙低功耗的最大传输单元 (Maximum Transmission Unit，MTU)。需在 wx.createBLEConnection 调用成功后调用。仅 Android 系统5.1以上版本有效，iOS 因系统限制不支持。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	–	是	蓝牙设备 id
mtu	number	–	是	最大传输单元。设置范围为 (22,512) 区间内，单位 bytes
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
mtu	number	最终协商的 MTU 值，与传入参数一致。Android 客户端8.0.9开始支持

getBLEMTU

> 该 API 使用方法为 wx.getBLEMTU(Object object)

> ⚠ **注意：**
> - 小程序中 MTU 为 ATT_MTU，包含 Op-Code 和 Attribute Handle 的长度，实际可以传输的数

据长度为 ATT_MTU - 3。
- iOS 系统中 MTU 为固定值；Android 系统中，MTU 会在系统协商成功之后发生改变，建议使用 wx.onBLEMTUChange 监听。

- **功能说明：**获取蓝牙低功耗的最大传输单元。需在 wx.createBLEConnection 调用成功后调用。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	-	是	蓝牙设备 id
writeType	string	write	否	写模式（iOS 特有参数），合法值为： • write：有回复写 • writeNoResponse：无回复写
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用成功的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
mtu	number	最大传输单元

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备

10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
wx.getBLEMTU({
  deviceId: '',
  writeType: 'write',
  success (res) {
    console.log(res)
  }
})
```

onBLEMTUChange

> 该 API 使用方法为 wx.onBLEMTUChange(function listener)

- **功能说明：**监听蓝牙低功耗的最大传输单元变化事件（仅 Android 触发）。
- **参数及说明：**Object res 参数，function listener，蓝牙低功耗的最大传输单元变化事件的监听函数。

属性	类型	说明
deviceId	string	蓝牙设备 id
mtu	number	最大传输单元

- **示例代码：**

```
wx.onBLEMTUChange(function (res) {
  console.log('bluetooth mtu is', res.mtu)
})
```

offBLEMTUChange

> 该 API 使用方法为 wx.offBLEMTUChange(function listener)

- **功能说明：**移除蓝牙低功耗的最大传输单元变化事件的监听函数。
- **参数及说明：**function listener，onBLEMTUChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onBLEMTUChange(listener)
wx.offBLEMTUChange(listener) // 需传入与监听时同一个的函数对象
```

onBLEConnectionStateChange

> 该 API 使用方法为 wx.onBLEConnectionStateChange(function listener)

- **功能说明：**监听蓝牙低功耗连接状态改变事件。包括开发者主动连接或断开连接，设备丢失，连接异常断开等。
- **参数及说明：**Object res 参数，function listener，蓝牙低功耗连接状态改变事件的监听函数。

属性	类型	说明
deviceId	string	蓝牙设备 id
connected	boolean	是否处于已连接状态

- **示例代码：**

```
wx.onBLEConnectionStateChange(function(res) {
  // 该方法回调中可以用于处理连接意外断开等异常情况
  console.log(`device ${res.deviceId} state has changed, connected:
${res.connected}`)
})
```

offBLEConnectionStateChange

> 该 API 使用方法为 wx.offBLEConnectionStateChange(function listener)

- **功能说明：**移除蓝牙低功耗连接状态改变事件的监听函数。
- **参数及说明：**function listener，onBLEConnectionStateChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onBLEConnectionStateChange(listener)
wx.offBLEConnectionStateChange(listener) // 需传入与监听时同一个的函数对象
```

onBLECharacteristicValueChange

> 该 API 使用方法为 wx.onBLECharacteristicValueChange(function listener)

- **功能说明：**监听蓝牙低功耗设备的特征值变化事件。必须先调用 wx.notifyBLECharacteristicValueChange 接口才能接收到设备推送的 notification。
- **参数及说明：**Object res 参数，function listener，蓝牙低功耗设备的特征值变化事件的监听函数。

属性	类型	说明
deviceId	string	蓝牙设备 id
serviceId	string	蓝牙特征对应服务的 UUID
characteristicId	string	蓝牙特征的 UUID
value	ArrayBuffer	特征最新的值

- **示例代码：**

```
// ArrayBuffer转16进制字符串示例
function ab2hex(buffer) {
  let hexArr = Array.prototype.map.call(
    new Uint8Array(buffer),
    function(bit) {
      return ('00' + bit.toString(16)).slice(-2)
    }
  )
  return hexArr.join('');
}
wx.onBLECharacteristicValueChange(function(res) {
```

```
  console.log(`characteristic ${res.characteristicId} has changed, now is
${res.value}`)
  console.log(ab2hex(res.value))
})
```

offBLECharacteristicValueChange

> 该 API 使用方法为 wx.offBLECharacteristicValueChange()

功能说明：移除蓝牙低功耗设备的特征值变化事件的全部监听函数。

notifyBLECharacteristicValueChange

> 该 API 使用方法为 wx.notifyBLECharacteristicValueChange(Object object)

> ⚠ **注意：**
>
> - 订阅操作成功后需要设备主动更新特征的 value，才会触发 wx.onBLECharacteristicValueChange 回调。
> - Android 平台上，在本接口调用成功后立即调用 wx.writeBLECharacteristicValue 接口，在部分机型上会发生 10008 系统错误。
> - 必须设备的特征支持 notify 或者 indicate 才可以成功调用。另外，必须先启用 wx.notifyBLECharacteristicValueChange 才能监听到设备 characteristicValueChange 事件。

- **功能说明：**启用蓝牙低功耗设备特征值变化时的 notify 功能，订阅特征。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	-	是	蓝牙设备 id
serviceId	string	-	是	蓝牙特征对应服务的 UUID
characteristicId	string	-	是	蓝牙特征的 UUID
state	boolean	-	是	是否启用 notify
type	string	indication	否	设置特征订阅类型，有效值有 notification 和 indication

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
// 向蓝牙设备发送一个0x00的16进制数据
let buffer = new ArrayBuffer(1)
let dataView = new DataView(buffer)
dataView.setUint8(0, 0)
```

```
wx.writeBLECharacteristicValue({
  // 这里的 deviceId 需要在 getBluetoothDevices 或 onBluetoothDeviceFound 接口中获取
  deviceId,
  // 这里的 serviceId 需要在 getBLEDeviceServices 接口中获取
  serviceId,
  // 这里的 characteristicId 需要在 getBLEDeviceCharacteristics 接口中获取
  characteristicId,
  // 这里的value是ArrayBuffer类型
  value: buffer,
  success (res) {
    console.log('writeBLECharacteristicValue success', res.errMsg)
  }
})
```

getBLEDeviceServices

> 该 API 使用方法为 wx.getBLEDeviceServices(Object object)

- **功能说明：**获取蓝牙低功耗设备所有服务 (service)。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	–	是	蓝牙设备 id。需要已经通过 wx.createBLEConnection 建立连接
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
services	Array.<Object>	设备服务列表

- **service 结构属性**

属性	类型	说明
uuid	string	蓝牙设备服务的 UUID
isPrimary	boolean	该服务是否为主服务

- **示例代码：**

```
wx.getBLEDeviceServices({
  // 这里的 deviceId 需要已经通过 wx.createBLEConnection 与对应设备建立连接
  deviceId,
  success (res) {
    console.log('device services:', res.services)
  }
})
```

getBLEDeviceRSSI

> 该 API 使用方法为 wx.getBLEDeviceRSSI(Object object)

- **功能说明：**获取蓝牙低功耗设备的信号强度 (Received Signal Strength Indication，RSSI)。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	–	是	蓝牙设备 id
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
RSSI	Number	信号强度，单位：dBm

getBLEDeviceCharacteristics

该 API 使用方法为 wx.getBLEDeviceCharacteristics(Object object)

- **功能说明：**获取蓝牙低功耗设备某个服务中所有特征 (characteristic)。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	–	是	蓝牙设备 id。需要已经通过 wx.createBLEConnection 建立连接
serviceId	string	–	是	蓝牙服务 UUID。需要先调用 wx.getBLEDeviceServices 获取
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
characteristics	Array.<Object>	设备特征列表

  - **characteristics 结构属性**

结构属性	类型	说明
uuid	string	蓝牙设备特征的 UUID
properties	Object	该特征支持的操作类型

    - **properties 结构属性**

属性	类型	说明
read	boolean	该特征是否支持 read 操作
write	boolean	该特征是否支持 write 操作
notify	boolean	该特征是否支持 notify 操作

indicate	boolean	该特征是否支持 indicate 操作
writeNoResponse	boolean	该特征是否支持 writeNoResponse 操作
writeDefault	boolean	该特征是否支持 writeDefault 操作

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
wx.getBLEDeviceCharacteristics({
  // 这里的 deviceId 需要已经通过 wx.createBLEConnection 与对应设备建立链接
  deviceId,
  // 这里的 serviceId 需要在 wx.getBLEDeviceServices 接口中获取
  serviceId,
  success (res) {
    console.log('device getBLEDeviceCharacteristics:', res.characteristics)
```

```
  }
})
```

createBLEConnection

该 API 使用方法为 wx.createBLEConnection(Object object)

注意：

- 请保证尽量成对的调用 wx.createBLEConnection 和 wx.closeBLEConnection 接口。Android 如果重复调用 wx.createBLEConnection 创建连接，有可能导致系统持有同一设备多个连接的实例，导致调用 closeBLEConnection 的时候并不能真正的断开与设备的连接。
- 蓝牙连接随时可能断开，建议监听 wx.onBLEConnectionStateChange 回调事件，当蓝牙设备断开时按需执行重连操作
- 若对未连接的设备或已断开连接的设备调用数据读写操作的接口，会返回 10006 错误，建议进行重连操作。

- **功能说明：**连接蓝牙低功耗设备。若小程序在之前已有搜索过某个蓝牙设备，并成功建立连接，可直接传入之前搜索获取的 deviceId 直接尝试连接该设备，无需再次进行搜索操作。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	–	是	蓝牙设备 id
timeout	number	–	否	超时时间，单位 ms，不填表示不会超时
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常

-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于4.3不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
wx.createBLEConnection({
  deviceId,
  success (res) {
    console.log(res)
  }
})
```

closeBLEConnection

> 该 API 使用方法为 wx.closeBLEConnection(Object object)

- **功能说明：**断开与蓝牙低功耗设备的连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
deviceId	string	-	是	蓝牙设备 id

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
-1	already connect	已连接
10000	not init	未初始化蓝牙适配器
10001	not available	当前蓝牙适配器不可用
10002	no device	没有找到指定设备
10003	connection fail	连接失败
10004	no service	没有找到指定服务
10005	no characteristic	没有找到指定特征
10006	no connection	当前连接已断开
10007	property not support	当前特征不支持此操作
10008	system error	其余所有系统上报的异常
10009	system not support	Android 系统特有，系统版本低于 4.3 不支持 BLE
10012	operate time out	连接超时
10013	invalid_data	连接 deviceId 为空或者是格式不正确

- **示例代码：**

```
wx.closeBLEConnection({
  deviceId,
  success (res) {
    console.log(res)
  }
```

```
})
```

蓝牙-低功耗外围设备

最近更新时间：2023-10-20 15:19:20

onBLEPeripheralConnectionStateChanged

> 该 API 使用方法为 wx.onBLEPeripheralConnectionStateChanged(function listener)。

- **功能说明：**监听当前外围设备被连接或断开连接事件。
- **参数及说明：**Object res 参数，function listener，当前外围设备被连接或断开连接事件的监听函数。

属性	类型	说明
deviceId	String	连接状态变化的设备 id
serverId	String	server 的 UUID
connected	Boolean	连接目前状态

offBLEPeripheralConnectionStateChanged

> 该 API 使用方法为 wx.offBLEPeripheralConnectionStateChanged(function listener)

- **功能说明：**移除当前外围设备被连接或断开连接事件的监听函数。
- **参数及说明：**function listener，onBLEPeripheralConnectionStateChanged 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

wx.onBLEPeripheralConnectionStateChanged(listener)
wx.offBLEPeripheralConnectionStateChanged(listener) // 需传入与监听时同一个的函数对象
```

createBLEPeripheralServer

> 该 API 使用方法为 wx.createBLEPeripheralServer(Object object)

- **功能说明：**建立本地作为蓝牙低功耗外围设备的服务端，可创建多个。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：** Object res。

属性	类型	说明
server	BLEPeripheralServer	外围设备的服务端

BLEPeripheralServer

> **说明：**
> 外围设备的服务端。

.addService

该 方法 使用方式为 BLEPeripheralServer.addService(Object object)

- **功能说明：** 添加服务。
- **参数及说明：** Object object。

属性	类型	必填	说明
service	Object	是	描述service的Object
success	function	否	接口调用成功的回调函数
fail	function	否	接口调用失败的回调函数
complete	function	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **service 结构属性**

结构属性	类型	必填	说明

uuid	string	是	蓝牙服务的 UUID
characteristics	Array.<Object>	是	characteristics 列表

characteristics 结构属性

结构属性	类型	必填	说明
uuid	string	是	characteristic 的 UUID
properties	Object	否	特征支持的操作
permission	Object	否	特征权限
value	Array.<Object>	否	特征对应的二进制值
descriptors	Array.<Object>	否	描述符数据

- **properties 结构属性**

结构属性	类型	默认值	必填	说明
write	boolean	false	否	写
writeNoResponse	boolean	false	否	无回复写
read	boolean	false	否	读
notify	boolean	false	否	订阅
indicate	boolean	false	否	回包

- **permission 结构属性**

结构属性	类型	默认值	必填	说明

readable	boolean	false	否	可读
writeable	boolean	false	否	可写
readEncryptionRequired	boolean	false	否	加密读请求
writeEncryptionRequired	boolean	false	否	加密写请求

- **descriptors 结构属性**

结构属性	类型	必填	说明
uuid	string	是	Descriptors 的 UUID
permission	Object	否	描述符的权限
value	Array Buffer	否	描述符数据

permission 结构属性

结构属性	类型	默认值	必填	说明
write	boolean	false	否	写
read	boolean	false	否	读

.removeService

> 该 方法 使用方式为 BLEPeripheralServer.removeService(Object object)

- **功能说明：**移除服务。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
serviceId	String	-	是	service 的 UUID

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

.startAdvertising

该 方法 使用方式为 BLEPeripheralServer.startAdvertising(Object Object)

- **功能说明：**开始广播本地创建的外围设备。
- **参数及说明：**Object Object。

属性	类型	默认值	必填	说明
advertiseRequest	object	–	是	广播自定义参数
powerLevel	string	medium	否	广播功率，合法值为 • low：功率低 • medium：功率适中 • high：功率高
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

advertiseRequest 结构属性

结构属性	类型	默认值	必填	说明
connectable	boolean	ture	否	当前设备是否可连接
deviceName	string	–	否	广播中 deviceName 字段，默认为空

serviceUuids	Array.<string>	-	否	要广播的服务 UUID 列表。使用 16/32 位 UUID 时请参考注意事项
manufactureData	Array.<Object>	-	否	广播的制造商信息。仅 Android 支持，iOS 因系统限制无法定制
beacon	object	-	否	以 beacon 设备形式广播的参数

- **manufactureData 结构属性**

结构属性	类型	必填	说明
manufacturerId	String	是	制造商ID，0x 开头的十六进制
manufacturerSpecificData	ArrayBuffer	否	制造商信息

- **beacon 结构属性**

结构属性	类型	必填	说明
uuid	number	是	Beacon 设备广播的 UUID
major	number	是	Beacon 设备的主 ID
minor	number	是	Beacon 设备的次 ID
measurePower	number	是	用于判断距离设备 1 米时 RSSI 大小的参考值

.stopAdvertising

> 该 方法 使用方式为 BLEPeripheralServer.stopAdvertising()

- **功能说明：**停止广播。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
comple	functi	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

te	on			

.writeCharacteristicValue

该 方法 使用方式为 BLEPeripheralServer.writeCharacteristicValue(Object Object)

- **功能说明：**往指定特征写入二进制数据值，并通知已连接的主机。从机的特征值已发生变化，该接口会处理是走回包还是走订阅。
- **参数及说明：**Object Object。

属性	类型	默认值	必填	说明
serviceId	string	-	是	蓝牙特征对应服务的 UUID
characteristicId	string	-	是	蓝牙特征的 UUID
value	ArrayBuffer	-	是	characteristic 对应的二进制值
needNotify	boolean	-	是	是否需要通知主机 value 已更新
callbackId	number	-	否	可选，处理回包时使用
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

.onCharacteristicWriteRequest

该 方法 使用方式为 BLEPeripheralServer.onCharacteristicWriteRequest(function listener)

- **功能说明：**监听已连接的设备请求写当前外围设备的特征值事件。收到该消息后需要立刻调用 writeCharacteristicValue 写回数据，否则主机不会收到响应。
- **参数及说明：**Object res。

属性	类型	说明
serviceId	String	蓝牙特征对应服务的 UUID
characteristicId	String	蓝牙特征的 UUID

callbackId	Number	唯一标识码，调用 writeCharacteristicValue 时使用
value	ArrayBuffer	请求写入特征的二进制数据值

.offCharacteristicWriteRequest

该 方法 使用方式为 BLEPeripheralServer.offCharacteristicWriteRequest(function listener)

- **功能说明：**移除已连接的设备请求写当前外围设备的特征值事件的监听函数，使用方法为 。
- **参数及说明：**function listener，onCharacteristicWriteRequest 传入的监听函数。不传此参数则移除所有监听函数。

.onCharacteristicReadRequest

该 方法 使用方式为 BLEPeripheralServer.onCharacteristicReadRequest(function listener)

- **功能说明：**监听已连接的设备请求读当前外围设备的特征值事件。收到该消息后需要立刻调用 writeCharacteristicValue 写回数据，否则主机不会收到响应。
- **参数及说明：**Object res。

属性	类型	说明
serviceId	String	蓝牙特征对应服务的 UUID
characteristicId	String	蓝牙特征的 UUID
callbackId	Number	唯一标识码，调用 writeCharacteristicValue 时使用

.offCharacteristicReadRequest

该 方法 使用方式为 BLEPeripheralServer.offCharacteristicReadRequest(function listener)

- **功能说明：**移除已连接的设备请求读当前外围设备的特征值事件的监听函数。
- **参数及说明：**function listener，onCharacteristicReadRequest 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

BLEPeripheralServer.onCharacteristicReadRequest(listener)
BLEPeripheralServer.offCharacteristicReadRequest(listener) // 需传入与监听时同一个的函数对象
```

.onCharacteristicSubscribed

该 方法 使用方式为 BLEPeripheralServer.onCharacteristicSubscribed(function listener)

- **功能说明：**监听特征订阅事件，仅 iOS 支持。
- **参数及说明：**Object res 参数，function listener，特征订阅事件的监听函数。

属性	类型	说明
serviceId	String	蓝牙特征对应服务的 UUID
characteristicId	String	蓝牙特征的 UUID

.offCharacteristicSubscribed

该 方法 使用方式为 BLEPeripheralServer.offCharacteristicSubscribed(function listener)

- **功能说明：**移除特征订阅事件的监听函数。
- **参数及说明：**function listener，onCharacteristicSubscribed 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

BLEPeripheralServer.onCharacteristicSubscribed(listener)
BLEPeripheralServer.offCharacteristicSubscribed(listener) // 需传入与监听时同一个的
函数对象
```

.onCharacteristicUnsubscribed

该 方法 使用方式为 BLEPeripheralServer.onCharacteristicUnsubscribed(function listener)

- **功能说明：**监听取消特征订阅事件，仅 iOS 支持。
- **参数及说明：**Object res 参数，function listener，取消特征订阅事件的监听函数。

属性	类型	说明
serviceId	String	蓝牙特征对应服务的 UUID
characteristicId	String	蓝牙特征的 UUID

.offCharacteristicUnsubscribed

该 方法 使用方式为 BLEPeripheralServer.offCharacteristicUnsubscribed(function listener)

- **功能说明：**移除取消特征订阅事件的监听函数。
- **参数及说明：**function listener，onCharacteristicUnsubscribed 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

BLEPeripheralServer.onCharacteristicUnsubscribed(listener)
BLEPeripheralServer.offCharacteristicUnsubscribed(listener) // 需传入与监听时同一个的函数对象
```

蓝牙-信标

最近更新时间：2023-10-20 15:19:20

stopBeaconDiscovery

> 该 API 使用方法为 wx.stopBeaconDiscovery(Object object)

- **功能说明：**停止搜索附近的 Beacon 设备。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
11000	unsupport	系统或设备不支持
11001	bluetooth service unavailable	蓝牙服务不可用
11002	location service unavailable	位置服务不可用
11003	already start	已经开始搜索
11004	not startBeaconDiscovery	还未开始搜索
11005	system error	系统错误
11006	invalid data	参数不正确

startBeaconDiscovery

> 该 API 使用方法为 wx.startBeaconDiscovery(Object object)

- **功能说明：**开始搜索附近的 Beacon 设备。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
uuids	Array.<string>	–	是	Beacon 设备广播的 UUID 列表
ignoreBluetoothAvailable	boolean	false	否	是否校验蓝牙开关，仅在 iOS 下有效。iOS 11 起，控制面板里关掉蓝牙，还是能继续使用 Beacon 服务
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
11000	unsupport	系统或设备不支持
11001	bluetooth service unavailable	蓝牙服务不可用
11002	location service unavailable	位置服务不可用
11003	already start	已经开始搜索
11004	not startBeaconDiscovery	还未开始搜索
11005	system error	系统错误
11006	invalid data	参数不正确

- **示例代码**

```
wx.startBeaconDiscovery({
  success(res) { }
```

```
})
```

onBeaconUpdate

> 该 API 使用方法为 wx.onBeaconUpdate(function listener)

- **功能说明：**监听 Beacon 设备更新事件，仅能注册一个监听。
- **参数及说明：**Object res 参数，function listener，Beacon 设备更新事件的监听函数。

属性	类型	说明
beacons	Array.<BeaconInfo>	当前搜寻到的所有 Beacon 设备列表

- **示例代码**

```
wx.onBeaconUpdate(res => {
  console.log(res.beacons)
})
```

offBeaconUpdate

> 该 API 使用方法为 wx.offBeaconUpdate()

- **功能说明：**移除 Beacon 设备更新事件的全部监听函数。

onBeaconServiceChange

> 该 API 使用方法为 wx.onBeaconServiceChange(function listener)

- **功能说明：**监听 Beacon 服务状态变化事件，仅能注册一个监听。
- **参数及说明：**Object res 参数，function listener，Beacon 服务状态变化事件的监听函数。

属性	类型	说明
available	boolean	服务目前是否可用
discovering	boolean	目前是否处于搜索状态

- **示例代码**

```
wx.onBeaconServiceChange(res => {
  console.log(res.available, res.discovering)
})
```

offBeaconServiceChange

> 该 API 使用方法为 wx.offBeaconServiceChange()

- **功能说明：**移除 Beacon 服务状态变化事件的全部监听函数。

getBeacons

> 该 API 使用方法为 wx.getBeacons(Object object)

- **功能说明：**获取所有已搜索到的 Beacon 设备。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
beacons	Array.<BeaconInfo>	Beacon 设备列表

- **错误码**

错误码	错误信息	说明
0	ok	正常
11000	unsupport	系统或设备不支持
11001	bluetooth service unavailable	蓝牙服务不可用
11002	location service unavailable	位置服务不可用
11003	already start	已经开始搜索
11004	not startBeaconDiscovery	还未开始搜索

11005	system error	系统错误
11006	invalid data	参数不正确

BeaconInfo

- **功能说明：** Beacon 设备。
- **属性及说明**

属性名	类型	说明
uuid	string	Beacon 设备广播的 UUID
major	number	Beacon 设备的主 ID
minor	number	Beacon 设备的次 ID
proximity	number	表示设备距离的枚举值（仅iOS）
accuracy	number	Beacon 设备的距离，单位 m。iOS 上，proximity 为 0 时，accuracy 为 -1
rssi	number	表示设备的信号强度，单位 dBm

  - **proximity 的合法值**

值	说明
0	信号太弱不足以计算距离，或非 iOS 设备
1	十分近
2	比较近
3	远

NFC – getNFCAdapter

最近更新时间：2023-10-20 15:19:20

该 API 使用方法为 NFCAdapter wx.getNFCAdapter()

说明：

支持度：

- Android 版：支持。
- iOS 版：不支持。

- **功能说明：**获取 NFC 实例。
- **参数及说明：**返回 NFCAdapter NFC 实例。
- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败

13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC - NFCAdapter

最近更新时间：2023-10-20 15:19:20

NFCAdapter

- **参数及说明：**Object tech，标签类型枚举。

属性	类型	说明
ndef	String	对应 Ndef 实例，实例支持对 NDEF 格式的 NFC 标签上的 NDEF 数据的读写
nfcA	String	对应 NfcA 实例，实例支持 NFC-A (ISO 14443-3A) 标准的读写
nfcB	String	对应 NfcB 实例，实例支持 NFC-B (ISO 14443-3B) 标准的读写
isoDep	String	对应 IsoDep 实例，实例支持 ISO-DEP (ISO 14443-4) 标准的读写
nfcF	String	对应 NfcF 实例，实例支持 NFC-F (JIS 6319-4) 标准的读写
nfcV	String	对应 NfcV 实例，实例支持 NFC-V (ISO 15693) 标准的读写
mifareClassic	String	对应 MifareClassic 实例，实例支持 MIFARE Classic 标签的读写
mifareUltralight	String	对应 MifareUltralight 实例，实例支持 MIFARE Ultralight 标签的读写

.startDiscovery

> 该 方法 使用方式为 NFCAdapter.startDiscovery()

- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	Function	-	否	接口调用成功的回调函数
fail	Function	-	否	接口调用成功的回调函数
complete	Function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.stopDiscovery

该 方法 使用方式为 NFCAdapter.stopDiscovery()

- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能

13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getNdef

> 该 方法 使用方式为 Ndef NFCAdapter.getNdef()

- **功能说明：**获取 Ndef 实例，实例支持对 NDEF 格式的 NFC 标签上的 NDEF 数据的读写。
- **返回值：**Ndef 。

.getNfcA

> 该 方法 使用方式为 NfcA NFCAdapter.getNfcA()

- **功能说明：**获取 NfcA 实例，实例支持 NFC-A (ISO 14443-3A) 标准的读写。
- **返回值：**NfcA 。

.getNfcB

> 该 方法 使用方式为 NfcB NFCAdapter.getNfcB()

- **功能说明：**获取 NfcB 实例，实例支持 NFC-B (ISO 14443-3B) 标准的读写。
- **返回值：**NfcB 。

.getNfcF

> 该 方法 使用方式为 NfcF NFCAdapter.getNfcF()

- **功能说明：**获取 NfcF 实例，实例支持 NFC-F (JIS 6319-4) 标准的读写。
- **返回值：**NfcF 。

.getNfcV

> 该 方法 使用方式为 NfcV NFCAdapter.getNfcV()

- **功能说明：**获取 NfcV 实例，实例支持 NFC-V (ISO 15693) 标准的读写。
- **返回值：**NfcV 。

.getIsoDep

> 该 方法 使用方式为 IsoDep NFCAdapter.getIsoDep()

- **功能说明：**获取 IsoDep 实例，实例支持 ISO-DEP (ISO 14443-4) 标准的读写。
- **返回值：**IsoDep 。

.getMifareClassic

该 方法 使用方式为 MifareClassic NFCAdapter.getMifareClassic()

- **功能说明：**获取 MifareClassic 实例，实例支持 MIFARE Classic 标签的读写。
- **返回值：**MifareClassic 。

.getMifareUltralight

该 方法 使用方式为 MifareUltralight NFCAdapter.getMifareUltralight()

- **功能说明：**获取 MifareUltralight 实例，实例支持 MIFARE Ultralight 标签的读写。
- **参数说明：**MifareUltralight 。

.onDiscovered

该 方法 使用方式为 NFCAdapter.onDiscovered(function listener)

- **功能说明：**监听 NFC Tag。
- **参数说明：**function listener 的监听函数，参数为 Object res。

属性	类型	说明
id	ArrayBuffer	-
techs	Array	tech 数组，用于匹配 NFC 卡片具体可以使用什么标准（ NfcA 等实例 ）处理
messages	Array	NdefMessage 数组，消息格式为 {id: ArrayBuffer, type: ArrayBuffer, payload: ArrayBuffer}

.offDiscovered

该 方法 使用方式为 NFCAdapter.offDiscovered(function listener)

- **功能说明：**移除 NFC Tag 的监听函数。
- **参数及说明：**function listener，onDiscovered 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

NFCAdapter.onDiscovered(listener)
NFCAdapter.offDiscovered(listener) // 需传入与监听时同一个的函数对象
```

NFC – IsoDep

最近更新时间：2023-10-20 15:19:20

IsoDep 标签。

.connect

该 方法 使用方式为 IsoDep.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接

13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.close

> 该 方法 使用方式为 IsoDep.close()

- **功能说明：**断开连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败

13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeout

> 该 方法 使用方式为 IsoDep.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
timeout	number	-	是	设置超时时间 (ms)
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function		否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明

13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

> 该 方法 使用方式为 IsoDep.isConnected()

- **功能说明：** 检查是否已连接。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持NFC	–
13001	系统NFC开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getMaxTransceiveLength

> 该 方法 使用方式为 IsoDep.getMaxTransceiveLength()

- **功能说明：**获取最大传输长度。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
length	number	最大传输长度

- **错误码**

错误码	错误信息	说明
13000	设备不支持NFC	–
13001	系统NFC开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术

13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.transceive

> 该 方法 使用方式为 IlsoDep.transceive(Object object)

- **功能说明：**发送数据。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
data	ArrayBuffer	-	是	需要传递的二进制数据
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
data	ArrayBuffer	-

- **错误码**

错误码	错误信息	说明
13000	设备不支持NFC	-
13001	系统NFC开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效

13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getHistoricalBytes

> 该 方法 使用方式为 IsoDep.getHistoricalBytes()

- **功能说明：**获取复位信息。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
histBytes	ArrayBuffer	返回历史二进制数据

- **错误码**

错误码	错误信息	说明
13000	设备不支持NFC	-
13001	系统NFC开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC – MifareClassic

最近更新时间：2023-10-20 15:19:20

MifareClassic 标签。

.connect

该 方法 使用方式为 MifareClassic.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描

13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.close

> 该 方法 使用方式为 MifareClassic.close()

- **功能说明：**断开连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
1300	设备不支持 NFC	–

0		
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeout

> 该 方法 使用方式为 MifareClassic.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object。

属性	类型	默认	必填	说明

		值		
timeout	number	–	是	设置超时时间 (ms)
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术

13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

> 该 方法 使用方式为 MifareClassic.isConnected()

> **说明：**
> 该接口已废弃，连接状态开发者自行维护即可

- **功能说明：**检查是否已连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效

13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getMaxTransceiveLength

> 该 方法 使用方式为 MifareClassic.getMaxTransceiveLength()

- **功能说明：**获取最大传输长度
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
length	number	最大传输长度

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败

13016	connect fail	连接失败

.transceive

> 该 方法 使用方式为 MifareClassic.transceive(Object object)

- **功能说明：**发送数据，对于 MifareClassic 的分块读写。
  - 指令 0x30 + 块号可以用于读取某个块的数据。
  - 指令 0xA0 + 块号 + 待写入数据可以用于往某个块写入数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	ArrayBuffer	–	是	需要传递的二进制数据
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
data	ArrayBuffer	–

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权

13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC – MifareUltralight

最近更新时间：2023-10-20 15:19:20

MifareUltralight 标签。

.connect

> 该 方法 使用方式为 MifareUltralight.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not	尝试在未开始 NFC 扫描时，停止 NFC 扫描

	started	
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.close

> 该 方法 使用方式为 MifareUltralight.close()

- **功能说明：** 断开连接。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–

13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeou

> 该 方法 使用方式为 MifareUltralight.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
timeo	numb	-	是	设置超时时间（ms）

ut	er			
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

该 方法 使用方式为 MifareUltralight.isConnected()

说明：
该接口已废弃，连接状态开发者自行维护即可

- **功能说明：**检查是否已连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描

13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getMaxTransceiveLength

> 该 方法 使用方式为 MifareUltralight.getMaxTransceiveLength()

- **功能说明：**获取最大传输长度。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
length	number	最大传输长度

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.transceive

> 该 方法 使用方式为 MifareUltralight.transceive(Object object)

- **功能说明：**发送数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	ArrayBuffer	–	是	需要传递的二进制数据
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
data	ArrayBuffer	–

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been	未扫描到 NFC 标签

	discovered	
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC – Ndef

最近更新时间：2023-10-20 15:19:21

Ndef 标签

.connect

> 该 方法 使用方式为 Ndef.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

.close

> 该 方法 使用方式为 Ndef.close()

- **功能说明：**断开连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效

13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeout

> 该 方法 使用方式为 Ndef.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
timeout	number	–	是	设置超时时间（ms）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

> 该 方法 使用方式为 Ndef.isConnected()

说明：
该接口已废弃，连接状态开发者自行维护即可

- **功能说明：**检查是否已连接。
- **参数及说明：**Object object。

属性	类型	默认值	必	说明

			填	
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.offNdefMessage

> 该 方法 使用方式为 Ndef.onNdefMessage(function callback)

- **功能说明：**取消监听 Ndef 消息。
- **参数及说明：**function callback。

.onNdefMessage

> 该 方法 使用方式为 Ndef.offNdefMessage(function callback)

- **功能说明：**监听 Ndef 消息。
- **参数及说明：**function callback。

.writeNdefMessage

> 该 方法 使用方式为 Ndef.writeNdefMessage(Object object)

- **功能说明：**重写 Ndef 标签内容。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
uris	Array	–	否	uri 数组
texts	Array	–	否	text 数组
records	Array	–	否	二进制对象数组, 需要指明 id，type 以及 payload (均为 ArrayBuffer 类型)
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明

13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC – NfcA

最近更新时间：2023-10-20 15:19:21

NfcA 标签。

.connect

> 该 方法 使用方式为 NfcA.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接

13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.close

> 该 方法 使用方式为 NfcA.close()

- **功能说明：**断开连接
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效

13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeout

> 该 方法 使用方式为 NfcA.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
timeout	number	–	是	设置超时时间（ms）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

该 方法 使用方式为 NfcA.isConnected()

说明：

该接口已废弃，连接状态开发者自行维护即可。

- **功能说明：**检查是否已连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术

13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getMaxTransceiveLength

> 该 方法 使用方式为 NfcA.getMaxTransceiveLength()

- **功能说明：**获取最大传输长度。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
length	number	最大传输长度

- 错误码

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权

13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.transceive

> 该 方法 使用方式为 NfcA.transceive(Object object)

- **功能说明：**发送数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	ArrayBuffer	-	是	需要传递的二进制数据
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明

data	ArrayBuffer	-

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getAtqa

> 该 方法 使用方式为 NfcA.getAtqa()

- **功能说明：** 获取 ATQA 信息。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
atqa	ArrayBuffer	返回 ATQA/SENS_RES 数据

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败

13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getSak

> 该 方法 使用方式为 NfcA.getSak()

- **功能说明：**获取 SAK 信息。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
sak	number	返回 SAK/SEL_RES 数据

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描

13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC – NfcB

最近更新时间：2023-10-20 15:19:21

.connect

该 方法 使用方式为 NfcB.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接

13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.close

> 该 方法 使用方式为 NfcB.close()

- **功能说明：** 断开连接。
- **参数及说明：** Object object

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败

13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始NFC扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeout

> 该 方法 使用方式为 NfcB.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
timeout	number	–	是	设置超时时间（ms）
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

> 该 方法 使用方式为 NfcB.isConnected()

- **功能说明：**检查是否已连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数

fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getMaxTransceiveLength

> 该 方法 使用方式为 NfcB.getMaxTransceiveLength()

- **功能说明：** 获取最大传输长度。

- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
length	number	最大传输长度

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术

13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.transceive

> 该 方法 使用方式为 NfcB.transceive(Object object)

- **功能说明：**发送数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	ArrayBuffer	–	是	需要传递的二进制数据
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
data	ArrayBuffer	–

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权

13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC – NfcF

最近更新时间：2023-10-20 15:19:21

.connect

该 方法 使用方式为 NfcF.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接

13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.close

> 该 方法 使用方式为 NfcF.close()

- **功能说明：**断开连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描

13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeout

> 该 方法 使用方式为 NfcF.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
timeout	number	–	是	设置超时时间 (ms)
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–

13001	系统 NFC开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

> 该 方法 使用方式为 NfcF.isConnected()

- **功能说明：** 检查是否已连接。
- **参数及说明：** Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NF C扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.getMaxTransceiveLength

> 该 方法 使用方式为 NfcF.getMaxTransceiveLength()

- **功能说明：**获取最大传输长度。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
succes	functi	-	否	接口调用成功的回调函数

s	on			
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
length	number	最大传输长度

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败

13016	connect fail	连接失败

.transceive

> 该 方法 使用方式为 NfcF.transceive(Object object)

- **功能说明：**发送数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	ArrayBuffer	–	是	需要传递的二进制数据
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
data	ArrayBuffer	–

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描

13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

NFC – NfcV

最近更新时间：2023-10-20 15:19:21

.connect

该 方法 使用方式为 NfcV.connect()

- **功能说明：**连接 NFC 标签。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	–
13001	系统 NFC 开关未打开	–
13010	未知错误	–
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been	未扫描到 NFC 标签

	discovered	
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.close

> 该 方法 使用方式为 NfcV.close()

- **功能说明：**断开连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery	已经开始 NFC 扫描

	already started	
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.setTimeout

> 该 方法 使用方式为 NfcV.setTimeout(Object object)

- **功能说明：**设置超时时间。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
timeout	number	-	是	设置超时时间（ms）
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明

13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.isConnected

> 该 方法 使用方式为 NfcV.isConnected()

> **说明：**
> 该接口已废弃，连接状态开发者自行维护即可。

- **功能说明：**检查是否已连接。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败

13016	connect fail	连接失败

.getMaxTransceiveLength

> 该 方法 使用方式为 NfcV.getMaxTransceiveLength()

- **功能说明：**获取最大传输长度。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
length	number	最大传输长度

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-
13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接

13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

.transceive

> 该 方法 使用方式为 NfcV.transceive(Object object)

- **功能说明：**发送数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	ArrayBuffer	-	是	需要传递的二进制数据
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
data	ArrayBuffer	-

- **错误码**

错误码	错误信息	说明
13000	设备不支持 NFC	-
13001	系统 NFC 开关未打开	-

13010	未知错误	-
13019	user is not authorized	用户未授权
13011	invalid parameter	参数无效
13012	parse NdefMessage failed	将参数解析为 NdefMessage 失败
13021	NFC discovery already started	已经开始 NFC 扫描
13018	NFC discovery has not started	尝试在未开始 NFC 扫描时，停止 NFC 扫描
13022	Tech already connected	标签已经连接
13023	Tech has not connected	尝试在未连接标签时断开连接
13013	NFC tag has not been discovered	未扫描到 NFC 标签
13014	invalid tech	无效的标签技术
13015	unavailable tech	从标签上获取对应技术失败
13024	function not support	当前标签技术不支持该功能
13017	system internal error	相关读写操作失败
13016	connect fail	连接失败

WiFi

最近更新时间：2023-10-20 15:19:21

onWifiConnected

该 API 使用方法为 wx.onWifiConnected(function listener)

说明：

支持度：

- Android：支持。
- iOS：支持。

- **功能说明：**监听连接上 Wi-Fi 的事件。
- **参数及说明：**function listener
- **连接上 Wi-Fi 的事件的监听函数参数：**Object res。

属性	类型	说明
wifi	WifiInfo	Wi-Fi 信息

onWifiConnected

该 API 使用方法为 wx.onWifiConnected(function listener)

说明：

支持度：

- Android：支持。
- iOS：支持。

- **功能说明：**监听连接上 Wi-Fi 的事件。
- **参数及说明：**function listener。
- **连接上 Wi-Fi 的事件的监听函数参数：**Object res。

属性	类型	说明
wifi	WifiInfo	Wi-Fi 信息

stopWifi

该 API 使用方法为 wx.stopWifi(Object object)

- **功能说明：**关闭 Wi-Fi 模块。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
12000	not init	未先调用 startWifi 接口
12001	system not support	当前系统不支持相关能力
12002	password error Wi-Fi	密码错误
12003	connection timeout	连接超时，仅 Android 支持
12004	duplicate request	重复连接 Wi-Fi
12005	wifi not turned on	Android 特有，未打开 Wi-Fi 开关
12006	gps not turned on	Android 特有，未打开 GPS 定位开关
12007	user denied	用户拒绝授权链接 Wi-Fi
12008	invalid SSID	无效 SSID
12009	system config err	系统运营商配置拒绝连接 Wi-Fi
12010	system internal error	系统其他错误，需要在 errmsg 打印具体的错误原因

12011	weapp in background	应用在后台无法配置 Wi-Fi
12013	wifi config may be expired	系统保存的 Wi-Fi 配置过期，建议忘记 Wi-Fi 后重试，仅 Android 支持
12014	invalid WEP / WPA password	iOS 特有，无效的 WEP / WPA 密码

- **示例代码**

```
wx.stopWifi({
  success (res) {
    console.log(res.errMsg)
  }
})
```

startWifi

> 该 API 使用方法为 wx.startWifi(Object object)

- **功能说明：**初始化 Wi-Fi 模块。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **错误码**

错误码	错误信息	说明
0	ok	正常
12000	not init	未先调用 startWifi 接口

12001	system not support	当前系统不支持相关能力
12002	password error Wi-Fi	密码错误
12003	connection timeout	连接超时, 仅 Android 支持
12004	duplicate request	重复连接 Wi-Fi
12005	wifi not turned on	Android 特有，未打开 Wi-Fi 开关
12006	gps not turned on	Android 特有，未打开 GPS 定位开关
12007	user denied	用户拒绝授权链接 Wi-Fi
12008	invalid SSID	无效 SSID
12009	system config err	系统运营商配置拒绝连接 Wi-Fi
12010	system internal error	系统其他错误，需要在 errmsg 打印具体的错误原因
12011	weapp in background	应用在后台无法配置 Wi-Fi
12013	wifi config may be expired	系统保存的 Wi-Fi 配置过期，建议忘记 Wi-Fi 后重试，仅 Android 支持
12014	invalid WEP / WPA password	iOS 特有，无效的 WEP / WPA 密码

- **示例代码**

```
wx.startWifi({
  success (res) {
    console.log(res.errMsg)
  }
})
```

onGetWifiList

> 该 API 使用方法为 wx.onGetWifiList(function listener)

- **功能说明：**监听获取到 Wi-Fi 列表数据事件。
- **参数及说明：**Object res 参数，function listener，获取到 Wi-Fi 列表数据事件的监听函数。

属性	类型	说明
wifiList	Array.<WifiInfo>	Wi-Fi 列表数据

offGetWifiList

> 该 API 使用方法为 wx.offGetWifiList(function listener)

- **功能说明：**移除获取到 Wi-Fi 列表数据事件的监听函数。
- **参数及说明：**function listener，onGetWifiList 传入的监听函数。不传此参数则移除所有监听函数。

getConnectedWifi

> 该 API 使用方法为 wx.getConnectedWifi(Object object)

- **功能说明：**获取已连接中的 Wi-Fi 信息。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
partialInfo	boolean	false	否	是否需要返回部分 Wi-Fi 信息
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res

属性	类型	说明
wifi	WifiInfo	Wi-Fi 信息

- **错误码**

错误码	错误信息	说明
0	ok	正常
12000	not init	未先调用 startWifi 接口
12001	system not support	当前系统不支持相关能力
12002	password error Wi-Fi	密码错误
12003	connection timeout	连接超时，仅 Android 支持
12004	duplicate request	重复连接 Wi-Fi
12005	wifi not turned on	Android 特有，未打开 Wi-Fi 开关
12006	gps not turned on	Android 特有，未打开 GPS 定位开关
12007	user denied	用户拒绝授权链接 Wi-Fi
12008	invalid SSID	无效 SSID
12009	system config err	系统运营商配置拒绝连接 Wi-Fi
12010	system internal error	系统其他错误，需要在 errmsg 打印具体的错误原因
12011	weapp in background	应用在后台无法配置 Wi-Fi
12013	wifi config may be expired	系统保存的 Wi-Fi 配置过期，建议忘记 Wi-Fi 后重试，仅 Android 支持
12014	invalid WEP / WPA password	iOS 特有，无效的 WEP / WPA 密码

WifiInfo

功能说明：Wifi 信息。

> **注意：**
> - Android wx.connectWifi / wx.getConnectedWifi 若设置了 partialInfo:true，或者调用了 wx.onWifiConnectedWithPartialInfo 事件。将会返回只包含 SSID 属性的 WifiInfo 对象。
> - iOS wx.getConnectedWifi 若设置了 partialInfo:true，将会返回只包含 SSID、BSSID 属性的 WifiInfo 对象，且需要用户开启宿主客户端定位权限才能正确返回结果。
> - 在某些情况下，可能 Wi-Fi 已经连接成功，但会因为获取不到完整的 WifiInfo 对象报错。具体错误信息为 `errCode:12010, errMsg: can't gain current wifi` 或 `no wifi is connected` 。如果开发者不需要完整的 WifiInfo 对象，则可以通过采取上述策略解决报错问题。

属性

- **string SSID：**Wi-Fi 的 SSID。
- **string BSSID：**Wi-Fi 的 BSSID。
- **boolean secure：**Wi-Fi 是否安全。
- **number signalStrength：**Wi-Fi 信号强度，Android 取值 0 ~ 100 ，iOS 取值 0 ~ 1，值越大强度越大。
- **number frequency：**Wi-Fi 频段单位 MHz。

日历

最近更新时间：2023-10-20 15:19:21

addPhoneCalendar

> 该 API 使用方法为 wx.addPhoneRepeatCalendar(Object object)

- **功能说明：**向系统日历添加事件。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
title	string	-	是	日历事件标题
startTime	number	-	是	开始时间的 unix 时间戳
allDay	boolean	-	否	是否全天事件，默认 false
description	string	-	否	事件说明
location	string	-	否	事件位置
endTime	string	-	否	结束时间的 unix 时间戳，默认与开始时间相同
alarm	boolean	-	否	是否提醒，默认 true
alarmOffset	number	-	否	提醒提前量，单位：秒，默认0表示开始时提醒
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

addPhoneRepeatCalendar

> 该 API 使用方法为 wx.addPhoneCalendar(Object object)

- **功能说明：**向系统日历添加重复事件。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
title	string	–	是	日历事件标题
startTime	number	–	是	开始时间的 unix 时间戳 (1970年1月1日开始所经过的秒数)
allDay	boolean	–	否	是否全天事件，默认 false
description	string	–	否	事件说明
location	string	–	否	事件位置
endTime	string	–	否	结束时间的 unix 时间戳，默认与开始时间相同
alarm	boolean	–	否	是否提醒，默认 true
alarmOffset	number	–	否	提醒提前量，单位：秒，默认0表示开始时提醒
repeatInterval	string	–	否	重复周期，默认 month 每月重复
repeatEndTime	number	–	否	重复周期结束时间的 unix 时间戳，不填表示一直重复
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **repeatInterval**

合法值	说明
day	每天重复

week	每周重复
month	每月重复。该模式日期不能大于 28 日
year	每年重复

剪切板

最近更新时间：2023-10-20 15:19:21

setClipboardData

> 该 API 使用方法为 wx.setClipboardData(Object object)

- **功能说明：**设置系统剪贴板的内容，对单个用户而言，每秒调用不超过1次。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
data	string	–	是	剪贴板的内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.setClipboardData({
  data: 'data',
  success(res) {
    wx.getClipboardData({
      success(res) {
        console.log(res.data) // data
      }
    })
  }
})
```

getClipboardData

> 该 API 使用方法为 wx.getClipboardData(Object object)

- **功能说明：**获取系统剪贴板的内容，对单个用户而言，每秒调用不超过 `1` 次。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object object。

属性	类型	说明
data	string	剪贴板的内容

- **示例代码：**

```
wx.getClipboardData({
  success(res) {
    console.log(res.data)
  }
})
```

网络

最近更新时间：2023-10-20 15:19:22

onNetworkStatusChange

> 该 API 使用方法为 wx.onNetworkStatusChange(function callback)

- **功能说明：**监听网络状态变化事件。
- **参数及说明：**function callback。
- **网络状态变化事件的回调函数：**Object res。

属性	类型	说明
isConnected	boolean	当前是否有网络连接
networkType	string	网络类型

- **networkType 的合法值**

值	说明
wifi	wifi 网络
2g	2g 网络
3g	3g 网络
4g	4g 网络
unknown	Android 下不常见的网络类型
none	无网络

- **示例代码：**

```
wx.onNetworkStatusChange(function (res) {
  console.log(res.isConnected)
  console.log(res.networkType)
})
```

getNetworkType

> 该 API 使用方法为 wx.getNetworkType(Object object)

- **功能说明：**获取网络类型。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object res。

属性	类型	说明
networkType	string	网络类型

- **res.networkType 的合法值**

值	说明
wifi	wifi 网络
2g	2g 网络
3g	3g 网络
4g	4g 网络
unknown	Android 下不常见的网络类型
none	无网络

- **示例代码：**

```
wx.getNetworkType({
  success(res) {
    const networkType = res.networkType
  }
})
```

offNetworkStatusChange

该 API 使用方法为 wx.offNetworkStatusChange(function listener)

- **功能说明：**移除网络状态变化事件的监听函数。
- **参数及说明：**function listener，onNetworkStatusChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }

wx.onNetworkStatusChange(listener)
wx.offNetworkStatusChange(listener) // 需传入与监听时同一个的函数对象
```

加密

最近更新时间：2024-05-24 15:58:42

getRandomValues

> 该 API 使用方法为 wx.getRandomValues(Object object)

- **功能说明：**获取密码学安全随机数。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
length	number	–	是	整数，生成随机数的字节数，最大 1048576
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
randomValues	ArrayBuffer	随机数内容，长度为传入的字节数

- **示例代码**

```
wx.getRandomValues({
  length: 6 // 生成 6 个字节长度的随机数,
  success: res => {
    console.log(wx.arrayBufferToBase64(res.randomValues)) // 转换为 base64 字符串
后打印
  }
})
```

键盘

最近更新时间：2023-10-20 15:19:22

onKeyboardHeightChange

该 API 使用方法为 wx.getSelectedTextRange(Object object)

- **功能说明：**在 input、textarea 等 focus 之后，获取输入框的光标位置。**只有在 focus 的时候调用此接口才有效。**
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object res。

属性	类型	说明
start	number	输入框光标起始位置
end	number	输入框光标结束位置

- **示例代码：**

```
wx.getSelectedTextRange({
  complete: res => {
    console.log('getSelectedTextRange res', res.start, res.end)
  }
})
```

hideKeyboard

该 API 使用方法为 wx.hideKeyboard(Object object)

- **功能说明：**在 input、textarea 等 focus 拉起键盘之后，手动调用此接口收起键盘，使用方法为。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.hideKeyboard({
  complete: res => {
    console.log('hideKeyboard res', res)
  }
})
```

offKeyboardHeightChange

> 该 API 使用方法为 为 wx.offKeyboardHeightChange(function listener)

- **功能说明：**移除键盘高度变化事件的监听函数。
- **参数及说明：**function listener，onKeyboardHeightChange 传入的监听函数。不传此参数则移除所有监听函数。。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onKeyboardHeightChange(listener)
wx.offKeyboardHeightChange(listener) // 需传入与监听时同一个的函数对象
```

getSelectTextRange

> 该 API 使用方法为 wx.getSelectedTextRange(Object object)

- **功能说明：**在 input、textarea 等 focus 之后，获取输入框的光标位置。**只有在 focus 的时候调用此接口才有效。**

- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object res。

属性	类型	说明
start	number	输入框光标起始位置
end	number	输入框光标结束位置

- **示例代码：**

```
wx.getSelectedTextRange({
  complete: res => {
    console.log('getSelectedTextRange res', res.start, res.end)
  }
})
```

电话

最近更新时间：2023-10-20 15:19:22

makePhonecall

> 该 API 使用方法为 wx.makePhoneCall(Object object)

- **功能说明：**拨打电话。
- **参数：**Object object。

属性	类型	默认值	必填	说明
phoneNumber	string	-	是	需要拨打的电话
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.makePhoneCall({
  phoneNumber: '1340000' //仅为示例，并非真实的电话号码
})
```

加速计

最近更新时间：2023-10-20 15:19:22

starAccelerometer

该 API 使用方法为 wx.startAccelerometer(Object object)

> **注意：**
> 根据机型性能、当前 CPU 与内存的占用情况，`interval` 的设置与实际 wx.onAccelerometerChange 回调函数的执行频率会有一些出入。

- **功能说明：**开始监听加速度数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
interval	string	normal	否	监听加速度数据回调函数的执行频率
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.interval 的合法值。**

值	说明
game	适用于更新游戏的回调频率，在20ms/次 左右
ui	适用于更新 UI 的回调频率，在60ms/次 左右
normal	普通的回调频率，在200ms/次 左右

- **示例代码：**

```
wx.startAccelerometer({
```

```
  interval: 'game'
})
```

stopAccelerometer

> 该 API 使用方法为 wx.stopAccelerometer(Object object)

- **功能说明：**停止监听加速度数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.stopAccelerometer()
```

onAccelerometerChange

> 该 API 使用方法为 wx.onAccelerometerChange(function listener)

- **功能说明：**监听加速度数据事件。频率根据 wx.startAccelerometer() 的 interval 参数。可使用 wx.stopAccelerometer() 停止监听。
- **参数及说明：**function callback。
- 加速度数据事件的回调函数：

属性	类型	说明
x	number	X 轴
y	number	Y 轴
z	number	Z 轴

- **示例代码：**

```
wx.onAccelerometerChange(function (res) {
  console.log(res.x)
  console.log(res.y)
  console.log(res.z)
})
```

offAccelerometerChange

> 该 API 使用方法为 wx.offAccelerometerChange(function listener)

- **功能说明：**移除加速度数据事件的监听函数。
- **参数及说明：**function listener，onAccelerometerChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onAccelerometerChange(listener)
wx.offAccelerometerChange(listener) // 需传入与监听时同一个的函数对
```

罗盘

最近更新时间：2023-10-20 15:19:22

starCompass

> 该 API 使用方法为 wx.startCompass(Object object)

- **功能说明：**开始监听罗盘数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.startCompass()
```

stopCompass

> 该 API 使用方法为 wx.stopCompass(Object object)

- **功能说明：**停止监听罗盘数据，使用方法为。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码**

```
wx.stopCompass()
```

onCompassChange

> 该 API 使用方法为 wx.onCompassChange(function callback)

- **功能说明：**监听罗盘数据变化事件。频率：5次/秒，接口调用后会自动开始监听，可使用 wx.stopCompass 停止监听。
- **参数及说明：**function callback。
- **罗盘数据变化事件的回调函数：**Object res

属性	类型	说明
direction	number	面对的方向度数
accuracy	number/string	精度

- **示例代码**

```
wx.onCompassChange(function (res) {
  console.log(res.direction)
})
```

- **accuracy 在 iOS/Android 的差异：**由于平台差异，accuracy 在 iOS/Android 的值不同。
  - iOS：accuracy 是一个 number 类型的值，表示相对于磁北极的偏差。0表示设备指向磁北，90表示指向东，180表示指向南，依此类推。
  - Android：accuracy 是一个 string 类型的枚举值。

值	说明
high	高精度
medium	中等精度
low	低精度
no-contact	不可信，传感器失去连接
unreliable	不可信，原因未知
unknow	未知的精度枚举值，即该 Android 系统此时返回的表示精度的 value 不是一个标

${value}	准的精度枚举值

offCompassChange

> 该 API 使用方法为 wx.offCompassChange(function listener)

- **功能说明：**移除罗盘数据变化事件的监听函数。
- **参数及说明：**function listener，onCompassChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onCompassChange(listener)
wx.offCompassChange(listener) // 需传入与监听时同一个的函数对象
```

陀螺仪

最近更新时间：2023-10-20 15:19:22

startGyroscope

> 该 API 使用方法为 wx.startGyroscope(Object object)

- **功能说明：**开始监听陀螺仪数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
interval	string	normal	否	监听陀螺仪数据回调函数的执行频率
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.interval 的合法值：**

值	说明
game	适用于更新游戏的回调频率，在20ms/次 左右
ui	适用于更新 UI 的回调频率，在60ms/次 左右
normal	普通的回调频率，在200ms/次 左右

stopGyroscope

> 该 API 使用方法为 wx.stopGyroscope(Object object)

- **功能说明：**停止监听陀螺仪数据。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

onGyroscopeChange

> 该 API 使用方法为 wx.onGyroscopeChange(function callback)

- **功能说明：**监听陀螺仪数据变化事件。频率根据 wx.startGyroscope() 的 interval 参数。可以使用 wx.stopGyroscope() 停止监听。
- **参数及说明：**function callback。
- **陀螺仪数据变化事件的回调函数：**Object res。

属性	类型	说明
res	Object	–

- **res 的结构**

属性	类型	说明
x	number	x 轴的角速度
y	number	y 轴的角速度
z	number	z 轴的角速度

offGyroscopeChange

> 该 API 使用方法为 wx.offGyroscopeChange(function listener)

- **功能说明：**移除陀螺仪数据变化事件的监听函数。
- **参数及说明：**function listener，onGyroscopeChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码**

```
const listener = function (res) { console.log(res) }
```

```
wx.onGyroscopeChange(listener)
wx.offGyroscopeChange(listener) // 需传入与监听时同一个的函数对象
```

扫码

最近更新时间：2023-10-20 15:19:22

scanCode

> 该 API 使用方法为 wx.scanCode(Object object)

- **功能说明：**调起客户端扫码界面进行扫码。
- **参数及说明：**Object object。

属性	类型	合法值及说明	默认值	必填	说明
onlyFromCamera	boolean	–	false	否	是否只能从相机扫码，不允许从相册选择图片
scanType	Array.	• barCode：一维码 • qrCode：二维码 • datamatrix：Data Matrix 码 • pdf417：PDF417 条码	['barCode', 'qrCode']	否	扫码类型
success	function	–	–	否	接口调用成功的回调函数
fail	function	–	–	否	接口调用失败的回调函数
complete	function	–	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	合法值及说明	说明
result	string	–	所扫码的内容
sca	stri	• QR_CODE：二维码	所扫码的类型

nType	string	• AZTEC：一维码 • CODABAR：一维码 • CODE_39：一维码 • CODE_93：一维码 • CODE_128：一维码 • DATA_MATRIX：二维码 • EAN_8：一维码 • EAN_13：一维码 • ITF：一维码 • MAXICODE：一维码 • PDF_417：二维码 • RSS_14：一维码 • RSS_EXPANDED：一维码 • UPC_A：一维码 • UPC_E：一维码 • UPC_EAN_EXTENSION：一维码 • WX_CODE：二维码 • CODE_25：一维码	
charSet	string	–	所扫码的字符集
path	string	–	当所扫的码为当前小程序二维码时，会返回此字段，内容为二维码携带的 path
rawData	string	–	原始数据，base64 编码

- **示例代码：**

```
// 允许从相机和相册扫码
wx.scanCode({
  success (res) {
    console.log(res)
  }
})

// 只允许从相机扫码
```

```
wx.scanCode({
  onlyFromCamera: true,
  success (res) {
    console.log(res)
  }
})
```

短信

最近更新时间：2023-10-20 15:19:22

sendSms

> 该 API 使用方法为 wx.sendSms(Object object)

- **功能说明：**拉起手机发送短信界面。
- **参数及说明：**Object object

属性	类型	默认值	必填	说明
phoneNumber	string	–	否	预填到发送短信面板的手
content	string	–	否	预填到发送短信面板的内容
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

振动

最近更新时间：2023-10-20 15:19:23

vibranteShort

> 该 API 使用方法为 wx.vibrateShort(Object object)

- **功能说明：**使手机发生较短时间的振动（15 ms）。仅在 iPhone 7/7 Plus 以上及 Android 机型生效。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

vibranteLong

> 该 API 使用方法为 wx.vibrateLong(Object object)

- **功能说明：**使手机发生较长时间的振动（400 ms)。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

联系人

最近更新时间：2023-10-20 15:19:23

chooseContact

> 该 API 使用方法为 wx.chooseContact(Object object)

- **功能说明：**拉起手机通讯录，选择联系人。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数：**Object object。

属性	类型	说明
phoneNumber	string	手机号
displayName	string	联系人姓名
phoneNumberList	string	选定联系人的所有手机号（部分 Android 系统只能选联系人而不能选特定手机号）

addPhoneContact

> 该 API 使用方法为 wx.addPhoneContact(Object object)

- **功能说明：**添加手机通讯录联系人。用户可以选择将该表单以**新增联系人**或**添加到已有联系人**的方式，写入手机系统通讯录。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明

firstName	string	-	是	名字
photoFilePath	string	-	否	头像本地文件路径
nickName	string	-	否	昵称
lastName	string	-	否	姓氏
middleName	string	-	否	中间名
remark	string	-	否	备注
mobilePhoneNumber	string	-	否	手机号
weChatNumber	string	-	否	微信号
addressCountry	string	-	否	联系地址国家
addressState	string	-	否	联系地址省份
addressCity	string	-	否	联系地址城市
addressStreet	string	-	否	联系地址街道
addressPostalCode	string	-	否	联系地址邮政编码
organization	string	-	否	公司
title	string	-	否	职位
workFaxNumber	string	-	否	工作传真
workPhoneNumber	string	-	否	工作电话
hostNumber	string	-	否	公司电话
email	string	-	否	电子邮件
url	string	-	否	网站
workAddressCountry	string	-	否	工作地址国家
workAddressState	string	-	否	工作地址省份

workAddressCity	string	-	否	工作地址城市
workAddressStreet	string	-	否	工作地址街道
workAddressPostalCode	string	-	否	工作地址邮政编码
homeFaxNumber	string	-	否	住宅传真
homePhoneNumber	string	-	否	住宅电话
homeAddressCountry	string	-	否	住宅地址国家
homeAddressState	string	-	否	住宅地址省份
homeAddressCity	string	-	否	住宅地址城市
homeAddressStreet	string	-	否	住宅地址街道
homeAddressPostalCode	string	-	否	住宅地址邮政编码
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

电量

最近更新时间：2023-10-20 15:19:23

getBatteryInfoSync

> 该 API 使用方法为 Object wx.getBatteryInfoSync()

- **功能说明：**wx.getBatteryInfo 的同步版本。
- **返回值：**Object res。

属性	类型	说明
level	number	设备电量，范围1 - 100
isCharging	boolean	是否正在充电中

getBatteryInfo

> 该 API 使用方法为 wx.getBatteryInfo(Object object)

- **功能说明：**获取设备电量。同步 API wx.getBatteryInfoSync 在 iOS 上不可用。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明
level	number	设备电量，范围1 - 100
isCharging	boolean	是否正在充电中

屏幕

最近更新时间：2023-10-20 15:19:23

setVisualEffectOnCapture

> 该 API 使用方法为 wx.setVisualEffectOnCapture(Object object)

- **功能说明：**设置截屏/录屏时屏幕表现，仅支持在 Android 端调用。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
visualEffect	string	none	否	截屏/录屏时的表现，仅支持 none / hidden，传入 hidden 则表示在截屏/录屏时隐藏屏幕
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）

setScreenBrightness

> 该 API 使用方法为 wx.setScreenBrightness(Object object)

- **功能说明：**设置屏幕亮度。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
value	number	-	是	屏幕亮度值，范围0 ~ 1，0最暗，1最亮。在 Android 端支持传入特殊值-1，表示屏幕亮度跟随系统变化

success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

setKeepScreenOn

> 该 API 使用方法为 wx.setKeepScreenOn(Object object)

- **功能说明：**设置是否保持常亮状态。仅在当前小程序生效，离开小程序后设置失效。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
keepScreenOn	boolean	–	是	是否保持屏幕常亮
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **示例代码：**

```
wx.setKeepScreenOn({
  keepScreenOn: true
})
```

onUserCaptureScreen

> 该 API 使用方法为 wx.onUserCaptureScreen(function listener)

- **功能说明：**监听用户主动截屏事件。用户使用系统截屏按键截屏时触发，只能注册一个监听。
- **参数及说明：**function listener，用户主动截屏事件的监听函数。
- **示例代码：**

```
wx.onUserCaptureScreen(function (res) {
  console.log('用户截屏了')
})
```

onScreenRecordingStateChanged

> 该 API 使用方法为 wx.onScreenRecordingStateChanged(function listener)

- **功能说明：**监听用户录屏事件。
- **参数及说明：**function listener，用户录屏事件的监听函数。

属性	类型	说明
state	string	录屏状态，合法值有： • start：开始录屏 • stop：结束录屏

- **示例代码：**

```
// 监听用户录屏事件
const handler = function (res) {
  console.log(res.state)
}
wx.onScreenRecordingStateChanged(handler)

// 取消监听用户录屏事件
wx.offScreenRecordingStateChanged(handler)
```

offUserCaptureScreen

> 该 API 使用方法为 wx.offUserCaptureScreen(function callback)

- **功能说明：**用户主动截屏事件。取消事件监听。
- **参数及说明：**function callback，用户主动截屏事件的回调函数。

offScreenRecordingStateChanged

> 该 API 使用方法为 wx.offScreenRecordingStateChanged(function listener)

- **功能说明：**移除用户录屏事件的监听函数。
- **参数及说明：**function listener，onScreenRecordingStateChanged 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
// 监听用户录屏事件
const handler = function (res) {
  console.log(res.state)
}
wx.onScreenRecordingStateChanged(handler)

// 取消监听用户录屏事件
wx.offScreenRecordingStateChanged(handler)
```

```
const listener = function (res) { console.log(res) }

wx.onScreenRecordingStateChanged(listener)
wx.offScreenRecordingStateChanged(listener) // 需传入与监听时同一个的函数对象
```

getScreenRecordingState

> 该 API 使用方法为 wx.getScreenRecordingState(Object object)

- **功能说明：**查询用户是否在录屏。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object res。

属性	类型	说明

state	string	录屏状态，合法值有： • on：开启 • off：关闭

- **示例代码：**

```
wx.getScreenRecordingState({
  success: function (res) {
    console.log(res.state)
  },
})
```

getScreenBrightness

该 API 使用方法为 wx.getScreenBrightness(Object object)

> **说明：**
> 若 Android 系统设置中开启了自动调节亮度功能，则屏幕亮度会根据光线自动调整，该接口仅能获取自动调节亮度之前的值，而非实时的亮度值。

- **功能说明：**获取屏幕亮度。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object object。

属性	类型	说明
value	number	屏幕亮度值，范围0～1，0最暗，1最亮

设备方向

最近更新时间：2023-10-20 15:19:23

stopDeviceMotionListening

该 API 使用方法为 wx.stopDeviceMotionListening(Object object)

- **功能说明：**停止监听设备方向的变化。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数
complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

startDeviceMotionListening

该 API 使用方法为 wx.startDeviceMotionListening(Object object)

- **功能说明：**开始监听设备方向的变化。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
interval	string	normal	否	监听设备方向的变化回调函数的执行频率，合法值为： • game：适用于更新游戏的回调帧率，在20ms/次左右 • ui：适用于更新 UI 的回调帧率，在60ms/次左右 • normal：普通的回调频率，在200ms/次左右
success	function	–	否	接口调用成功的回调函数
fail	function	–	否	接口调用失败的回调函数

complete	function	–	否	接口调用结束的回调函数（调用成功、失败都会执行）

onDeviceMotionChange

> 该 API 使用方法为 wx.onDeviceMotionChange(function listener)

- **功能说明：**监听设备方向变化事件。频率根据 wx.startDeviceMotionListening() 的 interval 参数。可以使用 wx.stopDeviceMotionListening() 停止监听。
- **参数及说明：**function listener，设备方向变化事件的监听函数。

属性	类型	说明
alpha	number	当 手机坐标 X/Y 和 地球 X/Y 重合时，绕着 Z 轴转动的夹角为 alpha，范围值为 [0, 2*PI)。逆时针转动为正。
beta	number	当手机坐标 Y/Z 和地球 Y/Z 重合时，绕着 X 轴转动的夹角为 beta。范围值为 [-1*PI, PI) 。顶部朝着地球表面转动为正。也有可能朝着用户为正。
gamma	number	当手机 X/Z 和地球 X/Z 重合时，绕着 Y 轴转动的夹角为 gamma。范围值为 [-1*PI/2, PI/2)。右边朝着地球表面转动为正。

offDeviceMotionChange

> 该 API 使用方法为 wx.offDeviceMotionChange(function listener)

- **功能说明：**移除设备方向变化事件的监听函数。
- **参数及说明：**function listener，onDeviceMotionChange 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }

wx.onDeviceMotionChange(listener)
wx.offDeviceMotionChange(listener) // 需传入与监听时同一个的函数对象
```

内存

最近更新时间：2023-10-20 15:19:23

onMemoryWarning

> 该 API 使用方法为 wx.onMemoryWarning(function listener)

- **功能说明：**监听内存不足告警事件。当 iOS/Android 向小程序进程发出内存警告时，触发该事件。触发该事件不意味小程序进程被清除，大部分情况下仅是告警，开发者可在收到通知后回收一些不必要资源避免进一步加剧内存紧张。
- **参数及说明：**function listener，内存不足告警事件的监听函数。

属性	类型	说明
level	number	内存告警等级，只有 Android 才有，对应系统宏定义

- **level 合法值**

合法值	说明
5	TRIM_MEMORY_RUNNING_MODERATE
10	TRIM_MEMORY_RUNNING_LOW
15	TRIM_MEMORY_RUNNING_CRITICAL

- **示例代码：**

```
wx.onMemoryWarning(function () {
  console.log('onMemoryWarningReceive')
})
```

offMemoryWarning

> 该 API 使用方法为 wx.offMemoryWarning(function listener)

- **功能说明：**移除内存不足告警事件的监听函数。
- **参数及说明：**function listener，onMemoryWarning 传入的监听函数。不传此参数则移除所有监听函数。
- **示例代码：**

```
const listener = function (res) { console.log(res) }
```

```
wx.onMemoryWarning(listener)
wx.offMemoryWarning(listener) // 需传入与监听时同一个的函数对象
```

无障碍

最近更新时间：2023-10-20 15:19:23

checkIsOpenAccessibility

> 该 API 使用方法为 wx.checkIsOpenAccessibility(Object object)

- **功能说明：**检测是否开启视觉无障碍功能。
- **参数及说明：**Object object。

属性	类型	默认值	必填	说明
success	function	-	否	接口调用成功的回调函数
fail	function	-	否	接口调用失败的回调函数
complete	function		否	接口调用结束的回调函数（调用成功、失败都会执行）

- **object.success 回调函数参数：**Object object。

属性	类型	说明 ·
open	Boolean	iOS 上开启辅助功能旁白，Android 开启 talkback 时返回 true

WXML

最近更新时间：2023-11-10 15:39:28

createSelectorQuery

> 该 API 使用方法为 SelectorQuery wx.createSelectorQuery()

- **功能说明：**返回一个 SelectorQuery 对象实例，在自定义组件或包含自定义组件的页面中，用 `this.createSelectorQuery()` 来代替。
- **返回值：** SelectorQuery 。
- **示例代码：**

```
const query = wx.createSelectorQuery()
query.select('#the-id').boundingClientRect()
query.selectViewport().scrollOffset()
query.exec(function (res) {
  res[0].top // #the-id节点的上边界坐标
  res[1].scrollTop // 显示区域的竖直滚动位置
})
```

createIntersectionObserver

> 该 API 使用方法为 IntersectionObserver wx.createIntersectionObserver(Object component, Object options)

- **功能说明：**创建并返回一个 IntersectionObserver 对象实例。在自定义组件或包含自定义组件的页面中，应使用 `this.createIntersectionObserver([options])` 来代替。
- **参数及说明：**Object this
  - **自定义组件实例：**Object options。
  - **选项**

属性	类型	默认值	必填	说明
thresholds	Array.<number>	[0]	否	一个数值数组，包含所有阈值。
initialRatio	number	0	否	初始的相交比例，如果调用时检测到的相交比例与这个值不相等且达到阈值，则会触发一次监听器的

				回调函数。
observeAll	boolean	false	否	是否同时观测多个目标节点（而非一个），如果设为 true，observe 的 targetSelector 将选中多个节点（注意：同时选中过多节点将影响渲染性能）。

- **返回值：** IntersectionObserver 。

IntersectionObserver

.relativeTo

该 方法 使用方式为 IntersectionObserver.relativeTo(string selector, Object margins)

- **功能说明：** 使用选择器指定一个节点，作为参照区域之一。
- **参数及说明：** string selector，用来扩展（或收缩）参照节点布局区域的边界。
- **选择器：** Object margins。

属性	类型	默认值	必填	说明
left	number	-	否	节点布局区域的左边界
right	number	-	否	节点布局区域的右边界
top	number	-	否	节点布局区域的上边界
bottom	number	-	否	节点布局区域的下边界

.relativeToViewport

该 方法 使用方式为 IntersectionObserver.relativeToViewport(Object margins)

- **功能说明：** 指定页面显示区域作为参照区域之一。
- **参数及说明：** Object margins ，用来扩展（或收缩）参照节点布局区域的边界。

属性	类型	默认值	必填	说明
left	number	-	否	节点布局区域的左边界
right	number	-	否	节点布局区域的右边界
top	number	-	否	节点布局区域的上边界
bottom	number	-	否	节点布局区域的下边界

- **示例代码**

  下面的示例代码中，如果目标节点（用选择器 .target-class 指定）进入显示区域以下100px时，就会触发回调函数。

```
Page({
  onLoad() {
    wx.createIntersectionObserver().rela
```

.disconnect

> 该 方法 使用方式为 IntersectionObserver.disconnect()

功能说明：停止监听，回调函数将不再触发。

.observe

> 该 方法 使用方式为 IntersectionObserver.observe(string targetSelector, function callback)

- **功能说明：**指定目标节点并开始监听相交状态变化情况。
- **参数及说明：**string targetSelector。
- **选择器：**function callback。
- **监听相交状态变化的回调函数参数：**Object res。

属性	类型	说明
intersectionRatio	number	相交比例
intersectionRect	object	相交区域的边界
boundingClientRect	object	目标边界
relativeRect	object	参照区域的边界
time	number	相交检测时的时间戳

- **res.intersectionRect 的结构**

属性	类型	说明
left	number	左边界
right	number	右边界
top	number	上边界

bottom	number	下边界

- **res.boundingClientRect 的结构**

属性	类型	说明
left	number	左边界
right	number	右边界
top	number	上边界
bottom	number	下边界

- **res.relativeRect 的结构**

属性	类型	说明
left	number	左边界
right	number	右边界
top	number	上边界
bottom	number	下边界

NodesRef

.fields

> 该 方法 使用方式为 NodesRef.fields(Object fields)

- **功能说明：**获取节点的相关信息。需要获取的字段在fields中指定。
- **参数及说明：**Object fields。

属性	类型	默认值	必填	说明
id	boolean	false	否	是否返回节点 id
dataset	boolean	false	否	是否返回节点 dataset
rect	boolean	false	否	是否返回节点布局位置（left、right、top、bottom）

size	boolean	false	否	是否返回节点尺寸（width、height）
scrollOffset	boolean	false	否	是否返回节点的 scrollLeft scrollTop，节点必须是 scroll-view 或者 viewport
properties	Array.\	–	否	指定属性名列表，返回节点对应属性名的当前属性值（只能获得组件文档中标注的常规属性值，id class style 和事件绑定的属性值不可获取）
computedStyle	Array.\	–	否	指定样式名列表，返回节点对应样式名的当前值
context	boolean	false	否	是否返回节点对应的 Context 对象

> ⚠ **注意：**
> computedStyle 的优先级高于 size，当同时在 computedStyle 里指定了 width/height 和传入了 size: true，则优先返回 computedStyle 获取到的 width/height。

- **返回值：**nodesRef 对应的 selectorQuery。

.boundingClientRect

该 方法 使用方式为 SelectorQuery
NodesRef.boundingClientRect(NodesRef.boundingClientRectCallback callback)

- **功能说明：**添加节点的布局位置的查询请求。相对于显示区域，以像素为单位。其功能类似于 DOM 的 getBoundingClientRect 。
- **参数及说明：**function callback。回调函数，在执行 SelectorQuery.exec 方法后，节点信息会在 callback 中返回。
  - Object res

属性	类型	说明
id	string	节点的 ID
dataset	Object	节点的 dataset
left	number	节点的左边界坐标
right	number	节点的右边界坐标
top	number	节点的上边界坐标

bottom	number	节点的下边界坐标
width	number	节点的宽度
height	number	节点的高度

- **返回值：** SelectorQuery 。
- **示例代码：**

```
Page({
  getRect() {
    wx.createSelectorQuery().select('#the-id').boundingClientRect(function (rect) {
      rect.id // 节点的ID
      rect.dataset // 节点的dataset
      rect.left // 节点的左边界坐标
      rect.right // 节点的右边界坐标
      rect.top // 节点的上边界坐标
      rect.bottom // 节点的下边界坐标
      rect.width // 节点的宽度
      rect.height // 节点的高度
    }).exec()
  },
  getAllRects() {
    wx.createSelectorQuery().selectAll('.a-class').boundingClientRect(function (rects)
{
      rects.forEach(function (rect) {
        rect.id // 节点的ID
        rect.dataset // 节点的dataset
        rect.left // 节点的左边界坐标
        rect.right // 节点的右边界坐标
        rect.top // 节点的上边界坐标
        rect.bottom // 节点的下边界坐标
        rect.width // 节点的宽度
        rect.height // 节点的高度
      })
    }).exec()
  }
})
```

.scrollOffset

> 该 方法 使用方式为 SelectorQuery NodesRef.scrollOffset(NodesRef.scrollOffsetCallback callback)

- **功能说明：**添加节点的滚动位置查询请求。以像素为单位。节点必须是 scroll-view 或者 viewport 。
- **参数及说明：**function callback，回调函数，在执行 SelectorQuery.exec 方法后，节点信息会在 callback 中返回。
  - Object res

属性	类型	说明
id	string	节点的 ID
dataset	Object	节点的 dataset
scrollLeft	number	节点的水平滚动位置
scrollTop	number	节点的竖直滚动位置

- **返回值：** SelectorQuery 。
- **示例代码：**

```
Page({
  getScrollOffset() {
    wx.createSelectorQuery().selectViewport().scrollOffset(function (res) {
      res.id // 节点的ID
      res.dataset // 节点的dataset
      res.scrollLeft // 节点的水平滚动位置
      res.scrollTop // 节点的竖直滚动位置
    }).exec()
  }
})
```

.context

该 方法 使用方式为 SelectorQuery NodesRef.context(NodesRef.contextCallback callback)

- **功能说明：**添加节点的 Context 对象查询请求。目前支持 MapContext 的获取。
- **参数及说明：**function callback，回调函数，在执行 SelectorQuery.exec 方法后，返回节点信息。
  - Object res。

属性	类型	说明
context	Object	节点对应的 Context 对象

- **返回值：** SelectorQuery 。
- **示例代码：**

```
Page({
  getContext() {
    wx.createSelectorQuery().select('.the-video-class').context(function (res) {
      console.log(res.context) // 节点对应的 Context 对象。如：选中的节点是 <video> 组件，那么此处即返回 VideoContext 对象
    }).exec()
  }
})
```

.node

该 方法 使用方式为 SelectorQuery NodesRef.node(NodesRef.nodeCallback callback)

- **功能说明：**获取 Node 节点实例。目前支持 ScrollViewContext 界面-滚动的获取。 目前支持 Canvas 的获取。
- **参数及说明：**function callback，回调函数，在执行 SelectorQuery.exec 方法后，返回节点信息。
  - Object res

属性	类型	说明
node	Object	节点对应的 Node 实例

- **返回值：**SelectorQuery 。
- **示例代码：**

```
Page({
  getNode() {
    wx.createSelectorQuery().select('.canvas').node(function(res){
      console.log(res.node) // 节点对应的 Canvas 实例。
    }).exec()
  }
})
```

SelectorQuery

.exec

该 方法 使用方式为 NodesRef SelectorQuery.exec(function callback)

- **功能说明：**执行所有的请求。请求结果按请求次序构成数组，在 callback 的第一个参数中返回。
- **参数及说明：**function callback，回调函数。

- **返回值：** NodesRef 。

.in

该 方法 使用方式为 SelectorQuery SelectorQuery.in(Component component)

- **功能说明：** 将选择器的选取范围更改为自定义组件 component 内。（初始时，选择器仅选取页面范围的节点，不会选取任何自定义组件中的节点）。
- **参数及说明：** Component component，自定义组件实例。
- **返回值：** SelectorQuery 。
- **示例代码：**

```
Component({
  queryMultipleNodes() {
    const query = wx.createSelectorQuery().in(this)
    query.select('#the-id').boundingClientRect(function (res) {
      res.top // 这个组件内 #the-id 节点的上边界坐标
    }).exec()
  }
})
```

.select

该 方法 使用方式为 NodesRef SelectorQuery.select(string selector)

- **功能说明：** 在当前页面下选择第一个匹配选择器 selector 的节点。返回一个 NodesRef 对象实例，可以用于获取节点信息。
- **参数及说明：** string selector，选择器。
- **返回值：** NodesRef
- **selector 语法：** selector 类似于 CSS 的选择器，但仅支持下列语法。
  - ID选择器：#the-id。
  - class选择器（可以连续指定多个）：.a-class.another-class。
  - 子元素选择器：.the-parent > .the-child。
  - 后代选择器：.the-ancestor .the-descendant。
  - 跨自定义组件的后代选择器：.the-ancestor >>> .the-descendant。
  - 多选择器的并集：#a-node, .some-other-nodes。

.selectAll

该 方法 使用方式为 NodesRef SelectorQuery.selectAll(string selector)

- **功能说明：**在当前页面下选择匹配选择器 selector 的所有节点。
- **参数及说明：**string selector，选择器。
- **返回值：**NodesRef
- **selector 语法：**selector类似于 CSS 的选择器，但仅支持下列语法。
  - ID选择器：#the-id。
  - class选择器（可以连续指定多个）：.a-class.another-class。
  - 子元素选择器：.the-parent > .the-child。
  - 后代选择器：.the-ancestor .the-descendant。
  - 跨自定义组件的后代选择器：.the-ancestor >>> .the-descendant。
  - 多选择器的并集：#a-node, .some-other-nodes。

.selectViewport

> 该 方法 使用方式为 NodesRef SelectorQuery.selectViewport()

- **功能说明：**选择显示区域。可用于获取显示区域的尺寸、滚动位置等信息。
- **返回值：**NodesRef

MediaQueryObserver

MediaQueryObserver 对象，用于监听页面 media query 状态的变化，如界面的长宽是不是在某个指定的范围内。

.disconnect

> 该 方法 使用方式为 MediaQueryObserver.disconnect()

- **功能说明：**停止监听。回调函数将不再触发。

.observe

> 该 方法 使用方式为 MediaQueryObserver.observe(Object descriptor, function callback)

- **功能说明：**开始监听页面 media query 变化情况。
- **参数及说明：**
  - Object descriptor，media query 描述符。

属性	类型	默认值	必填	说明
minWidth	number	-	是	页面最小宽度（ px 为单位 ）
maxWidth	number	-	是	页面最大宽度（ px 为单位 ）

width	number	-	是	页面宽度（ px 为单位 ）
minHeight	number	-	是	页面最小高度（ px 为单位 ）
maxHeight	number	-	是	页面最大高度（ px 为单位 ）
height	number	-	是	页面高度（ px 为单位 ）
orientation	string	-	是	屏幕方向（ landscape 或 portrait ）

- function callback，监听 media query 状态变化的回调函数。
- Object res

属性	类型	说明
matches	boolean	页面的当前状态是否满足所指定的 media query

自定义API

最近更新时间：2023-07-12 16:22:40

invokeNativePlugin

TMF 小程序 sdk 提供接口，可实现一些开放平台 API 可以直接透传给宿主客户端来实现，如登录、获取用户信息等；一些宿主客户端可以扩展，以保持 UI 和功能的一致性，如扫码，分享等；也可以扩展自定义 API，进行透传宿主客户端来。

小程序端使用方式

```
var opts = {
  api_name: '', // api名称
  success: function(res) {},
  fail: function(res) {},
  complete: function(res) {},
  data: { // 入参
    name : 'kka',
    age : 22,
    data: {...}
  }
}
wx.invokeNativePlugin(opts); // 调用
```

Methods

方法	说明
View onCreateLoadingView()	创建加载视图，允许宿主客户端呈现小程序/小游戏品牌和自定义显示样式，如返回空，则展示默认加载视图。
View onCreateCapsuleView()	创建胶囊按钮，允许宿主自定义胶囊按钮样式，如返回空，则展示默认胶囊按钮。
boolean onShowMenu()	单击 “...” 更多按钮时触发，如果 onCreateCapsuleView 返回空且 onShowMenu 返回 false，则显示默认菜单。
void onExit()	单击 “ ” 退出按钮时触发，如果 onCreateCapsuleView 返回空且 onExit 返回 false，则执行默认操作。

boolean onAuthorize(JSONObject params, Value Callback callback)	wx.authorize 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onOpenSetting(JSONObject params, ValueCallback callback)	wx.openSetting 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onGetSetting(JSONObject params, ValueCallback callback)	wx.getSetting 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onLogin(JSONObject params, ValueCallback callback)	wx.login 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onRefreshSession(JSONObject params, ValueCallback callback)	wx.checkSession 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onRequestPayment(JSONObject params, ValueCallback callback)	wx.requestPayment 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onGetUserInfo(JSONObject params, ValueCallback callback)	wx.getUserInfo 接口实现，如返回 false，则直接通知 wx api 调用失败，详情请参见 onGetUserInfo 。
boolean onShareAppMessage(JSONObject params, ValueCallback callback)	wx.shareAppMessage 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onNavigateToMiniProgram(JSONObject params, ValueCallback callback)	wx.navigateToMiniProgram 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onScanCode(JSONObject params, ValueCallback callback)	wx.scanCode 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onOpenDocument(JSONObject params, ValueCallback callback)	wx.openDocument 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onOpenLocation(JSONObject params, ValueCallback callback)	wx.openLocation 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onChooseLocation(JSONObject params, ValueCallback callback)	wx.chooseLocation 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onPreviewImage(JSONObject params, ValueCallback callback)	wx.rviewImage 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onChooseImage(JSONObject params, ValueCallback callback)	wx.chooseImage 接口实现，如返回 false，则直接通知 wx api 调用失败。
boolean onChooseVideo(JSONObject params,	wx.chooseVideo 接口实现，如返回

ValueCallback callback)	false，则直接通知 wx api 调用失败。
boolean onShowToast(JSONObject params, ValueCallback callback)	wx.showToast 和 wx.showLoading 接口实现，如返回 false，则使用默认样式展示。
boolean onHideToast(JSONObject params, ValueCallback callback)	wx.hideToast 和 wx.hideLoading 接口实现，如返回 false，则使用默认样式展示。
boolean onShowModal(JSONObject params, ValueCallback callback)	wx.showModal 接口实现，如返回 false，则使用默认样式展示。
boolean onInvokeWebAPI(String event, JSONObject params, ValueCallback callback)	实现自定义 wx api，如返回 false，则直接通知 wx api 调用失败，详情请参见 onInvokeWebAPI 。
boolean onReportEvent(String event, Map params)	详情请参见 onReportEvent 。

boolean onGetUserInfo(JSONObject params, ValueCallback callback)

```
-(void)onGetUserInfoWithParams:(NSDictionary *)params inApp:(NSString *)appId
callbackHandler:(WebAPICallbackHandler)handler {
   //TODO
      NSDictionary* userInfo = @{
                           @"nickname" : @"morven",
                           @"headimgurl" :
@"https://wx.qlogo.cn/mmopen/vi_32/Q0j4TwGTfTKav1ib8qG43xy0resTpgfeCqH00vRp
HicEdk0kKMxqTMMUG1WmBuAdgB2tmCf6joGVKlGbsicelhluw/0",
                           @"sex" : @(1),
                           @"province" : @"广东",
                           @"city" : @"深圳",
                           @"country" : @"中国"
                           };
      NSMutableDictionary* result = [NSMutableDictionary dictionaryWithCapacity:1];
      NSMutableDictionary* userInfoDic = [NSMutableDictionary
dictionaryWithCapacity:6];
      NSMutableDictionary* resultDataDic = [NSMutableDictionary
dictionaryWithCapacity:1];
      [userInfoDic setValue:userInfo[@"nickname"] forKey:@"nickName"];
      [userInfoDic setValue:userInfo[@"headimgurl"] forKey:@"avatarUrl"];
      [userInfoDic setValue:userInfo[@"sex"] forKey:@"gender"];
      [userInfoDic setValue:userInfo[@"province"] forKey:@"province"];
      [userInfoDic setValue:userInfo[@"city"] forKey:@"city"];
      [userInfoDic setValue:userInfo[@"country"] forKey:@"country"];
```

```
        NSData *data=[NSJSONSerialization dataWithJSONObject:userInfoDic
options:NSJSONWritingPrettyPrinted error:nil];

        [resultDataDic setValue:[[NSString alloc]initWithData:data
encoding:NSUTF8StringEncoding] forKey:@"data"];
        [result setValue:resultDataDic forKey:@"data"];
        [result setValue:@"getUserInfo:ok" forKey:@"errMsg"];

    if (handler) {
        //success
        handler(result, nil);
    }
}
```

boolean onInvokeWebAPI(String event, JSONObject params, ValueCallback callback)

```
- (BOOL)onInvokeWebAPIWithEvent:(NSString*)event params:(NSDictionary*)params
callbackHandler:(WebAPICallbackHandler _Nullable)handler {
    NSLog(@"onInvokeWebAPIWithEvent, event:%@, params:%@", event, params);
    if (handler) {
        handler(@{@"errMsg" : @"onInvokeWebAPIWithEvent"}, nil);
    }
    return YES;
}
```

boolean onReportEvent(String event, Map<String, String> params)

事件上报，事件如下:
1 启动小程序，event:MS_EVENT_LAUNCH，params 的 key:pagePath,d;
2 小程序使用时长，event:MS_EVENT_USE_TIME，params 的 key:useTime, startId, appId，useTime 的单位是毫秒; 3 成功启动小程序，event:MS_EVENT_LAUNCH_SUCCESS，params 的 key:appId
4 打开小程序内的页面，event:MS_EVENT_OPEN_PAGE，params 的 key:pagePath, appId
5 启动小程序失败，event:MS_EVENT_LAUNCH_FAIL，params 的 key:pagePath, reason, appId

敏感 API

最近更新时间：2023-05-30 21:51:32

> **警告：**
> 本页所列举的 API 均为涉及获取用户信息类的敏感型 API，请开发者阅读下文相关 API 说明后再按需提交申请审批。一旦发现使用方有违法违规行为，平台有权回收权限并进行封号处理，并保留追究法律责任的。

位置

getLocation

- 中文名称：获取当前的地理位置、速度。
- 备注说明：获取当前的地理位置、速度，详情请参见 getLocation 。

choosePoi

- 中文名称：打开POI列表选择位置。
- 备注说明：打开POI列表选择位置，支持模糊定位（精确到市）和精确定位混选，详情请参见 choosePoi 。

chooseLocation

- 中文名称：打开地图选择位置。
- 备注说明：打开地图选择位置，详情请参见 chooseLocation 。

startLocationUpdateBackground

- 中文名称：开启小程序进入前后台时均接收位置消息。
- 备注说明：开启小程序进入前后台时均接收位置消息，需引导用户开启授权，详情请参见 startLocationUpdateBackground 。

startLocationUpdate

- 中文名称：开启小程序进入前台时接收位置消息。
- 备注说明：开启小程序进入前台时接收位置消息，详情请参见 startLocationUpdate 。

onLocationChange

- 中文名称：监听实时地理位置变化事件。
- 备注说明：监听实时地理位置变化事件，需结合 wx.startLocationUpdateBackground、wx.startLocationUpdate 使用，详情请参见 onLocationChange 。

getFuzzyLocation

- 中文名称：获取当前的模糊地理位置。
- 备注说明：获取当前的模糊地理位置，详情请参见 getFuzzyLocation 。

开放接口

startSoterAuthentication

- 中文名称：生物认证。
- 备注说明：开始 SOTER 生物认证（调用系统 API），详情请参见 startSoterAuthentication 。

checkIsSupportSoterAuthentication

- 中文名称：获取生物认证方式。
- 备注说明：获取本机支持的 SOTER 生物认证方式（调用系统 API），详情请参见 checkIsSupportSoterAuthentication 。

checkIsSoterEnrolledInDevice

- 中文名称：调取生物认证接口。
- 备注说明：获取设备内是否录入如指纹等生物信息的接口（调用系统 API），详情请参见 checkIsSoterEnrolledInDevice 。

联系人

chooseContact

- 中文名称：获取通讯录。
- 备注说明：拉起手机通讯录，选择联系人，详情请参见 chooseContact 。

组件层
组件概览

最近更新时间：2024-03-21 15:51:52

视图容器

名称	功能说明	最低版本
scroll-view	可滚动视图区域	1.5.0
swiper	滑块视图容器	1.5.0
swiper-item	仅可放置在 swiper 组件中，宽高自动设置为100%	1.5.0
view	视图容器	1.5.0
movable-area	movable-view 的可移动区域	1.5.0
movable-view	可移动的视图容器，在页面中可以拖拽滑动	1.5.0
cover-image	覆盖在原生组件之上的图片视图	1.5.0
cover-view	覆盖在原生组件之上的文本视图	1.5.0

基础内容

名称	功能说明	最低版本
icon	图标组件	1.5.0
progress	进度条	1.5.0
text	文本	1.5.0
rich-text	富文本	1.5.0

表单组件

名称	功能说明	最低版本
button	按钮	1.5.0

checkbox	多选项目	1.5.0
checkbox-group	多项选择器，内部由多个 checkbox 组成	1.5.0
form	表单	1.5.0
input	输入框	1.5.0
label	用来改进表单组件的可用性	1.5.0
picker	从底部弹起的滚动选择器	1.5.0
radio	单选项目	1.5.0
radio-group	单项选择器，内部由多个 radio 组成	1.5.0
slider	滑动选择器	1.5.0
switch	开关选择器	1.5.0
textarea	多行输入框	1.5.0

导航

名称	功能说明	最低版本
navigator	页面链接	1.5.0

媒体组件

名称	功能说明	
image	图片	1.5.0
video	视频	1.5.0
animation-video	透明视频	1.5.0
camera	系统相机	1.5.0
live-player	实时音视频播放	1.5.0
live-pusher	实时音视频录制	1.5.0

地图

名称	功能说明	最低版本
map	地图	1.5.0

画布

名称	功能说明	最低版本
Canvas	画布	1.5.0

开放能力

名称	功能说明	最低版本
web-view	承载网页的容器	1.5.0

原生组件

最近更新时间：2024-03-21 15:51:52

原生组件

在腾讯云小程序平台开发中，部分组件是由客户端创建的原生组件。这些原生组件是由小程序客户端实现的，具有更高的性能和更丰富的功能。
原生组件包括：

- camera：相机组件，用于调用摄像头拍照或录像。
- Canvas：画布组件，用于绘制图形、动画等。
- input（仅在 focus 时表现为原生组件）
- textarea：多行输入框组件，用于输入多行文本。
- video：视频组件，用于播放视频。
- liver-player：直播播放器组件，用于播放直播流。
- map：地图组件，用于显示地图和地图上的标记、路线等。

原生组件的使用限制

由于原生组件脱离在 WebView 渲染流程外，因此在使用时有以下限制：

- 原生组件的层级是 `最高` 的，所以页面中的其他组件无论设置 `z-index` 为多少，都无法盖在原生组件上。后插入的原生组件可以覆盖之前的原生组件。
- 原生组件无法在 `<picker-view>` 中使用。
- 部分 CSS 样式无法应用于原生组件，例如：
  - 无法对原生组件设置 CSS 动画。
  - 无法定义原生组件为 `position: fixed` 。
  - 不能在父级节点使用 `overflow: hidden` 来裁剪原生组件的显示区域。
- 原生组件的事件监听不能使用 `bind:eventname` 的写法，只支持 `bindeventname` 。原生组件也不支持 `catch` 和 `capture` 的事件绑定方式。
- 原生组件会遮挡 vConsole 弹出的调试面板。

> **说明：**
> 在工具上，原生组件是用 web 组件模拟的，因此很多情况并不能很好的还原真机的表现，建议开发者在使用到原生组件时尽量在真机上进行调试。

基础内容

最近更新时间：2024-05-11 17:21:31

icon

- **功能说明：**图标组件。
- **参数及说明：**

属性	类型	默认值	说明
type	string	-	icon 的类型，有效值：success，success_no_circle，info，warn，waiting，cancel，download，search，clear
size	umber / string	23	icon 的大小，单位默认为 px
color	string	-	icon 的颜色，同 css 的 color
aria-label	string	-	无障碍访问，（属性）元素的额外描述

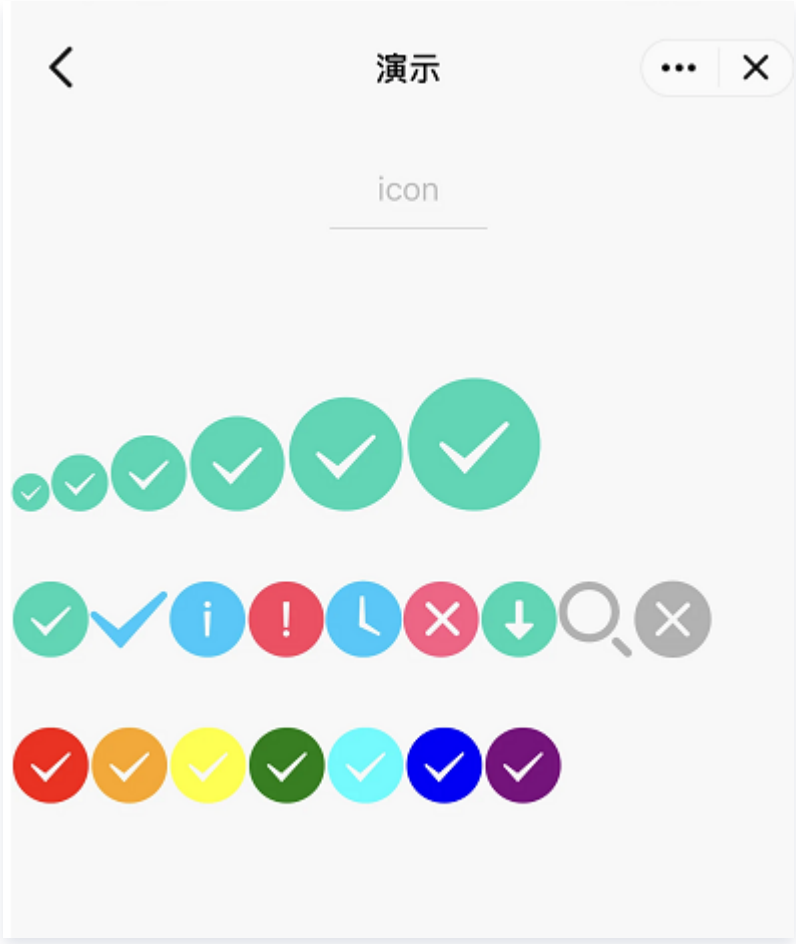
- **示例代码：**

对应的 WXML 文件

```
<view class="group">
  <block wx.for="{{iconSize}}">
    <icon type="success" size="{{item}}" />
  </block>
</view>

<view class="group">
  <block wx.for="{{iconType}}">
    <icon type="{{item}}" size="40" />
  </block>
</view>

<view class="group">
  <block wx.for="{{iconColor}}">
    <icon type="success" size="40" color="{{item}}" />
  </block>
</view>
```

对应的 js 文件

```
Page({
  data: {
    iconSize: [20, 30, 40, 50, 60, 70],
    iconColor: [
      'red', 'orange', 'yellow', 'green', 'rgb(0,255,255)', 'blue', 'purple'
    ],
    iconType: [
      'success', 'success_no_circle', 'info', 'warn', 'waiting', 'cancel', 'download',
'search', 'clear'
    ]
  }
})
```



progress

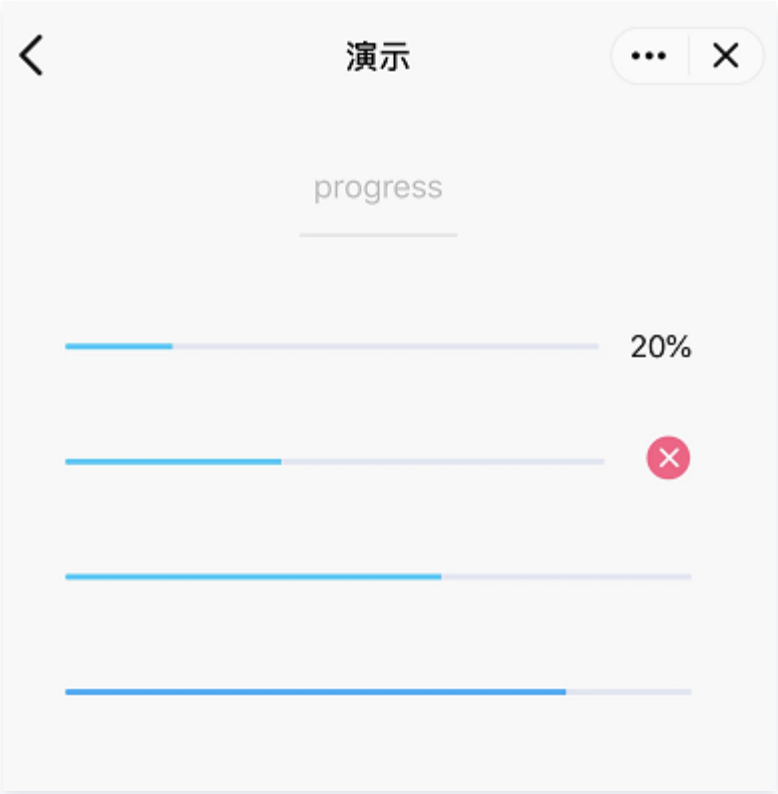
- **功能说明：**进度条。
- **参数及说明：**

属性	类型	默认值	说明

percent	float	无	百分比0~100
show-info	boolean	false	在进度条右侧显示百分比
border-radius	number / string	0	圆角大小，单位：px
font-size	number / string	16	右侧百分比字体大小，单位：px
stroke-width	number / string	6	进度条线的宽度
color	color	#09BB07	进度条颜色（请使用 activeColor）
activeColor	color	-	已选择的进度条的颜色
backgroundColor	color	-	未选择的进度条的颜色
active	boolean	false	进度条从左往右的动画
active-mode	string	backwards	• backwards：动画从头播 • forwards：动画从上次结束点接着播
bindactiveend	eventhandle	-	动画完成事件
aria-label	string	-	无障碍访问，（属性）元素的额外描述

- **示例代码：**

```
<progress percent="20" show-info />
<progress percent="40" stroke-width="12" />
<progress percent="60" color="blue" />
<progress percent="80" active />
```



text

- **功能说明：**文本。
- **参数及说明：**

属性	类型	默认值	说明
selectable	boolean	false	文本是否可选
space	string	-	显示连续空格
decode	boolean	false	是否解码

值	说明
ensp	中文字符空格一半大小
emsp	中文字符空格大小
nbsp	根据字体设置的空格大小

说明：

- decode 可以解析的有   < > & '   。
- 各个操作系统的空格标准并不一致。
- <text> 组件内只支持 <text> 嵌套。
- 除了文本节点以外的其他节点都无法长按选中。

对应的 WXML 文件

```
<view class="btn-area">
  <view class="body-view">
    <text>{{text}}</text>
    <button bindtap="add">add line</button>
    <button bindtap="remove">remove line</button>
  </view>
</view>
```

对应的 js 文件

```
const initData = 'this is first line\nthis is second line'
const extraLine = []
Page({
  data: {
    text: initData
  },
  add(e) {
    extraLine.push('other line')
    this.setData({
      text: initData + '\n' + extraLine.join('\n')
    })
  },
  remove(e) {
    if (extraLine.length > 0) {
      extraLine.pop()
      this.setData({
        text: initData + '\n' + extraLine.join('\n')
      })
    }
  }
})
```

rich-text

- **功能说明：**富文本。
- **参数及说明：**

属性	类型	默认值	说明
nodes	array / string	–	节点列表 / HTML String
space	string	–	显示连续空格

- **space有效值：**

值	说明
ensp	中文字符空格一半大小
emsp	中文字符空格大小
nbsp	根据字体设置空格大小

支持默认事件，包括： `tap` 、 `touchstart` 、 `touchmove` 、 `touchcancel` 、 `touchend` 和 `longtap` 。

- **nodes**

nodes 属性推荐使用 Array 类型，由于组件会将 String 类型转换为 Array 类型，因而性能会有所下降。现支持两种节点，通过 type 来区分，分别是元素节点和文本节点，默认是元素节点，在富文本区域里显示的 HTML 节点。

元素节点：type = node

属性	说明	类型	必填	备注
name	标签名	string	是	支持部分受信任的 HTML 节点
attrs	属性	object	否	支持部分收信人的属性，遵循 Pascal 命名法
children	子节点列表	array	否	结构和 nodes 一致

- **文本节点：type=text**

属性	说明	类型	必填	备注
text	文本	string	是	支持 entities

- **受信任的 HTML 节点及属性**

全局支持 class 和 style 属性，**不支持 id 属性。**

节点	属性	节点	属性
a	-	img	alt,src,height,width
abbr	-	ins	-
b	-	label	-
br	-	legend	-
code	-	li	-
col	span,width	ol	start,type
colgroup	span,width	p	-
dd	-	q	-
del	-	span	-
div	-	strong	-
dl	-	sub	-

dt	–	sup	–
em	–	table	width
fieldset	–	tbody	–
h1	–	td	colspan,height,rowspan,width
h2	–	tfoot	–
h3	–	th	colspan,height,rowspan,width
h4	–	thead	–
h5	–	tr	–
h6	–	ul	–
hr	–	–	–

- **示例代码**

```
<!-- rich-text.wxml -->
<rich-text nodes="{{nodes}}" bindtap="tap"></rich-text>
```

```
// rich-text.js
Page({
  data: {
    nodes: [{
      name: 'div',
      attrs: {
        class: 'div_class',
        style: 'line-height: 60px; color: red;'
      },
      children: [{
        type: 'text',
        text: 'Hello World!'
      }]
    }]
  },
  tap() {
    console.log('tap')
  }
```

```
})
```

说明：

- nodes 不推荐使用 String 类型，性能会有所下降。
- rich-text 组件内屏蔽所有节点的事件。
- attrs 属性不支持 id，支持 class。
- name 属性大小写不敏感。
- 如果使用了不受信任的 HTML 节点，该节点及其所有子节点将会被移除。
- img 标签仅支持网络图片。
- 如果在自定义组件中使用 rich-text 组件，那么仅自定义组件的 wxss 样式对 rich-text 中的 class 生效。`nodes` 不推荐使用 String 类型，性能会有所下降。

视图容器

最近更新时间：2023-05-31 15:17:32

scroll-view

- **功能说明：**可滚动视图区域。使用竖向滚动时，需要给 `<scroll-view>` 一个固定高度，通过 WXSS 设置 height。
- **参数及说明：**

属性	类型	默认值	说明
scroll-x	boolean	false	允许横向滚动
scroll-y	boolean	false	允许纵向滚动
upper-threshold	number/string	50	距顶部/左边多远时，触发 scrolltoupper
lower-threshold	number/tring	50	距底部/右边多远时，触发 scrolltolower 事件
scroll-top	number/string	–	设置竖向滚动条位置
scroll-left	number/string	–	设置横向滚动条位置
scroll-into-view	string	–	值应为某子元素 id（id 不能以数字开头）。设置哪个方向可滚动，则在哪个方向滚动到该元素
scroll-with-animation	boolean	false	在设置滚动条位置时使用动画过渡
enable-back-to-top	boolean	false	iOS 单击顶部状态栏、Android 双击标题栏时，滚动条返回顶部，只支持竖向
bindscrolltoupper	eventhandle	–	滚动到顶部/左边时触发
bindscrolltolower	eventhandle	–	滚动到底部/右边时触发
bindscroll	eventhan	–	滚动时触发，event.detail = {scrollLeft,

	dle		scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}
aria-label	string	false	-
enhanced	boolean	true	-
bounces	boolean	true	-
show-scrollbar	boolean	false	-
paging-enabled	boolean	false	-
fast-deceleration	boolean	false	-

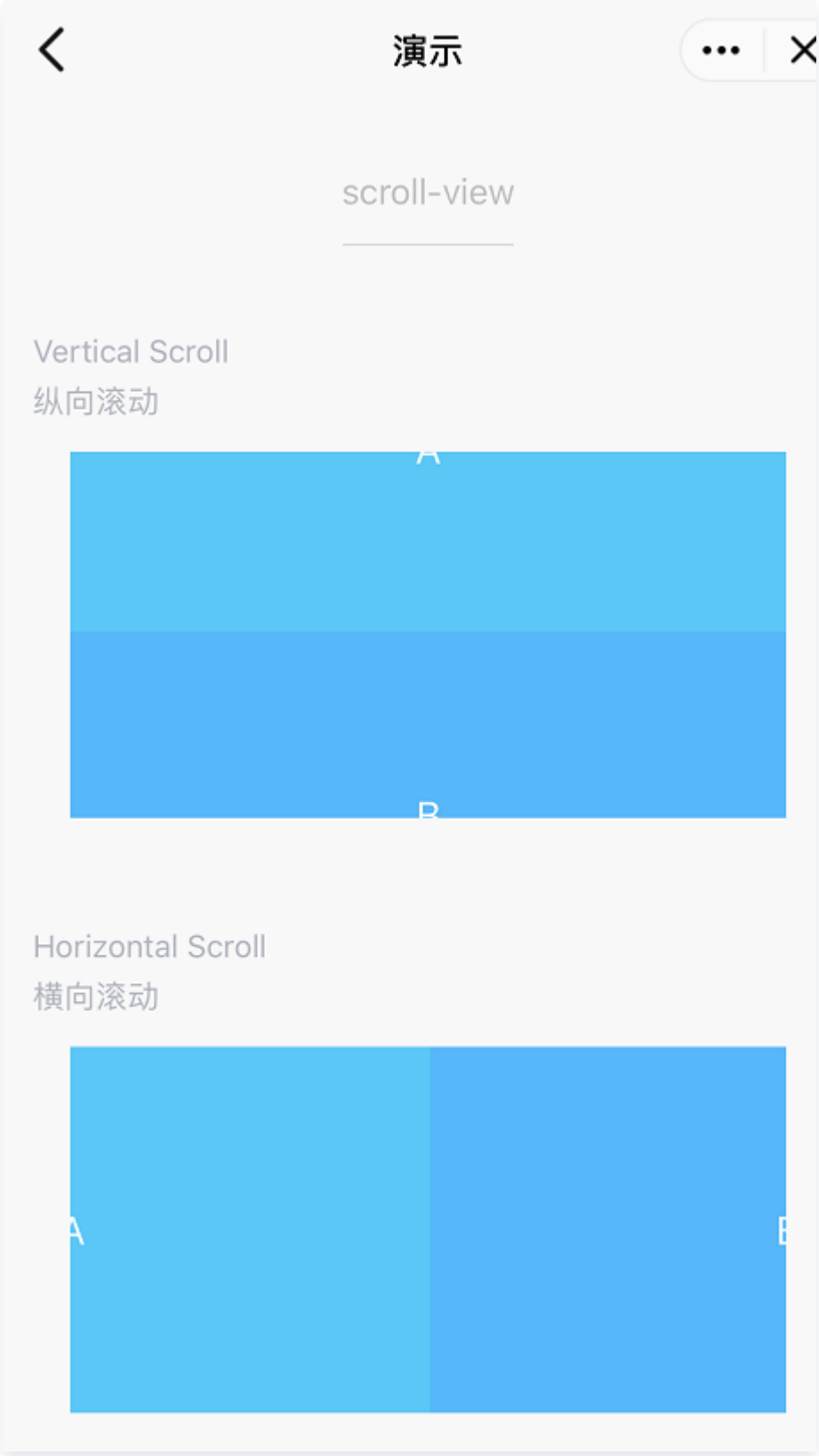
- **示例代码：**

  对应的 WXML 文件

```
<view class="section">
  <view class="section__title">vertical scroll</view>
  <view class="section__title">纵向滚动</view>
  <scroll-view
    scroll-y
    style="height: 200px;"
    bindscrolltoupper="upper"
    bindscrolltolower="lower"
    bindscroll="scroll"
    scroll-into-view="{{toView}}"
    scroll-top="{{scrollTop}}"
  >
    <view id="green" class="scroll-view-item bc_cyanblue">A</view>
    <view id="red" class="scroll-view-item bc_blue">B</view>
  </scroll-view>
</view>
<view class="section section_gap">
  <view class="section__title">horizontal scroll</view>
  <view class="section__title">横向滚动</view>
  <scroll-view class="scroll-view_H" scroll-x style="width: 100%">
    <view id="green" class="scroll-view-item_H bc_cyanblue"></view>
    <view id="red" class="scroll-view-item_H bc_blue"></view>
  </scroll-view>
</view>
```

对应的 js 文件

```
const order = ['red', 'yellow', 'blue', 'green', 'red']
Page({
  data: {
    toView: 'red',
    scrollTop: 100
  },
  upper(e) {
    console.log(e)
  },
  lower(e) {
    console.log(e)
  },
  scroll(e) {
    console.log(e)
  },
  tap(e) {
    for (let i = 0; i < order.length; ++i) {
      if (order[i] === this.data.toView) {
        this.setData({
          toView: order[i + 1]
        })
        break
      }
    }
  },
  tapMove(e) {
    this.setData({
      scrollTop: this.data.scrollTop + 10
    })
  }
})
```



说明：

- 请勿在 scroll-view 中使用 textarea、map、canvas、video 组件。
- scroll-into-view 的优先级高于 scroll-top。

- 在滚动 scroll-view 时会阻止页面回弹，所以在 scroll-view 中滚动，是无法触发 onPullDownRefresh。
- 若要使用下拉刷新，请使用页面的滚动，而不是 scroll-view ，这样也能通过点击顶部状态栏回到页面顶部。

swiper

- **功能说明：**滑块视图容器。`change` 事件返回 `detail` 中包含一个 `source` 字段，表示导致变更的原因，可能值如下：
  - `autoplay` 自动播放导致 swiper 变化；
  - `touch` 用户滑动引起 swiper 变化；
  - 其他原因将用空字符串表示。

> ⚠ **注意：**
> 其中只可放置 `<swiper-item/>` 组件，否则会导致未定义的行为。

- **参数及说明：**

属性	类型	默认值	说明
indicator-dots	boolean	false	是否显示面板指示点
indicator-color	color	rgba(0, 0, 0, .3)	指示点颜色
indicator-active-color	color	#000000	当前选中的指示点颜色
autoplay	coolean	false	是否自动切换
current	number	0	当前所在滑块的 index
interval	number	5000	自动切换时间间隔
duration	number	500	滑动动画时长
circular	boolean	false	是否采用衔接滑动

vertical	boolean	false	滑动方向是否为纵向
previous-margin	string	"0px"	前边距，可用于露出前一项的一小部分，接受 px 和 rpx 值
next-margin	string	"0px"	后边距，可用于露出后一项的一小部分，接受 px 和 rpx 值
display-multiple-items	number	1	同时显示的滑块数量
skip-hidden-item-layout	boolean	false	是否跳过未显示的滑块布局，设为 true 可优化复杂情况下的滑动性能，但会丢失隐藏状态滑块的布局信息
bindchange	eventhandle	-	current 改变时会触发 change 事件，event.detail = {current: current, source: source}
bindtransition	eventhandle	-	swiper-item 的位置发生改变时会触发 transition 事件，event.detail = {dx: dx, dy: dy}
bindanimationfinish	eventhandle	-	动画结束时会触发 animationfinish 事件，event.detail 同上

swiper-item

- **功能说明：**仅可放置在 `<swiper>` 组件中，宽高自动设置为100%。
- **参数及说明：**

属性名	类型	默认值	说明
item-id	string	“”	该 swiper-item 的标识符

- **示例代码：**

  对应的 WXML 文件

```
<swiper
  indicator-dots="{{indicatorDots}}"
  autoplay="{{autoplay}}"
  interval="{{interval}}"
  duration="{{duration}}"
>
```

```
  <block wx.for="{{imgUrls}}">
    <swiper-item>
      <image src="{{item}}" class="slide-image" width="355" height="150" />
    </swiper-item>
  </block>
</swiper>
<button bindtap="changeIndicatorDots">indicator-dots</button>
<button bindtap="changeAutoplay">autoplay</button>
<slider bindchange="intervalChange" show-value min="500" max="2000" />
interval
<slider bindchange="durationChange" show-value min="1000" max="10000" />
duration
```

对应的 js 文件

```
Page({
  data: {
    imgUrls: [
      'https://img02.tooopen.com/images/20150928/tooopen_sy_143912755726.jpg',
      'https://img06.tooopen.com/images/20160818/tooopen_sy_175866434296.jpg',
      'https://img06.tooopen.com/images/20160818/tooopen_sy_175833047715.jpg'
    ],
    indicatorDots: false,
    autoplay: false,
    interval: 5000,
    duration: 1000
  },
  changeIndicatorDots(e) {
    this.setData({
      indicatorDots: !this.data.indicatorDots
    })
  },
  changeAutoplay(e) {
    this.setData({
      autoplay: !this.data.autoplay
    })
  },
  intervalChange(e) {
    this.setData({
      interval: e.detail.value
    })
  },
  durationChange(e) {
    this.setData({
      duration: e.detail.value
```

```
  })
 }
})
```

view

- **功能描述：**视图容器。
- **参数及说明：**

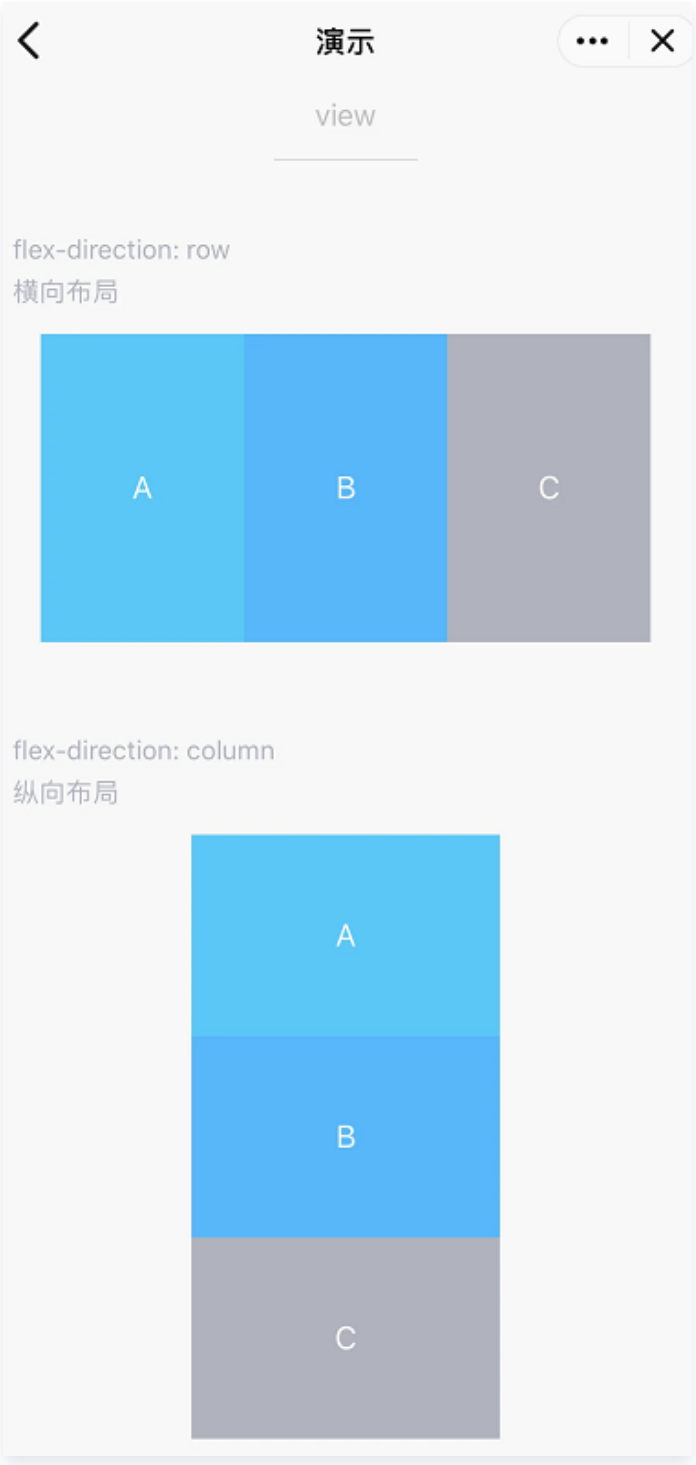
属性名	类型	默认值	说明
hover-class	string	none	指定按下去的样式类。当 hover-class="none" 时，没有点击态效果
hover-stop-propagation	boolean	false	指定是否阻止本节点的祖先节点出现点击态
hover-start-time	number	50	按住后多久出现点击态，单位毫秒
hover-start-time	number	400	手指松开后点击态保留时间，单位毫秒
aria-role	string	-	无障碍访问，（角色）标识元素的作用
aria-label	string	-	无障碍访问，（属性）元素的额外描述

- **示例代码：**

  对应的 WXML 文件

```
<view class="section">
 <view class="section__title">flex-direction: row</view>
 <view class="flex-wrp" style="flex-direction:row;">
  <view class="flex-item bc_cyanblue">A</view>
  <view class="flex-item bc_blue">B</view>
  <view class="flex-item bc_grey">C</view>
 </view>
</view>
<view class="section">
 <view class="section__title">flex-direction: column</view>
 <view class="flex-wrp" style="height: 300px;flex-direction:column;">
  <view class="flex-item bc_cyanblue">A</view>
  <view class="flex-item bc_blue">B</view>
  <view class="flex-item bc_grey">C</view>
 </view>
```

```
</view>
```



> **说明：**
> 如果需要使用滚动视图，请使用 scroll-view 。

movable-area

- **功能说明：** movable-view 的可移动区域。

- **参数及说明：**

属性名	类型	默认值	说明
scale-area	boolean	false	当里面的 movable-view 设置为支持双指缩放时，设置此值可将缩放手势生效区域修改为整个movable-area。

> **注意：**
> 当 movable-view 小于 movable-area 时，movable-view 的移动范围是在 movable-area 内。
> 当 movable-view 大于 movable-area 时，movable-view 的移动范围必须包含 movable-area（x 轴方向和 y 轴方向分开考虑）。

- **示例代码：**

对应的 WXML 文件

```
<view class="section">
<view class="section__title">movable-view区域小于movable-area</view>
<movable-area style="height: 200px; width: 200px; background: red;">
 <movable-view
  style="height: 50px; width: 50px; background: blue;"
  x="{{x}}"
  y="{{y}}"
  direction="all"
 ></movable-view>
</movable-area>
<view class="btn-area">
 <button size="mini" bindtap="tap">click me to move to (30px, 30px)
</button>
</view>
<view class="section__title">movable-view区域大于movable-area</view>
<movable-area style="height: 100px; width: 100px; background: red;">
 <movable-view
  style="height: 200px; width: 200px; background: blue;"
  direction="all"
 ></movable-view>
</movable-area>
<view class="section__title">可放缩</view>
<movable-area
 style="height: 200px; width: 200px; background: red;"
 scale-area
>
 <movable-view
```

```
      style="height: 50px; width: 50px; background: blue;"
      direction="all"
      bindchange="onChange"
      bindscale="onScale"
      scale
      scale-min="0.5"
      scale-max="4"
      scale-value="2"
    ></movable-view>
  </movable-area>
</view>
```

对应的 js 文件

```
Page({
  data: {
    x: 0,
    y: 0
  },
  tap(e) {
    this.setData({
      x: 30,
      y: 30
    })
  },
  onChange(e) {
    console.log(e.detail)
  },
  onScale(e) {
    console.log(e.detail)
  }
})
```

movable-view

- **功能说明：**可移动的视图容器，在页面中可以拖拽滑动。
- **参数及说明：**

属性名	类型	默认值	说明
direction	string	none	movable-view 的移动方向，属性值有 all、vertical、horizontal、none
inertia	boolean	false	movable-view 是否带有惯性

out-of-bounds	boolean	false	超过可移动区域后，movable-view 是否还可以移动
x	number / string	–	定义 x 轴方向的偏移，如果 x 的值不在可移动范围内，会自动移动到可移动范围；改变 x 的值会触发动画
y	number / string	–	定义 y 轴方向的偏移，如果 y 的值不在可移动范围内，会自动移动到可移动范围；改变 y 的值会触发动画
damping	number	20	阻尼系数，用于控制 x 或 y 改变时的动画和过界回弹的动画，值越大移动越快
friction	number	2	摩擦系数，用于控制惯性滑动的动画，值越大摩擦力越大，滑动越快停止；必须大于0，否则会被设置成默认值
disabled	boolean	false	是否禁用
scale	boolean	false	是否支持双指缩放，默认缩放手势生效区域是在 movable-view 内
scale-min	number	0.5	定义缩放倍数最小值
scale-max	number	10	定义缩放倍数最大值
scale-value	number	1	定义缩放倍数，取值范围为0.5 - 10
animation	boolean	true	定义缩放倍数，取值范围为0.5 - 10
bindchange	eventhandle	–	拖动过程中触发的事件，event.detail = {x: x, y: y, source: source}，其中 source 表示产生移动的原因，值可为 touch（拖动）、touch-out-of-bounds（超出移动范围）、out-of-bounds（超出移动范围后的回弹）、friction（惯性）和空字符串（setData）
bindscale	eventhandle	–	缩放过程中触发的事件，event.detail = {x: x, y: y, scale: scale}

除了基本事件外，movable-view 提供了两个特殊事件。

类型	触发条件
htouchmo	初次手指触摸后移动为横向的移动，如果 catch 此事件，则意味着 touchmove 事件

ve	也被 catch
vtouchmove	初次手指触摸后移动为纵向的移动，如果 catch 此事件，则意味着 touchmove 事件也被 catch

说明：

- movable-view 必须设置 width 和 height 属性，不设置默认为10px。
- movable-view 默认为绝对定位，top 和 left 属性为0px。
- movable-view 必须在 <movable-area/> 组件中，并且必须是直接子节点，否则不能移动。

cover-image

- **功能说明：**覆盖在原生组件之上的图片视图，可覆盖的原生组件同 cover-view，支持嵌套在 cover-view 里。
- **参数及说明：**

属性名	类型	默认值	说明
src	string	–	图标路径，支持临时路径、网络地址、云文件 ID。暂不支持 base64 格式
bindload	eventhandle	–	图片加载失败时触发
binderror	eventhandle	–	图片加载失败时触发
aria-role	string	–	无障碍访问，（角色）标识元素的作用
aria-lable	string	–	无障碍访问，（角色）标识元素的作用

说明：

- <cover-view> 和 <cover-image> 的 aria-role 仅可设置为 button，读屏模式下才可以单击，并朗读出“按钮”；为空时可以聚焦，但不可单击。
- 事件模型遵循冒泡模型，但不会冒泡到原生组件。
- 文本建议都套上 cover-view 标签，避免排版错误。
- 只支持基本的定位、布局、文本样式。不支持设置单边的 border、background-image、shadow、overflow: visible 等。
- 建议子节点不要溢出父节点。
- 默认设置的样式有：white-space: nowrap; line-height: 1.2; display: block。
- 自定义组件嵌套 cover-view 时，自定义组件的 slot 及其父节点暂不支持通过 wx:if 控制显隐，

> 否则会导致 cover-view 不显示。

- **示例代码：**

对应的 WXML 文件

```
<video
  id="myVideo"
  src="https://qzonestyle.gtimg.cn/qzone/qzact/act/external/qq-video/qq-video.mp4"
  controls="{{false}}"
  event-model="bubble"
>
  <cover-view class="controls">
    <cover-view class="play" bindtap="play">
      <cover-image class="img" src="/path/to/icon_play" />
    </cover-view>
    <cover-view class="pause" bindtap="pause">
      <cover-image class="img" src="/path/to/icon_pause" />
    </cover-view>
    <cover-view class="time">00:00</cover-view>
  </cover-view>
</video>
```

对应的 WXSS 文件

```
.controls {
  position: relative;
  top: 50%;
  height: 50px;
  margin-top: -25px;
  display: flex;
}
.play,
.pause,
.time {
  flex: 1;
  height: 100%;
}
.time {
  text-align: center;
  background-color: rgba(0, 0, 0, 0.5);
  color: white;
  line-height: 50px;
```

```
}
.img {
  width: 40px;
  height: 40px;
  margin: 5px auto;
}
```

对应的 js 文件

```
Page({
  onReady() {
    this.videoCtx = wx.createVideoContext('myVideo')
  },
  play() {
    this.videoCtx.play()
  },
  pause() {
    this.videoCtx.pause()
  }
})
```

cover-view

- **功能说明：覆盖在原生组件之上的文本视图，可覆盖的原生组件包括** `map` 、 `video` 、 `canvas` 、 `camera` 、 `live-player` 、 `live-pusher` **，只支持嵌套** `cover-view` 、 `cover-image` **，可在** `cover-view` **中使用** `button` 。
- **参数及说明：**

属性名	类型	默认值	说明
scroll-top	number / string	–	设置顶部滚动偏移量，仅在设置了 overflow-y: scroll 成为滚动元素后生效
aria-role	string	–	无障碍访问，（角色）标识元素的作用
aria-label	string	–	无障碍访问，（属性）元素的额外描述

表单组件

最近更新时间：2024-03-21 15:51:52

button

- **功能说明：**按钮。
- **参数及说明：**

属性	类型	合法值及说明	默认值	必填	说明
size	string	default：默认大小 mini：小尺寸	default	否	按钮的大小
type	string	primary：绿色 default：白色 warn：红色	default	否	按钮的样式类型
plain	boolean	-	false	否	按钮是否镂空，背景色透明
disabled	boolean	-	false	否	是否禁用
loading	boolean	-	false	否	名称前是否带 loading 图标
form-type	string	submit：提交表单 reset：重置表单	-	否	用于 form 组件，单击分别会触发 form 组件的 submit/reset 事件
hover-class	string	-	button-hover	否	指定按钮按下去的样式类。当 `hover-class="none"` 时，没有单击态效果

hover-start-time	number	-	20	否	按住后多久出现单击态，单位毫秒
hover-stay-time	number	-	70	否	手指松开后单击态保留时间，单位毫秒

checkbox

- **功能说明：**多选项目。
- **参数及说明：**

属性	类型	默认值	必填	说明
value	string	-	否	checkbox 标识，选中时触发 checkbox-group 的 change 事件，并携带 checkbox 的 value
disabled	boolean	false	否	是否禁用
checked	boolean	false	否	当前是否选中，可用来设置默认选中
color	string	#09BB07	否	checkbox 的颜色，同 css 的 color

checkbox-group

- **功能描述：**多项选择器，内部由多个 checkbox 组成。
- **属性说明：**

属性	类型	默认值	必填	说明
bindchange	event handle	-	否	checkbox-group 中选中项发生改变时触发 change 事件，detail = {value:[选中的 checkbox 的 value 的数组]}

form

功能说明：表单，将组件内的用户输入的 switch、input、checkbox 、slider、radio、picker 提交。

当单击 form 表单中 form-type 为 submit 的 button 组件时，会将表单组件中的 value 值进行提交，需要在表单组件中加上 name 来作为 key。

- **参数及说明**

属性	类型	默认值	必填	说明
bindsubmit	event handle	-	否	携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'value'} , formId: ''}
bindreset	event handle	-	否	表单重置时会触发 reset 事件

input

- **功能说明：**输入框。该组件是 原生组件，使用时请注意相关限制。
- **参数及说明**

属性	类型	合法值及说明	默认值	必填	说明
value	string	-	-	是	输入框的初始内容
type	string	text：文本输入键盘 number：数字输入键盘 idcard：身份证输入键盘 digit：带小数点的数字键盘	text	否	input 的类型
password	boolean	-	false	否	是否是密码类型
placeholder	string	-		是	输入框为空时占位符
placeholder-style	string	-	-	是	指定 placeholder 的样式
placehol	string	-	input	否	指定 placeholder 的样式类

der-class			-placeholder		
disabled	boolean	-	false	否	是否禁用
maxlength	number	-	140	否	最大输入长度，设置为 -1 的时候不限制最大长度
cursor-spacing	number	-	0	否	指定光标与键盘的距离，取 input 距离底部的距离和 cursor-spacing 指定的距离的最小值作为光标与键盘的距离
auto-focus	boolean	-	false	否	(即将废弃，请直接使用 focus)自动聚焦，拉起键盘
focus	boolean		false	否	获取焦点
bindinput	eventhandle	-	-	是	键盘输入时触发，event.detail = {value, cursor, keyCode}，keyCode 为键值，2.1.0起支持，处理函数可以直接 return 一个字符串，将替换输入框的内容
bindfocus	eventhandle	-	-	是	输入框聚焦时触发，event.detail = { value, height }，height 为键盘高度，在基础库 1.9.90 起支持
bindblur	eventhandle	-	-	是	输入框失去焦点时触发，event.detail = { value, encryptedValue, encryptError }
bindconfirm	eventhandle	-	-	是	单击完成时触发，event.detail = { value }

label

- **功能说明：**用来改进表单组件的可用性。

  使用 for 属性找到对应的 id，或者将控件放在该标签下，当单击时，就会触发对应的控件。 for 优先级高于内部控件，内部有多个控件的时候默认触发第一个控件。目前可以绑定的控件有：button，checkbox，radio，switch，input。

- **参数及说明：**

属性	类型	默认值	必填	说明
for	string	–	否	绑定控件的 id

picker

- **功能描述：**从底部弹起的滚动选择器。
- **属性说明：**

属性	类型	合法值及说明	默认值	必填	说明
mode	string	selector：普通选择器 multiSelector：多列选择器 time：时间选择器 date：日期选择器 region：省市区选择器	selector	否	选择器类型
disabled	boolean	–	false	否	是否禁用

除了上述通用的属性，对于不同的 mode， `picker` 拥有不同的属性。

普通选择器：mode = selector

属性名	类型	默认值	说明
range	array/object array	[]	mode 为 selector 或 multiSelector 时，range 有效
range-key	string	–	当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	number	0	表示选择了 range 中的第几个（下标从0开始）
bindchange	eventhandle	–	value 改变时触发 change 事件，event.detail = {value}

多列选择器：mode = multiSelector

属性名	类型	默认值	说明

range	array/object array	[]	mode 为 selector 或 multiSelector 时，range 有效
range-key	string	–	当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	array	[]	表示选择了 range 中的第几个（下标从0开始）
bindchange	eventhandle	–	value 改变时触发 change 事件，event.detail = {value}
bindcolumnchange	eventhandle	–	列改变时触发

时间选择器：mode = time

属性名	类型	默认值	说明
value	string	–	表示选中的时间，格式为"hh:mm"
start	string	–	表示有效时间范围的开始，字符串格式为"hh:mm"
end	string	–	表示有效时间范围的结束，字符串格式为"hh:mm"
bindchange	eventhandle	–	value 改变时触发 change 事件，event.detail = {value}

日期选择器：mode = date

属性名	类型	有效值及说明	默认值	说明
value	string	–	当天	表示选中的日期，格式为"YYYY-MM-DD"
start	string	–	–	表示有效日期范围的开始，字符串格式为"YYYY-MM-DD"
end	string	–	–	表示有效日期范围的结束，字符串格式为"YYYY-MM-DD"
fields	string	year：选择器粒度为年	da	有效值 year,month,day，表示选择器的粒度

		month：选择器粒度为月份 day：选择器粒度为天	y	
bindchange	eventhandle	-	-	value 改变时触发 change 事件，event.detail = {value}

省市区选择器：mode = region

属性名	类型	默认值	说明
value	Array	[]	表示选中的省市区，默认选中每一列的第一个值
custom-item	string	-	可为每一列的顶部添加一个自定义的项
bindchange	evenhandle	-	value 改变时触发 change 事件，event.detail = {value: value, code: code, postcode: postcode}，其中字段 code 是统计用区划代码，postcode 是邮政编码
bindcancel	evenhandle	-	取消选择时触发
disabled	boolean	false	是否禁用

radio

- **功能说明：**单选项目。
- **参数及说明**

属性	类型	默认值	必填	说明
value	string	-	否	radio 标识。当该 radio 选中时，radio-group 的 change 事件会携带 radio 的 value
checked	boolean	false	否	当前是否选中
disabled	boolean	false	否	是否禁用

color	string	#09BB07	否	radio 的颜色，同 css 的 color

radio-group

- **功能说明：**单项选择器，内部由多个 radio 组成。
- **参数及说明**

属性	类型	默认值	必填	说明
bindchange	eventhandle	-	否	radio-group 中选中项发生改变时触发 change 事件，detail = {value:[选中的 radio 的 value 的数组]}

slider

- **功能说明：**滑动选择器。
- **参数及说明**

属性	类型	默认值	必填	说明
min	number	0	否	最小值
max	number	100	否	最大值
step	number	1	否	步长，取值必须大于 0，并且可被(max - min)整除
disabled	boolean	false	否	是否禁用
value	number	0	否	当前取值
color	color	#e9e9e9	否	背景条的颜色（请使用 backgroundColor）
selected-color	color	#1aad19	否	已选择的颜色（请使用 activeColor）
activeColor	color	#1aad19	否	已选择的颜色

backgroundColor	color	#e9e9e9	否	背景条的颜色
show-value	boolean	false	否	是否显示当前 value
bindchange	eventhandle	-	否	完成一次拖动后触发的事件，event.detail = {value}
block-size	number	28	否	滑块的大小，取值范围为 12 - 28
block-color	color	#ffffff	否	滑块的颜色

switch

- **功能说明：**开关选择器。
- **参数及说明**

属性	类型	默认值	必填	说明
checked	boolean	false	否	是否选中
disabled	boolean	false	否	是否禁用
type	string	switch	否	样式，有效值：switch，checkbox
color	string	#04BE02	否	switch 的颜色，同 css 的 color
bindchange	event handle	-	否	点击导致 checked 改变时会触发 change 事件，event.detail={ value}

textarea

- **功能说明：**多行输入框。该组件是 原生组件 ，使用时请注意相关限制。
- **参数及说明：**

属性	类型	合法值及说明	默认值	必填	说明

value	string	–	–	否	输入框的内容
placeholder	string	–	–	否	输入框为空时占位符
placeholder-style	string	–	–	否	指定 placeholder 的样式，目前仅支持 color，font-size 和 font-weight
placeholder-class	string	–	textarea-placeholder	否	指定 placeholder 的样式类
disabled	boolean	–	false	否	是否禁用
maxlength	number	–	140	否	最大输入长度，设置为 -1 的时候不限制最大长度
auto-focus	boolean	–	false	否	自动聚焦，拉起键盘。
focus	boolean	–	false	否	获取焦点
auto-height	boolean	–	false	否	是否自动增高，设置 auto-height 时，style.height 不生效
cursor	number	–	-1	否	指定 focus 时的光标位置
selection-start	number	–	-1	否	光标起始位置，自动聚集时有效，需与 `selection-end` 搭配使用
selection-end	number	–	-1	否	光标结束位置，自动聚集时有效，需与 `selection-start` 搭配使用
confirm-type	string	send：右下角按钮为“发送” search：右下角按钮	return	否	设置键盘右下角按钮的文字

		为“搜索” next：右下角按钮为“下一个” go：右下角按钮为“前往” done：右下角按钮为“完成” return：右下角按钮为“换行”			
confirm-hold	boolean	-	false	否	点击键盘右下角按钮时是否保持键盘不收起

导航组件

最近更新时间：2024-05-11 17:21:31

navigator

- **功能说明：**页面链接。
- **参数及说明**

属性	类型	默认值	必填	说明
target	string	self	–	在哪个目标上发生跳转，默认当前小程序，可选值 self/miniProgram
url	string	–	否	当前小程序内的跳转
open-type	string	navigate	否	跳转方式
path	string	–	–	当 target="miniProgram"时有效，打开的页面路径，如果为空则打开首页
extra-data	object	–	–	当 target="miniProgram"时有效，需要传递给目标小程序的数据，目标小程序可在 `App.onLaunch()`，`App.onShow()` 中获取到这份数据。
hover-class	string	navigator-hover	–	指定点击时的样式类，当 `hover-class="none"` 时，没有点击态效果
hover-stop-propagation	boolean	false	–	指定是否阻止本节点的祖先节点出现点击态
hover-start-time	number	50	–	按住后多久出现点击态，单位毫秒
hover-stay-time	number	600	–	手指松开后点击态保留时间，单位毫秒
bindsuccess	stri	–	–	当 target="miniProgram" 时有效，跳转小程序

	ng			成功
bindfail	string	–	–	当 target="miniProgram" 时有效，跳转小程序失败
bindcomplete	string	–	–	当 target="miniProgram" 时有效，跳转小程序完成
aria-label	string	–	–	无障碍访问，（属性）元素的额外描述

- **open-type有效值：**

合法值	说明
navigate	对应 wx.navigateTo 或 wx.navigateToMiniProgram 的功能
redirect	对应 wx.redirectTo 的功能
switchTab	对应 wx.switchTab 的功能
reLaunch	对应 wx.reLaunch 的功能
navigateBack	对应 wx.navigateBack 的功能

- **使用限制：需要用户确认跳转**

  在跳转至其他小程序前，将统一增加弹窗，询问是否跳转，用户确认后才可以跳转至其他小程序。如果用户单击取消，则回调 `fail cancel` 。

  > ⚠ **注意：**
  > `navigator-hover` **默认为** `{background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;},<navigator>` **的子节点背景色应为透明色。**

- **示例代码：**

```
/** wxss **/
/** 修改默认的navigator点击态 **/
.navigator-hover {
  color: blue;
```

```
}
/** 自定义其他点击态样式类 **/
.other-navigator-hover {
  color: red;
}
```

```
<!-- sample.wxml -->
<view class="btn-area">
  <navigator
    url="/page/navigate/navigate?title=navigate"
    hover-class="navigator-hover"
  >
    跳转到新页面
  </navigator>
  <navigator
    url="../../redirect/redirect/redirect?title=redirect"
    open-type="redirect"
    hover-class="other-navigator-hover"
  >
    在当前页打开
  </navigator>
  <navigator
    url="/page/index/index"
    open-type="switchTab"
    hover-class="other-navigator-hover"
  >
    切换 Tab
  </navigator>
</view>
```

```
<!-- navigator.wxml -->
<view style="text-align:center">{{title}}</view>
<view>点击左上角返回回到之前页面</view>
```

```
// redirect.js navigator.js
Page({
  onLoad(options) {
    this.setData({
      title: options.title
    })
  }
})
```

媒体组件

最近更新时间：2024-05-11 17:21:31

image

- **功能说明：**图片。
- **参数及说明**

属性	类型	默认值	说明
src	string	-	图片资源地址，支持云文件 ID（2.2.3起）
mode	string	'scaleToFill'	图片裁剪、缩放的模式
lazy-load	boolean	false	图片懒加载。只针对 page 与 scroll-view 下的 image 有效
binderror	handleevent	-	当错误发生时，发布到 AppService 的事件名，事件对象 event.detail = {errMsg: 'something wrong'}
bindload	handleevent	-	当图片载入完毕时，发布到 AppService 的事件名，事件对象 event.detail = {height:'图片高度px', width:'图片宽度px'}
aria-lable	string	-	无障碍访问，（属性）元素的额外描述

> **注意：**
> - image 组件默认宽度300px、高度225px。
> - image组件中二维码/小程序码图片不支持长按识别。仅在 wx.previewImage 中支持长按识别。

mode 有效值：mode 有13种模式，其中4种是缩放模式，9种是裁剪模式。

模式	值	说明
缩放	scaleToFill	不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素
缩放	aspectFit	保持纵横比缩放图片，使图片的长边能完全显示出来。即可以完整地将图片显示出来
缩放	aspectFill	保持纵横比缩放图片，只保证图片的短边能完全显示出来。即图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取

缩放	widthFix	宽度不变，高度自动变化，保持原图宽高比不变
裁剪	top	不缩放图片，只显示图片的顶部区域
裁剪	bottom	不缩放图片，只显示图片的底部区域
裁剪	center	不缩放图片，只显示图片的中间区域
裁剪	left	不缩放图片，只显示图片的左边区域
裁剪	right	不缩放图片，只显示图片的右边区域
裁剪	top left	不缩放图片，只显示图片的左上角区域
裁剪	top right	不缩放图片，只显示图片的右上角区域
裁剪	bottom left	不缩放图片，只显示图片的左下边区域
裁剪	bottom right	不缩放图片，只显示图片的右下边区域

- **示例代码：**

  对应的 WXML 文件

```
<view class="page">
  <view class="page__hd">
    <text class="page__title">image</text>
    <text class="page__desc">图片</text>
  </view>
  <view class="page__bd">
    <view class="section section_gap" wx.for="{{array}}" wx.for-item="item">
      <view class="section__title">{{item.text}}</view>
      <view class="section__ctn">
        <image
          style="width: 200px; height: 200px; background-color: #eeeeee;"
          mode="{{item.mode}}"
          src="{{src}}"
        ></image>
      </view>
    </view>
  </view>
</view>
```

  对应的 js 文件

```
Page({
  data: {
    array: [{
      mode: 'scaleToFill',
      text: 'scaleToFill：不保持纵横比缩放图片，使图片完全适应'
    }, {
      mode: 'aspectFit',
      text: 'aspectFit：保持纵横比缩放图片，使图片的长边能完全显示出来'
    }, {
      mode: 'aspectFill',
      text: 'aspectFill：保持纵横比缩放图片，只保证图片的短边能完全显示出来'
    }, {
      mode: 'top',
      text: 'top：不缩放图片，只显示图片的顶部区域'
    }, {
      mode: 'bottom',
      text: 'bottom：不缩放图片，只显示图片的底部区域'
    }, {
      mode: 'center',
      text: 'center：不缩放图片，只显示图片的中间区域'
    }, {
      mode: 'left',
      text: 'left：不缩放图片，只显示图片的左边区域'
    }, {
      mode: 'right',
      text: 'right：不缩放图片，只显示图片的右边区域'
    }, {
      mode: 'top left',
      text: 'top left：不缩放图片，只显示图片的左上边区域'
    }, {
      mode: 'top right',
      text: 'top right：不缩放图片，只显示图片的右上边区域'
    }, {
      mode: 'bottom left',
      text: 'bottom left：不缩放图片，只显示图片的左下边区域'
    }, {
      mode: 'bottom right',
      text: 'bottom right：不缩放图片，只显示图片的右下边区域'
    }],
    src: '../resources/cat.jpg'
  },
  imageError(e) {
    console.log('image3发生error事件，携带值为', e.detail.errMsg)
  }
})
```

video

- **功能说明：**视频。低版本为原生组件，使用时请注意 原生组件 使用限制。

属性	类型	默认值	说明
src	string	–	要播放视频的资源地址，支持云文件 ID
duration	number	–	指定视频时长
controls	boolean	true	是否显示默认播放控件（播放/暂停按钮、播放进度、时间）
danmu-list	Object Array	–	弹幕列表
danmu-btn	boolean	false	是否显示弹幕按钮，只在初始化时有效，不能动态变更
enable-danmu	boolean	false	是否展示弹幕，只在初始化时有效，不能动态变更
autoplay	boolean	false	是否自动播放
loop	boolean	false	是否循环播放
muted	boolean	false	是否静音播放
initial-time	number	–	指定视频初始播放位置
page-gesture	boolean	false	在非全屏模式下，是否开启亮度与音量调节手势
direction	number	–	设置全屏时视频的方向，不指定则根据宽高比自动判断。 • 有效值为0（正常竖向） • 有效值为90（屏幕逆时针90度） • 有效值为-90（屏幕顺时针90度）
show-progress	boolean	true	若不设置，宽度大于240时才会显示
show-fullscreen-btn	boolean	true	是否显示全屏按钮
show-play-btn	boolean	true	是否显示视频底部控制栏的播放按钮
show-center-	boolean	true	是否显示视频中间的播放按钮

play-btn			
enable-progress-gesture	boolean	true	是否开启控制进度的手势
object-fit	string	contain	当视频大小与 video 容器大小不一致时，视频的表现形式。 • contain：包含 • fill：填充 • cover：覆盖
poster	string	-	视频封面的图片网络资源地址或云文件 ID。若 controls 属性值为 false 则设置 poster 无效
show-mute-btn	boolean	false	是否显示静音按钮
title	string	-	视频的标题，全屏时在顶部展示
play-btn-position	string	bottom	播放按钮的位置， • 有效值为 bottom（controls bar 上） • 有效值为 center（视频中间）
enable-play-gesture	boolean	false	是否开启播放手势，即双击切换播放/暂停
auto-pause-if-navigate	boolean	true	当跳转到其它小程序页面时，是否自动暂停本页面的视频
auto-pause-if-open-native	boolean	true	当跳转到其它原生页面时，是否自动暂停本页面的视频
bindplay	eventhandle	-	当开始/继续播放时触发 play 事件
bindpause	eventhandle	-	当暂停播放时触发 pause 事件
bindended	eventhandle	-	当播放到末尾时触发 ended 事件
bindtimeupdate	eventhandle	-	播放进度变化时触发，event.detail = {currentTime, duration} 。触发频率250ms一次
bindfullscreen	eventhan	-	视频进入和退出全屏时触发，event.detail =

change	dle		{fullScreen, direction}，direction 有效值为 vertical 或 horizontal
bindwaiting	eventhandle	–	视频出现缓冲时触发
binderror	eventhandle	–	视频播放出错时触发
bindprogress	eventhandle	–	加载进度变化时触发，只支持一段加载。event.detail = {buffered}，百分比

`<video>` 默认宽度300px、高度225px，可通过 wxss 设置宽高。

- **示例代码：**

对应的 WXML 文件

```
<view class="section tc">
  <video src="{{src}}" controls></video>
  <view class="btn-area">
    <button bindtap="bindButtonTap">获取视频</button>
  </view>
</view>

<view class="section tc">
  <video
    id="myVideo"
    src="https://qzonestyle.gtimg.cn/qzone/qzact/act/external/qq-video/qq-
video.mp4"
    danmu-list="{{danmuList}}"
    enable-danmu
    danmu-btn
    controls
  ></video>
  <view class="btn-area">
    <button bindtap="bindButtonTap">获取视频</button>
    <input bindblur="bindInputBlur" />
    <button bindtap="bindSendDanmu">发送弹幕</button>
  </view>
</view>
```

对应的 js 文件

```
function getRandomColor() {
  const rgb = []
```

```
  for (let i = 0; i < 3; ++i) {
    let color = Math.floor(Math.random() * 256).toString(16)
    color = color.length == 1 ? '0' + color : color
    rgb.push(color)
  }
  return '#' + rgb.join('')
}

Page({
  onReady(res) {
    this.videoContext = wx.createVideoContext('myVideo')
  },
  inputValue: '',
  data: {
    src: '',
    danmuList: [
      {
        text: '第 1s 出现的弹幕',
        color: '#ff0000',
        time: 1
      },
      {
        text: '第 3s 出现的弹幕',
        color: '#ff00ff',
        time: 3
      }]
  },
  bindInputBlur(e) {
    this.inputValue = e.detail.value
  },
  bindButtonTap() {
    const that = this
    wx.chooseVideo({
      sourceType: ['album', 'camera'],
      maxDuration: 60,
      camera: ['front', 'back'],
      success(res) {
        that.setData({
          src: res.tempFilePath
        })
      }
    })
  },
  bindSendDanmu() {
    this.videoContext.sendDanmu({
```

```
      text: this.inputValue,
      color: getRandomColor()
    })
  }
})
```

animation-video

- **功能说明：**animation-video 属于前端组件，为小程序提供了将特定视频资源渲染为透明背景动效的能力，可以帮助开发者低成本实现更为沉浸，丰富的动画效果。animation-video 组件还提供丰富的 API 来控制动画的播放，暂停，跳到指定位置等。相关 API 可参见 wx.createAnimationVideo 。
- **参数及说明：**

属性	类型	默认值	必填	说明	最低版本
resource-width	number	800	否	组件使用的 video 视频资源的宽度（单位：px）	1.4.99
resource-height	number	400	否	组件使用的 video 视频资源的高度（单位：px）	1.4.99
canvas-style	string	width: 400px; height: 400px;	否	用于设置动画画布的 CSS 样式	1.4.99
path	string		是	动画资源地址，支持相对路径以及远程地址。如果是远程路径，**response header 里需要设置 Access-Control-Allow-Origin 来防止跨域问题**	1.4.99
loop	boolean	false	否	动画是否循环播放	1.4.99
autoplay	boolean	false	否	动画是否自动	1.4.99
alpha-direction	string	left	否	视频资源中 alpha 通道的方向，left 表示 alpha 通道在资源的左边，right 表示 alpha 通道在资源的右边。	1.4.99
bindstarted	event handle	-	否	动画开始播放的回调	1.4.99

bindended	event handle	-	否	当播放到末尾时触发 ended 事件（自然播放结束会触发回调，循环播放结束及暂停动画不会触发）	1.4.99

- **示例代码：**

  对应的 WXML 文件

```
<view class="wrap">
  <view class="card-area">
    <view class="top-description border-bottom">
      <view>视频右侧为动画 alpha 通道信息</view>
      <view>alpha-direction='right'</view>
    </view>
    <view class="video-area">
      <animation-video
        path="{{rightAlphaSrcPath}}"
        loop="{{loop}}"
        resource-width="800"
        resource-height="400"
        canvas-style="width:200px;height:200px"
        autoplay="{{autoplay}}"
        bindstarted="onStarted"
        bindended="onEnded"
        alpha-direction='right'
      ></animation-video>
    </view>
  </view>

  <view class="card-area">
    <view class="top-description border-bottom">
      <view>视频左侧为动画 alpha 通道信息</view>
      <view>alpha-direction='left'</view>
    </view>
    <view class="video-area">
      <animation-video
        path="{{leftAlphaSrcPath}}"
        loop="{{loop}}"
        resource-width="800"
        resource-height="400"
        canvas-style="width:200px;height:200px"
        autoplay="{{autoplay}}"
        bindstarted="onStarted"
        bindended="onEnded"
        alpha-direction='left'
      ></animation-video>
```

```
      </view>
    </view>
  </view>
  Page({
    data: {
      loop: true,
      leftAlphaSrcPath: 'https://efe-h2.cdn.bcebos.com/ceug/resource/res/2020-
1/1577964961344/003e2f0dcd81.mp4',
      rightAlphaSrcPath: 'https://b.bdstatic.com/miniapp/assets/docs/alpha-right-
example.mp4',
      autoplay: true
    },
    onStarted() {
      console.log('===onStarted');
    },
    onEnded() {
      console.log('===onEnded');
    }
  })
```

camera

- **功能说明：**系统相机。该组件是 原生组件，使用时请注意相关限制。

  需要“用户授权”：scope.camera。

  相关 API 请参见 wx.createCameraContext。

  > **说明：**
  > 同一页面只能插入一个 camera 组件。

- **参数及说明：**

属性	类型	默认值	说明
mode	string	normal	有效值为 normal, scanCode
device-position	string	back	前置或后置，值为 front, back
flash	string	auto	闪光灯，值为 auto, on, off
bindstop	eventhandle	-	摄像头在非正常终止时触发，如退出后台等情况
binderror	eventha	-	用户不允许使用摄像头时触发

	ndle		
bindscancode	eventhandle	_	在扫码识别成功时触发，仅在 mode="scanCode" 时生效

- **示例代码：**

```
<!-- camera.wxml -->
<camera
  device-position="back"
  flash="off"
  binderror="error"
  style="width: 100%; height: 300px;"
></camera>
<button type="primary" bindtap="takePhoto">拍照</button>
<view>预览</view>
<image mode="widthFix" src="{{src}}"></image>
```

```
// camera.js
Page({
  takePhoto() {
    const ctx = wx.createCameraContext()
    ctx.takePhoto({
      quality: 'high',
      success: (res) => {
        this.setData({
          src: res.tempImagePath
        })
      }
    })
  },
  error(e) {
    console.log(e.detail)
  }
})
```

live-player

- **功能说明：**实时音视频播放。该组件是原生组件，使用时请注意 原生组件的限制 。
- **参数及说明：**

属性	类型	默认值	必填	说明	最低版本

src	string	–	否	音视频地址。目前仅支持 flv, rtmp 格式	1.4.96
mode	string	live	否	模式	1.4.96
autoplay	boolean	false	否	自动播放	1.4.96
muted	boolean	false	否	是否静音	1.4.96
orientation	string	vertical	否	画面方向	1.4.96
object-fit	string	contain	否	填充模式，可选值有 contain, fillCrop	1.4.96
min-cache	number	1	否	最小缓冲区，单位 s	1.4.96
max-cache	number	3	否	最大缓冲区，单位 s	1.4.96
sound-mode	string	speaker	否	声音输出方式	1.4.96
auto-pause-if-navigate	boolean	true	否	当跳转到其它小程序页面时，是否自动暂停本页面的实时音视频播放	1.4.96
auto-pause-if-open-native	boolean	true	否	当跳转到其它 QQ 原生页面时，是否自动暂停本页面的实时音视频播放	1.4.96
enable-metadata	boolean	false	否	是否回调 metadata	1.4.96
bindstatechange	eventhandle	–	否	播放状态变化事件，detail = {code}	1.4.96
bindfullscreenchange	eventhandle	–	否	全屏变化事件，detail = {direction, fullScreen}	1.4.96
bindnetstatus	eventhandle	–	否	网络状态通知，detail = {info}	1.4.96
bindmetadatachange	eventhandle	–	否	metadata通知，detail = {info}	1.4.96

- **mode 的合法值**

值	说明	最低版本
live	直播	1.4.96

- **orientation 的合法值**

值	说明	最低版本
vertical	竖直	1.4.96
horizontal	水平	1.4.96

- **object-fit的合法值**

值	说明	最低版本
contain	图像长边填满屏幕，短边区域会被填充黑色	1.4.96
fillCrop	图像铺满屏幕，超出显示区域的部分将被裁掉	1.4.96

- **sound-mode 的合法值**

值	说明	最低版本
speaker	扬声器	1.4.96

- **状态码**

代码	说明
2001	已经连接服务器
2002	已经连接服务器，开始拉流
2003	网络接收到首个视频数据包（IDR）
2004	视频开始播放
2006	视频播放结束
2007	视频播放 Loading
2008	解码器启动

2009	视频分辨率改变
-2301	网络断连，且多次重连无效，更多重试请自行重启播放
-2302	获取加速拉流地址失败
2101	当前视频帧解码失败
2102	当前音频帧解码失败
2103	网络断连，已启动自动重连
2104	网络来包不稳。可能是下行带宽不足，或由于主播端出流不均匀
2105	当前视频播放出现卡顿
2106	硬解启动失败，采用软解
2107	当前视频帧不连续，可能丢帧
2108	当前流硬解第一个I帧失败，SDK 自动切软解
3001	RTMP -DNS 解析失败
3002	RTMP 服务器连接失败
3003	RTMP 服务器握手失败
3005	RTMP 读/写失败

- **网络状态数据**

键名	说明
videoBitrate	当前视频数据接收比特率，单位：kbps
audioBitrate	当前音频数据接收比特率，单位：kbps
videoFPS	当前视频帧率
videoGOP	当前视频GOP，也就是每两个关键帧（I帧）间隔时长，单位：s
netSpeed	当前的发送/接收速度
netJitter	网络抖动情况，抖动越大，网络越不稳定
videoWidth	视频画面的宽度

videoHeight	视频画面的高度

- **示例代码：**

对应的 WXML 文件

```
<live-player id="playerid" src="https://domain/pull_stream" mode="live" autoplay bindstatechange="statechange" binderror="error" />
```

对应的 js 文件

```
Page({
  statechange(e) {
    console.log('live-player code:', e.detail.code)
  },
  error(e) {
    console.error('live-player error:', e.detail.errMsg)
  }
})
```

> **说明：**
> - `live-player` 默认宽度300px、高度225px，可通过 wxss 设置宽高。
> - 开发者工具上暂不支持。

live-pusher

- **功能说明：**实时音视频录制（暂不支持同层渲染）需要“用户授权” `scope.camera` 、`scope.record` 。
- **参数及说明：**

属性	类型	默认值	必填	说明	最低版本
url	string	-	否	推流地址。目前仅支持 rtmp	1.4.96
mode	string	RTC	否	SD（标清），HD（高清），FHD（超清），RTC（实时通话）	1.4.96
autopush	boolean	false	否	自动推流	1.4.96

muted	boolean	false	否	是否静音	1.4.96
enable-camera	boolean	true	否	开启摄像头	1.4.96
auto-focus	boolean	true	否	自动聚集	1.4.96
orientation	string	vertical	否	画面方向	1.4.96
beauty	number	0	否	美颜，取值范围 0-9 ，0 表示关闭	1.4.96
whiteness	number	0	否	美白，取值范围 0-9 ，0 表示关闭	1.4.96
aspect	string	9：16	否	宽高比，可选值有 3:4，9:16	1.4.96
min-bitrate	number	200	否	最小码率	1.4.96
max-bitrat	number	1000	否	最大码率	1.4.96
audio-quality	string	height	否	高音质(48KHz)或低音质(16KHz)，值为high, low	1.4.96
waiting-image	string	-	否	进入后台时推流的等待画面	1.4.96
waiting-image-hash	string	-	否	等待画面资源的 MD5 值	1.4.96
zoom	boolean	false	否	调整焦距	1.4.96
device-position	string	-	否	前置或后置，值为 front，back	1.4.96
background-mute	boolean	false	否	进入后台时是否静音	1.4.96
mirror	boolean	false	否	设置推流画面是否镜像，产生	1.4.9

				的效果在 live-player 反应到	6
bindstatechange	eventhandle	-	否	状态变化事件，detail = {code}	1.4.96
bindnetstatus	eventhandle	-	否	网络状态通知，detail = {info}	1.4.96
binderror	eventhandle	-	否	渲染错误事件，detail = {errMsg, errCode}	1.4.96
bindbgmstart	eventhandle	-	否	背景音开始播放时触发	1.4.96
bindbgmprogress	eventhandle	-	否	背景音进度变化时触发，detail = {progress, duration}	1.4.96
bindbgmcomplete	eventhandle	-	否	背景音播放完成时触发	1.4.96
audio-reverb-type	number	0	否	混响模式(0～6)："关闭混响"，"KTV"，"小房间"，"大会堂"，"低沉"，"洪亮"，"磁性"	1.4.96

- **orientation 的合法值**

值	说明	最低版本
vertical	竖直	1.4.96
horizontal	水平	1.4.96

- **错误码（errCode）**

代码	说明
10001	用户禁止使用摄像头
10002	用户禁止使用录音
10003	背景音资源（BGM）加载失败
10004	等待页面资源（waiting-image）加载失败

- **状态码（code）**

代码	说明
1001	已经连接推流服务器
1002	已经与服务器握手完毕,开始推流
1003	打开摄像头成功
1004	录屏启动成功
1005	推流动态调整分辨率
1006	推流动态调整码率
1007	首帧画面采集完成
1008	编码器启动
-1301	打开摄像头失败
-1302	打开麦克风失败
-1303	视频编码失败
-1304	音频编码失败
-1305	不支持的视频分辨率
-1306	不支持的音频采样率
-1307	网络断连，且经多次重连抢救无效，更多重试请自行重启推流
-1308	开始录屏失败，可能是被用户拒绝
-1309	录屏失败，不支持的 Android 系统版本，需要5.0以上的系统
-1310	录屏被其他应用打断了
-1311	Android Mic 打开成功，但是录不到音频数据
-1312	录屏动态切横竖屏失败
1101	网络状况不佳：上行带宽太小，上传数据受阻
1102	网络断连, 已启动自动重连

1103	硬编码启动失败,采用软编码
1104	视频编码失败
1105	新美颜软编码启动失败，采用老的软编码
1106	新美颜软编码启动失败，采用老的软编码
3001	RTMP -DNS 解析失败
3002	RTMP 服务器连接失败
3003	RTMP 服务器握手失败
3004	RTMP 服务器主动断开，请检查推流地址的合法性或防盗链有效期
3005	RTMP 读/写失败

- **网络状态数据（info）**

键名	说明
videoBitrate	当前视频编/码器输出的比特率，单位：kbps
audioBitrate	当前音频编/码器输出的比特率，单位：kbps
videoFPS	当前视频帧率
videoGOP	当前视频 GOP，也就是每两个关键帧（I帧）间隔时长，单位：s
netSpeed	当前的发送/接收速度
netJitter	网络抖动情况，抖动越大，网络越不稳定
videoWidth	视频画面的宽度
videoHeight	视频画面的高度

- **示例代码：**

```
 <live-pusher url="https://domain/push_stream" mode="RTC" autopush
bindstatechange="statechange" style="width: 300px; height: 225px;" />
Page({
 statechange(e) {
  console.log('live-pusher code:', e.detail.code)
 }
```

```
})
```

地图组件

最近更新时间：2024-03-21 15:51:53

map

功能说明：地图。该组件是 原生组件，使用时请注意相关限制。相关 API 可参见 createMapContext 。

支持度说明：

- 系统地图：仅 iOS 支持
- 谷歌地图：Android 支持、IDE 支持
- 华为地图：仅 Android 支持
- 腾讯地图：Android 支持、IDE 支持

属性	类型	默认值	必填	说明	iOS 支持度	Android 支持度	IDE 支持度
type	string	tencent	否	地图类型，tencent 或者 google	否	否	是
longitude	number	-	是	中心经度	是	是	是
latitude	number	-	是	中心纬度	是	是	是
scale	number	16	否	缩放级别，取值范围为3~20	是	是	是
min-scale	number	3	否	最小缩放级别		是	是
max-scale	number	20	否	最大缩放级别		是	是
markers	array.<marker>	-	否	标记点	是	是	是
polyline	array.<polyline>	-	否	路线	是	是	是

circles	array.<circle>	-	否	圆	是	是	是
controls	array.<control>	-	否	控件	否	否	否
include-points	array.<point>	-	否	缩放视野以包含所有给定的坐标点	是	是	是
show-location	boolean	false	否	显示带有方向的当前定位点	是	是	是
polygons	array.<polygon>	-	否	多边形	是	是	是
subkey	string	-	否	个性化地图使用的 key	否	否	是 type=tencent
layer-style	number	1	否	个性化地图配置的style，不支持动态修改	否	否	否
rotate	number	0	否	旋转角度，范围 0 ~ 360，地图正北与设备 y 轴的夹角角度	否	是	否
skew	number	0	否	倾斜角度，范围 0 ~ 40 , 关于 z 轴的倾角	• 系统地图: 否 • 腾讯	是	否

					地图: 是 • 百度地图: 是 • 高德地图: 是		
enable-3D	boolean	false	否	展示3D楼块	• 系统地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	• 谷歌地图: 是 • 华为地图: 是 • 腾讯地图: 是 • 百度地图: 是 • 高德地图: 否	否
show-compass	boolean	false	否	显示指南针	是	是	否
show-scale	boolean	false	否	显示比例尺	是	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是	是

						• 百度地图: 是 • 高德地图: 是	
enable-overlooking	boolean	false	否	开启俯视	• 系统地图: 否 • 腾讯地图: 是 • 百度地图: 是 • 高德地图: 是	是	否
enable-zoom	boolean	true	否	是否支持缩放	是	是	是
enable-scroll	boolean	true	否	是否支持拖动	是	是	是
enable-rotate	boolean	-	否	是否支持旋转	是	是	是
enable-satellite	boolean	false	否	是否开启卫星图	是	• 谷歌地图: 是 • 华为地图: 否 • 腾讯地图: 是	是 type=google

						• 百度地图: 是 • 高德地图: 是	
enable-traffic	boolean	false	否	是否开启实时路况	是	是	是
enable-poi	boolean	true	否	是否展示 POI 点	• 系统地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	• 谷歌地图: 是 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 是 • 高德地图: 是	是 type=google
enable-building	boolean	true	否	是否展示建筑物	• 系统地图: 否 • 腾讯地图: 是 • 百度地图: 是	• 谷歌地图: 是 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 是	是 type=google

					• 高德地图: 是		
setting	object	–	否	配置项	否	否	是
bindtap	event handle	–	否	单击地图时触发，返回经纬度信息	是	是	是
bindmarkertap	event handle	–	否	点击标记点时触发 e.detail = {markerId}	是	是	是
bindlabeltap	event handle	–	否	点击 label 时触发，e.detail = {markerId}	是	否	是 type=tencent
bindcontroltap	event handle	–	否	点击控件时触发，e.detail = {controlId}	否	否	否
bindcalloutap	event handle	–	否	点击标记点对应的气泡时触发 e.detail = {markerId}	否	否	是
bindupdated	event handle	–	否	在地图渲染更新完成时触发	否	是	是
bindregionchange	event handle	–	否	视野发生变化时触发两次，返回的 type 值分别为 begin/end。	是	是	是
bindpoitap	event handle	–	否	单击地图 poi 点时触发 e.detail = {name, longitude, latitude}	否	是	是

bindanchorpointtap	eventhandle	-	否	点击定位标时触发，e.detail = {longitude, latitude}	否	否	否

regionchange 返回值

- **功能说明：视野改变时，regionchange 会触发两次，返回的 type 值分别为 begin 和 end。**

  begin 阶段返回 causedBy，有效值为 gesture（手势触发）、update（接口触发）。
  end 阶段返回 causedBy，有效值为 drag（拖动导致）、scale（缩放导致）、update（调用更新接口导致）。
  `e = {causedBy, type, detail: {rotate, skew, scale, centerLocation, region}}`

setting

- **功能说明：**提供 setting 对象统一设置地图配置。
- **示例代码：**

```
// 默认值
const setting = {
  rotate: 0,
  showLocation: false,
  subKey: '',
  layerStyle: 1,
  enableZoom: true,
  enableScroll: true,
  enableRotate: false,
  showCompass: false,
  enableSatellite: false,
}

this.setData({
  // 仅设置的属性会生效，其它的不受影响
  setting: {
    enableZoom: false,
    enableScroll: false,
  }
})
```

maker

- **功能说明：**标记点用于在地图上显示标记的位置。

属性	说明	类型	必填	备注	iOS 支	Androi	IDE 支持

					持	d 支持	
id	标记点 id	number	否	marker 点击事件回调会返回此 id。	是	是	是
clusterId	聚合簇的 id	number	否	自定义点聚合簇效果时使用	否	否	否
joinCluster	是否参与点聚合	boolean	否	默认不参与点聚合	否	• 谷歌地图: 是 • 华为地图: 是 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	否
latitude	纬度	number	是	浮点数，范围 -90 ~ 90	是	是	是
longitude	经度	number	是	浮点数，范围 -90 ~ 90	是	是	是
title	标注点名	string	否	点击时显示，callout 存在时将被忽略	是	是	是 type=tencent
zIndex	显示层级	number	否	-	否	是	是

iconPath	显示的图标	string	是	项目目录下的图片路径，支持网络路径、本地路径、代码包路径	是	是	是
rotate	旋转角度	number	否	顺时针旋转的角度，范围 0~360，默认为 0	否	是	是 type=tencent
alpha	标注的透明度	number	否	默认1，无透明，范围0~1	否	是	是 type=tencent
width	标注图标宽度	number/string	否	默认为图片实际宽度	否	是	否
height	标注图标高度	number/string	否	默认为图片实际高度	否	是	否
callout	标记点上方的气泡窗口	object	否	支持的属性见下表，可识别换行符。	是	是	是
customCallout	自定义气泡窗口	object	否	支持的属性见下表	否	否	否
label	为标记点旁边增加标签	object	否	支持的属性见下表，可识别换行符。	否	否	是
anchor	经纬度在标注图标的锚点，	object	否	{x, y}，x 表示横向(0-1)，y 表示竖向(0-1)。{x: .5, y: 1} 表示底边中点	否	是	否

	默认底边中点						
aria-label	无障碍访问，（属性）元素的额外描述	string	否	–	否	否	否

> ⚠ **注意：**
> 建议为每个 marker 设置上 number 类型 id，保证更新 marker 时有更好的性能。

marker 上的气泡 callout

属性	说明	类型	iOS 支持度	Android 支持度	IDE 支持度
content	文本	string	是	是	是
color	文本颜色	string	否	是	是
fontSize	文字大小	number	否	是	是
borderRadius	边框圆角	number	否	是	是
borderWidth	边框宽度	number	否	是	是
borderColor	边框颜色	string	否	是	是
bgColor	背景色	string	否	是	是
padding	文本编译留白	number	否	是	是
display	'BYCLICK':点击显示; 'ALWAYS':常显	string	否	否	是
textAlign	文本对齐方式。有效值: left,	string	否	是	否

	right, center				
anchorX	横向偏移量，向右为正数	number	否	• 谷歌地图: 是 • 华为地图: 是 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 是	否
anchorY	纵向偏移量，向下为正数	number	否	• 谷歌地图: 是 • 华为地图: 是 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 是	否

marker 上的自定义气泡 customCallout

- **功能说明：** `customCallout` 存在时将忽略 `callout` 与 `title` 属性。自定义气泡采用采用 `cover-view` 定制，灵活度更高。

属性	说明	类型	iOS 支持度	Android 支持度	IDE 支持度
display	'BYCLICK':点击显示; 'ALWAYS':常显	string	否	否	否
anchorX	横向偏移量，向右为正数	number	否	否	否
anchorY	纵向偏移量，向下为正数	number	否	否	否

使用方式如下， `map` 组件下添加名为 callout 的 slot 节点，其内部的 `cover-view` 通过 `marker-id` 属性与 `marker` 绑定。当 `marker` 创建时，该 `cover-view` 显示的内容将作为 `callout` 显示在标记点上方。

```
<map>
  <cover-view slot="callout">
    <cover-view marker-id="1"></cover-view>
    <cover-view marker-id="2"></cover-view>
  </cover-view>
</map>
```

marker 上的气泡 label

属性	说明	类型	iOS 支持度	Android 支持度	IDE 支持度
content	文本	string	否	否	是
color	文本颜色	string	否	否	是
fontSize	文字大小	number	否	否	是
anchorX	label 的坐标，原点是 marker 对应的经纬度	number	否	否	是
anchorY	label 的坐标，原点是 marker 对应的经纬度	number	否	否	是
borderWidth	边框宽度	number	否	否	是
borderColor	边框颜色	string	否	否	是
borderRadius	边框圆角	number	否	否	是
bgColor	背景色	string	否	否	是
padding	文本边缘留白	number	否	否	是
textAlign	文本对齐方式。有效值：left，right，center	string	否	否	是

点聚合

- **功能说明：**当地图上需要展示的标记点 marker 过多时，可能会导致界面上 marker 出现压盖，展示不全，并导致整体性能变差。针对此类问题，推出点聚合能力。
- **系统地图（仅 iOS 支持）**

使用流程如下：

- MapContext.addMarkers 指定参与聚合的 marker。
- （可选）MapContext.on('markerClusterCreate', callback) 触发时，通过 MapContext.addMarkers 更新聚合簇的样式。
- MapContext.removeMarkers 移除参与聚合的 marker。

- **谷歌地图（仅 Android 支持）**

使用方法为添加 marker 时指定 joinCluster 即可，符合条件的 marker 将会自动进行聚合。

polyline

- **功能说明：**指定一系列坐标点，从数组第一项连线至最后一项。绘制彩虹线时，需指定不同分段的颜色，如 points 包含 5 个点，则 colorList 应传入 4 个颜色值；若 colorList 长度小于 points.length – 1，则剩下的分段颜色与最后一项保持一致。

属性	说明	类型	必填	备注	iOS 支持度	Android 支持度	IDE 支持度
points	经纬度数组	Array	是	[{latitude: 0, longitude: 0}]	是	是	是
color	线的颜色	string	否	十六进制	是	是	是
colorList	彩虹线	Array	否	存在时忽略 color 值	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 是 • 高德地图: 是	否
width	线的宽度	number	否		是	是	是
dottedLine	是否虚线	boolean	否	默认 false	是	是	是
arrowLine	带箭头的线	boolean	否	–	否	是	否
arrowl	更换箭头	string	否	–	否	是	否

conPath	图标						
borderColor	线的边框颜色	string	否	-	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	是
borderWidth	线的厚度	number	否	-	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	是
level	压盖关系	string	否	默认为abovelabels	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	否
textStyle	文字样式	TextStyle	否	折线上文本样式	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地	否

						图: 否 • 高德地图: 否	
segmentTexts	分段文本	Array<SegmentText>	否	折线上文本内容和位置	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	否

SegmentText

属性	说明	类型
name	名称	string
startIndex	起点	number
endIndex	终点	number

TextStyle

属性	说明	类型	默认值
textColor	文本颜色	string	#000000
strokeColor	描边颜色	number	#ffffff
fontSize	文字大小	number	14

`level` 字段表示与其它地图元素的压盖关系，可选值如下：

值	说明
abovelabels	显示在所有 POI 之上
abovebuildings	显示在楼块之上 POI 之下
aboveroads	显示在道路之上楼块之下

polygon

- **功能说明：**指定一些列坐标点，根据points坐标数据生成闭合多边形。

属性	说明	类型	必填	备注	iOS支持	Android支持	IDE支持
dashArray	边线虚线	Array<number>	否	默认值 [0, 0] 为实线，[10, 10]表示十个像素的实线和十个像素的空白（如此反复）组成的虚线	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	否
points	经纬度数组	array	是	[{latitude: 0, longitude: 0}]	是	是	是
strokeWidth	描边的宽度	number	否	-	是	是	否
strokeColor	描边的颜色	string	否	十六进制	是	是	否
fillColor	填充颜色	string	否	十六进制	是	是	是
zIndex	设置多边形 Z 轴数值	number	否	-	否	是	是
level	压盖关系	string	否	默认为 abovelabels	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地	否

						图: 否	

circle

- **功能说明：**在地图上显示图。

属性	说明	类型	必填	备注	iOS 支持度	Android 支持度	IDE 支持度
latitude	纬度	number	是	浮点数，范围 -90 ~ 90	是	是	是
longitude	经度	number	是	浮点数，范围 -180 ~ 180	是	是	是
color	描边的颜色	string	否	浮点数，范围 -180 ~ 180	是	是	是
fillColor	填充颜色	string	否	十六进制	是	是	是
radius	半径	number	是	-	是	是	是
strokeWidth	描边的宽度	number	否	-	是	是	否
level	压盖关系	string	否	默认为 abovelabels	否	• 谷歌地图: 否 • 华为地图: 否 • 腾讯地图: 是 • 百度地图: 否 • 高德地图: 否	否

control

- **功能说明：**在地图上显示控件，控件不随着地图移动。即将废弃，推荐使用 cover-view。

属性	说明	类型	必填	备注	iOS 支持度	Android 支持度	IDE 支持度

id	控件 id	number	否	在控件点击事件会回调会返回此 id	否	否	否
position	空间在地图的位置	object	是	空间相对地图位置	否	否	否
iconPath	显示的图标	string	是	项目目录下的图片路径，支持本地路径、代码包路径	否	否	否
clickable	是否可点击	boolean	否	默认不可点击	否	否	否

position

属性	说明	类型	必填	备注	iOS 支持度	Android 支持度	IDE 支持度
left	距离地图的左边界多远	number	否	默认为0	否	否	否
top	距离地图的上边界多远	number	否	默认为0	否	否	否
width	空间宽度	number	否	默认为图片宽度	否	否	否
height	空间高度	number	否	默认为图片高度	否	否	否

bindregionchange 返回值

属性	说明	类型	备注
type	视野变化开始、结束时触发	string	视野变化开始为 begin，结束为 end
causedBy	导致视野变化的原因	string	拖动地图导致（drag）、缩放导致（scale）、调用接口导致（update）

比例尺

scale	3	4	5	6	7	8	9	10	11

比例	1000km	500km	200km	100km	50km	50km	20km	10km	5km
scale	12	13	14	15	16	17	18	19	20
比例	2km	1km	500m	200m	100m	50m	50m	20m	10m

说明：

- 个性化地图暂不支持工具中调试。请先使用客户端进行测试。
- 地图中的颜色值color/borderColor/bgColor等需使用6位（8位）十六进制表示，8位时后两位表示 alpha 值，如：#000000AA。
- 地图组件的经纬度必填, 如果不填经纬度则默认值是北京的经纬度。
- map 组件使用的经纬度是火星坐标系，调用 wx.getLocation 接口需要指定 type 为 gcj02。
- 请注意 原生组件 使用限制。
- 若当前组件所在的页面或全局开启了 enablePassiveEvent 配置项，该内置组件可能会出现非预期表现。

Canvas 画布

最近更新时间：2024-05-17 11:15:41

- **功能说明：**画布。该组件是 原生组件 ，使用时请注意相关限制。

> **说明：**
> - 相关 API 可参见 wx.createCanvasContext 。
> - 避免设置过大的宽高，在 Android 环境下会有 crash 的问题。

属性名	类型	默认值	说明
canvas-id	string	-	canvas 组件的唯一标识符
disable-scroll	boolean	false	当在 canvas 中移动时且有绑定手势事件时，禁止屏幕滚动以及下拉刷新
bindtouchstart	eventhandle	-	手指触摸动作开始
bindtouchmove	eventhandle	-	手指触摸后移动
bindtouchend	eventhandle	-	手指触摸动作结束
bindtouchcancel	eventhandle	-	手指触摸动作被打断，如来电提醒，
bindlongtap	eventhandle	-	手指长按500ms之后触发，触发了长按事件后进行移动不会触发屏幕的滚动
binderror	eventhandle	-	当发生错误时触发 error 事件，detail = {errMsg: 'something wrong'}

> **说明：**
> - canvas 标签默认宽度300px、高度150px。
> - 同一页面中的 canvas-id 不可重复，如果使用一个已经出现过的 canvas-id，该 canvas 标签对应的画布将被隐藏并不再正常工作。

- **示例代码：**

```
<!-- canvas.wxml -->
<canvas style="width: 300px; height: 200px;" canvas-id="firstCanvas">
</canvas>
<!-- 当使用绝对定位时，文档流后边的 canvas 的显示层级高于前边的 canvas -->
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas">
</canvas>
<!-- 因为 canvas-id 与前一个 canvas 重复，该 canvas 不会显示，并会发送一个错误事件到
AppService -->
<canvas
  style="width: 400px; height: 500px;"
  canvas-id="secondCanvas"
  binderror="canvasIdErrorCallback"
></canvas>
```

```
// canvas.js
Page({
  canvasIdErrorCallback(e) {
    console.error(e.detail.errMsg)
  },
  onReady(e) {
    // 使用 wx.createContext 获取绘图上下文 context
    const context = wx.createCanvasContext('firstCanvas')

    context.setStrokeStyle('#00ff00')
    context.setLineWidth(5)
    context.rect(0, 0, 200, 200)
    context.stroke()
    context.setStrokeStyle('#ff0000')
    context.setLineWidth(2)
    context.moveTo(160, 100)
    context.arc(100, 100, 60, 0, 2 * Math.PI, true)
    context.moveTo(140, 100)
    context.arc(100, 100, 40, 0, Math.PI, false)
    context.moveTo(85, 80)
    context.arc(80, 80, 5, 0, 2 * Math.PI, true)
    context.moveTo(125, 80)
    context.arc(120, 80, 5, 0, 2 * Math.PI, true)
    context.stroke()
    context.draw()
  }
})
```

开放能力

最近更新时间：2024-05-11 17:21:31

web-view

- **功能说明：**web-view 组件是一个可以用来承载网页的容器，会自动铺满整个小程序页面，个人类型小程序暂不支持使用，企业开发者需先在开发者平台上选择服务类目。
- **参数及说明：**

属性	类型	默认值	说明
src	string	–	webview 指向网页的链接。网页需登录TMF 小程序管理后台配置业务域名
bindload	eventhandler	–	网页向小程序 postMessage 时，会在特定时机（小程序后退、组件销毁、分享）触发并收到消息。e.detail = { data }，data 是多次 postMessage 的参数组成的数组
binderror	eventhandler	–	网页加载成功时候触发此事件。e.detail = { src }
binderror	eventhandler	–	网页加载失败的时候触发此事件。e.detail = { src }

- **示例代码：**

```
<!-- wxml -->
<web-view src="https://www.qq.com/"></web-view>
```

- **相关接口1**
- `web-view>` **网页中可使用 JSSDK 提供的接口返回小程序页面。 支持的接口有：**

接口名	说明
wx.miniProgram.navigateTo	参数与小程序接口一致
wx.miniProgram.navigateBack	参数与小程序接口一致

wx.miniProgram.switchTab	参数与小程序接口一致
wx.miniProgram.reLaunch	参数与小程序接口一致
wx.miniProgram.redirectTo	参数与小程序接口一致
wx.miniProgram.postMessage	向小程序发送消息，会在以下特定时机触发组件的 message 事件：小程序后退、组件销毁、分享、复制链接
wx.miniProgram.getEnv	获取当前环境

- **示例代码：**

```
<!-- html -->
<script type="text/javascript"
src="https://qqq.gtimg.cn/miniprogram/webview_jssdk/qqjssdk-1.0.0.js"></script>
```

```
wx.miniProgram.navigateTo({url: '/path/to/page'})
wx.miniProgram.postMessage({ data: 'foo' })
wx.miniProgram.postMessage({ data: {foo: 'bar'} })
wx.miniProgram.getEnv(function(res) { console.log(res.miniprogram) })
```

- **相关接口2**

  `<web-view>` **页中暂不支持其他媒体相关接口。**

- **相关接口3**

  用户分享时可获取当前 `<web-view>` **的 URL，即在** `onShareAppMessage` **回调中返回** `webViewUrl` **参数。**

- **示例代码：**

```
Page({
  onShareAppMessage(options) {
    console.log(options.webViewUrl)
  },
})
```

- **相关接口4**

在网页内可通过 window.__wxjs_environment 变量判断是否在小程序环境，建议在 WeixinJSBridgeReady 回调中使用，也可以使用 JSSDK 提供的 getEnv 接口。

- **示例代码：**

```
// web-view下的页面内
function ready() {
  console.log(window.__wxjs_environment === 'miniprogram') // true
}
if (!window.QQJSBridge || !QQJSBridge.invoke) {
  document.addEventListener('WeixinJSBridgeReady', ready, false)
} else {
  ready()
}

// 或者
wx.miniProgram.getEnv(function(res) {
  console.log(res.miniprogram) // true
})
```

- **相关接口 5**

可以通过判断 userAgent 中包含 miniProgram 字样来判断小程序 web-view 环境。

> **说明：**
> - 网页内 iframe 的域名也需要配置到域名白名单；
> - 开发者工具上，可以在 <web-view>组件上通过右键 - 调试，打开 <web-view>组件的调试；
> - 每个页面只能有一个 <web-view>，<web-view>会自动铺满整个页面，并覆盖其他组件；
> - <web-view>网页与小程序之间不支持除 JSSDK 提供的接口之外的通信；
> - 避免在链接中带有中文字符，在 iOS 中会有打开白屏的问题，建议加一下 encodeURIComponent；
> - 提示 insertHTMLWebView:fail no permission 时即没有权限使用<web-view>，目前个人开发者以及未选定服务类目的企业开发者不支持调用。

IDE 操作说明

最近更新时间：2024-02-26 10:34:51

TCMPP IDE 是一款跨平台的桌面应用程序，旨在帮助开发人员完成小程序的开发、预览和调试。在完成调试和预览后，开发人员需要在真机上进行回归测试，因为部分设备 API（如蓝牙、NFC、WIFI 等）必须在真机上测试。

支持平台及获取

TCMPP 小程序开发者工具支持常见主流平台，下载流程请参见 无代码构建小程序 。

使用

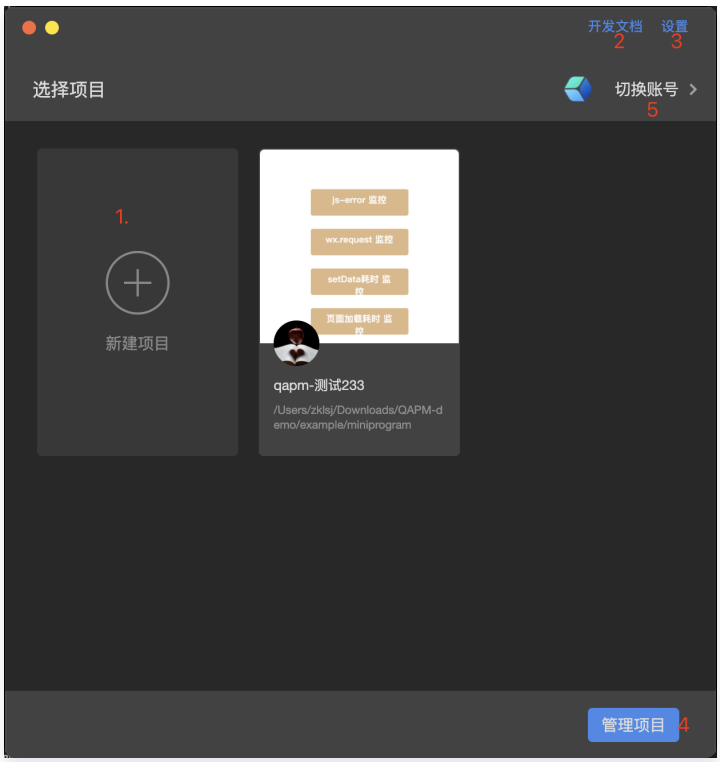
登录

需要用腾讯云控制台账号绑定的微信进行扫码，微信扫码时会出现账号信息，**请选择时仔细确认后再确认登录**。

界面概述

成功登录后，您可以进入项目列表管理页面，该页面提供以下功能：

1. 新建项目：您可以在此处创建新的项目。
2. 开发文档：您可以前往对应的开发文档页面，以查看更多小程序信息。
3. 设置：您可以在此处进行设置，与登录时相同。
4. 管理项目：您可以在此处对已有项目进行删除和管理。
5. 切换账号：您可以在此处切换账号，并进入开发调试页面。

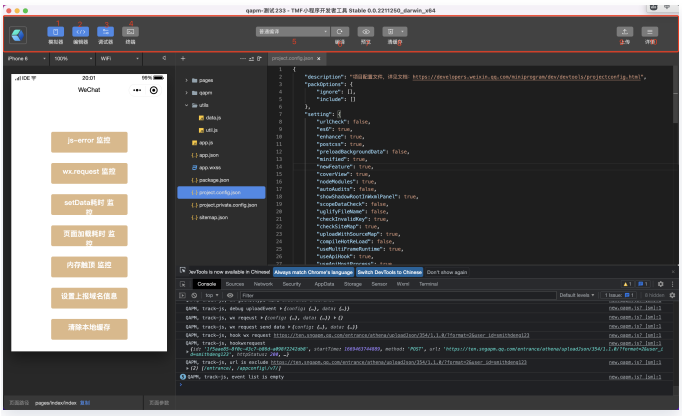


新建项目页面

1. 在本地文件系统选择新建项目的位置。
2. 填写 AppID（可在 小程序主页 中获取），或选择测试号进行使用，之后可以在开发面板修改 AppID。
3. 输入项目名称。
4. 确认返回到项目列表页面。

开发菜单

选中或新建项目后，单击进入开发面板页面。



开发面板顶部区域介绍：

1. 模拟器开关。
2. 编辑器开关。
3. 调试区开关。
4. 终端入口。
5. 编译方式：

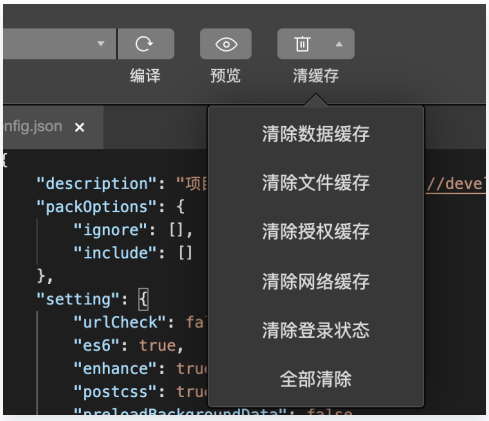
   默认普通编译是从小程序的入口开始加载 app.json 的 pages 数组第一项。

   如果想在IDE中默认打开指定页面，可以选择添加编译模式进行配置。这样可以加载指定页面，同时可以模拟参数传递。配置保存后会存储在小程序项目路径 `project.config.json` 中。

   - **模式名称**: 自定义编译的名字。

- **启动页面**: 编译加载的默认入口页面 (路径取值范围: app.json 中 pages 数组其中一项)。
- **参数**: 加载页面的参数 eg:name=tmf&age=20，然后逻辑层代码中onLoad方法可以获取参数。
- **下次编译时模拟更新**: 结合 wx.getUpdateManager 使用，默认生效一次后取消。

6. 单击**编译**，刷新页面。
7. 单击页面右上角**预览**，会把小程序上传到后台，等待上传完成以后，IDE 会弹出二维码，可以用小程序预览小程序或集成了小程序 SDK 的 Android/iOS App 扫码进行预览。
8. 单击**清缓存**后，可选择清除数据缓存、文件缓存、授权缓存、网络缓存、登录缓存或全部缓存。



9. 上传小程序：在开发菜单右上角单击**上传**，在弹窗中，填写小程序 AppID 以及版本备注后，单击**上传**，上传成功后，开发者即可在开发平台对上传版本进行后续操作，如体验、上线等。

10. 详情面板：开发者可以在详情面板进行以下操作

- 切换、编辑或者复制 AppID。
- 打开本地目录，查看本地代码大小。
- 在项目设置里面查看当前小程序使用的基础库，以及是否对代码进行转义、自动补全、代码压缩混淆、上传携带 sourceMap、使用 npm 模块、以及是否校验请求的域名白名单。
- 在域名信息里，开发者可以查看当前业务设置的业务域名以及请求域名，同时可以刷新设置。

> **说明：**
> **下文对详情面板内的字段进行了说明解读。**

- **调试基础库：**开发者可根据需求切换对应小程序基础库，不同的基础库版本对应了不同的能力。新建项目默认使用最新版，建议使用最新版。
- **ES6 转 ES5：**开启此选项后，开发者工具将使用 babel 将 JS 代码编译成符合 ES5 标准的代码，以便在低版本手机系统上运行。
- **上传代码样式补全：**开启此选项，开发工具会自动检测并补全缺失样式，保证在低版本系统上的正常显示。尽管可以规避大部分的问题 ，还是建议开发者需要在 iOS 和 Android 上分别检查小程序的真实表现。
- **上传代码时自动压缩混淆：**开启此选项，开发者工具在上传代码时候将会帮助开发者压缩和混淆 javascript 代码，减小代码包体积。
- **预览/上传时携带 sourcemap：**是否上传小程序编译后的 sourceMap 文件，方便正式包报错代码分析。
- **上传时进行代码保护：**开启此选项，开发者工具会尝试对项目代码进行保护，主要是对文件进行扁平化处理并替

换 require 引用的文件名，对于小程序只有简单页面的情况下，开启此功能效果不佳，**主要有以下几种情况：**

- 有文件超过 500kb，且其中有使用 require 引用项目中的文件；
- 在运行时可能会报文件没有找到，动态引用的情况，如 var a = 'somefile.js'; require(a)；
- 将 require 函数赋值给其他变量的情况，如 var a = require; a('somefile.js')；
- 将 require 作为二元运算符的参数的情况，如 require + 1; 使用 ... 运算符且未开启 ES6 转 ES5；

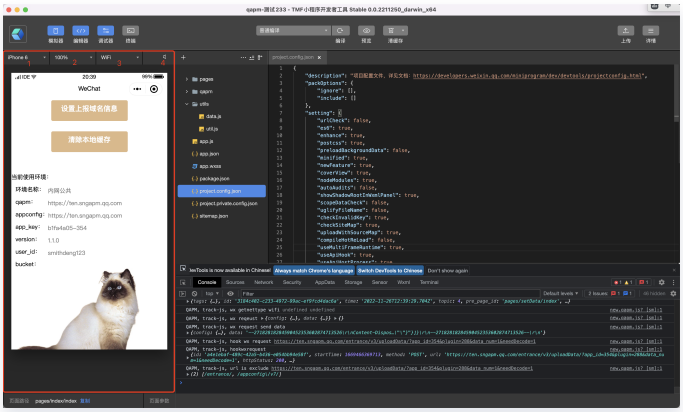
> **建议使用如下操作：**
>
> - **使用 npm 模块:** --build-npm [project_root]: 按照 `project.config.json` 中的配置构建，行为和菜单栏的构建 NPM 一致。
> - **不校验合法域名、web-view(业务域名):** 在 IDE 中忽略合法域名和 web-view 域名的校验。

小程序预览

预览区域是小程序编译加载后展示的区域，开发者可以在此模拟操作小程序，测试其功能和能力。

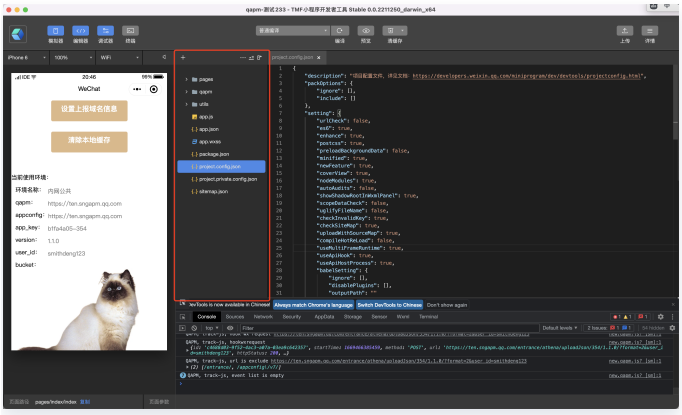
在预览区域中，您可以进行以下操作：

- 切换不同机型，预览小程序在不同设备上的效果。
- 缩放屏幕比例，以便更好地查看小程序的展示效果。
- 设置当前运行的网络环境，以模拟不同网络环境下小程序的表现。
- 开启或关闭声音。



文件目录树

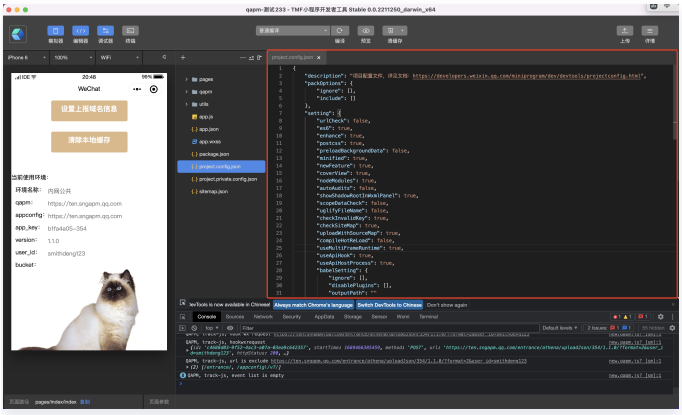
文件树是一款方便开发者进行文件管理的工具，可以对打开的文件夹进行新增、删除、打开目录以及折叠、展开文件树等操作。



编辑器

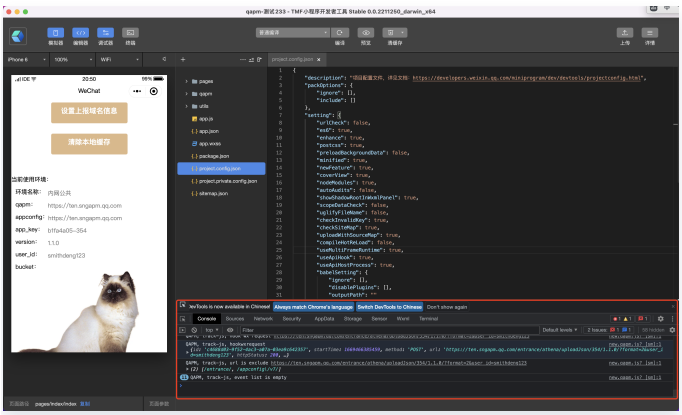
在 IDE 编辑器主页面，您可以看到具有丰富功能的编辑器窗口，其中包含了代码编写，调试及版本控制等功能，方便您的项目开发及管理。

其中编辑器窗口除了代码编写外，还支持代码格式化、语法高亮、智能提示、快捷键设置。



调试区

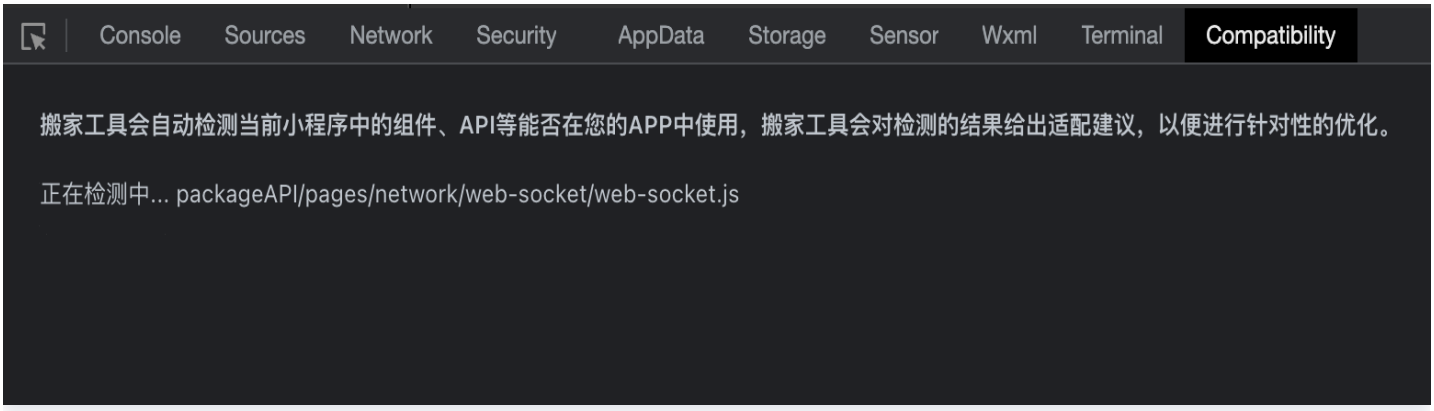
调试区功能类似于 Chrome 的 devtools 工具，开发者可以在此进行日志查看、断点调试、网络请求审查以及本地存储信息查看等操作。



小程序搬家工具 Compatibility

您可以在 **Compatibility** 处，发起微信小程序兼容性检测，单击**开始检测**。

搬家工具会遍历当前小程序中的组件、API、json 文件等，检测完成后，在下方窗口会展示检测结果，可将检测结果 excel 文件导出。



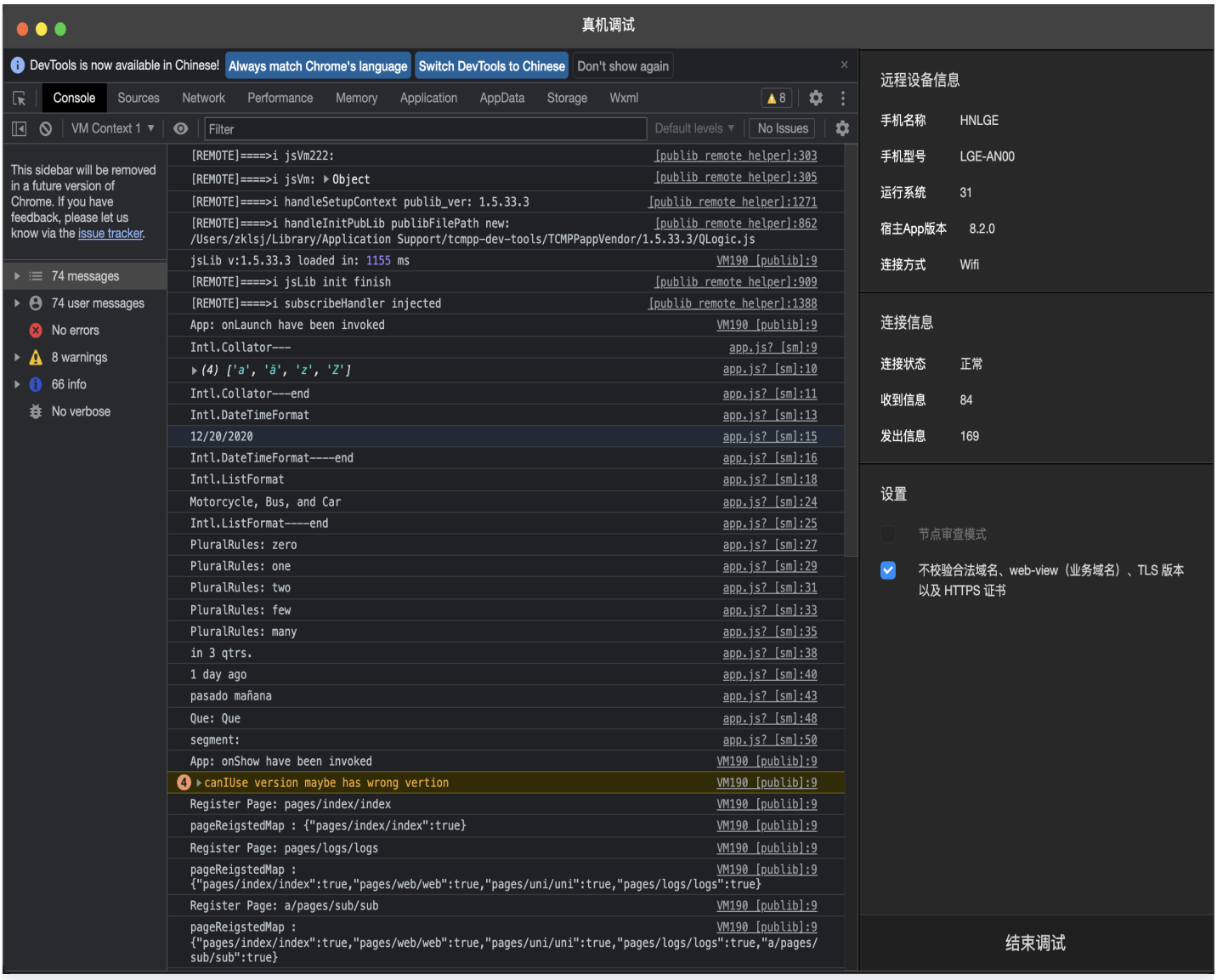
小程序真机调试能力

真机调试入口：

单击真机调试，此时，工具会将本地代码进行处理打包并上传，就绪之后，使用手机客户端扫描二维码即可弹出调试窗口，开始真机调试。

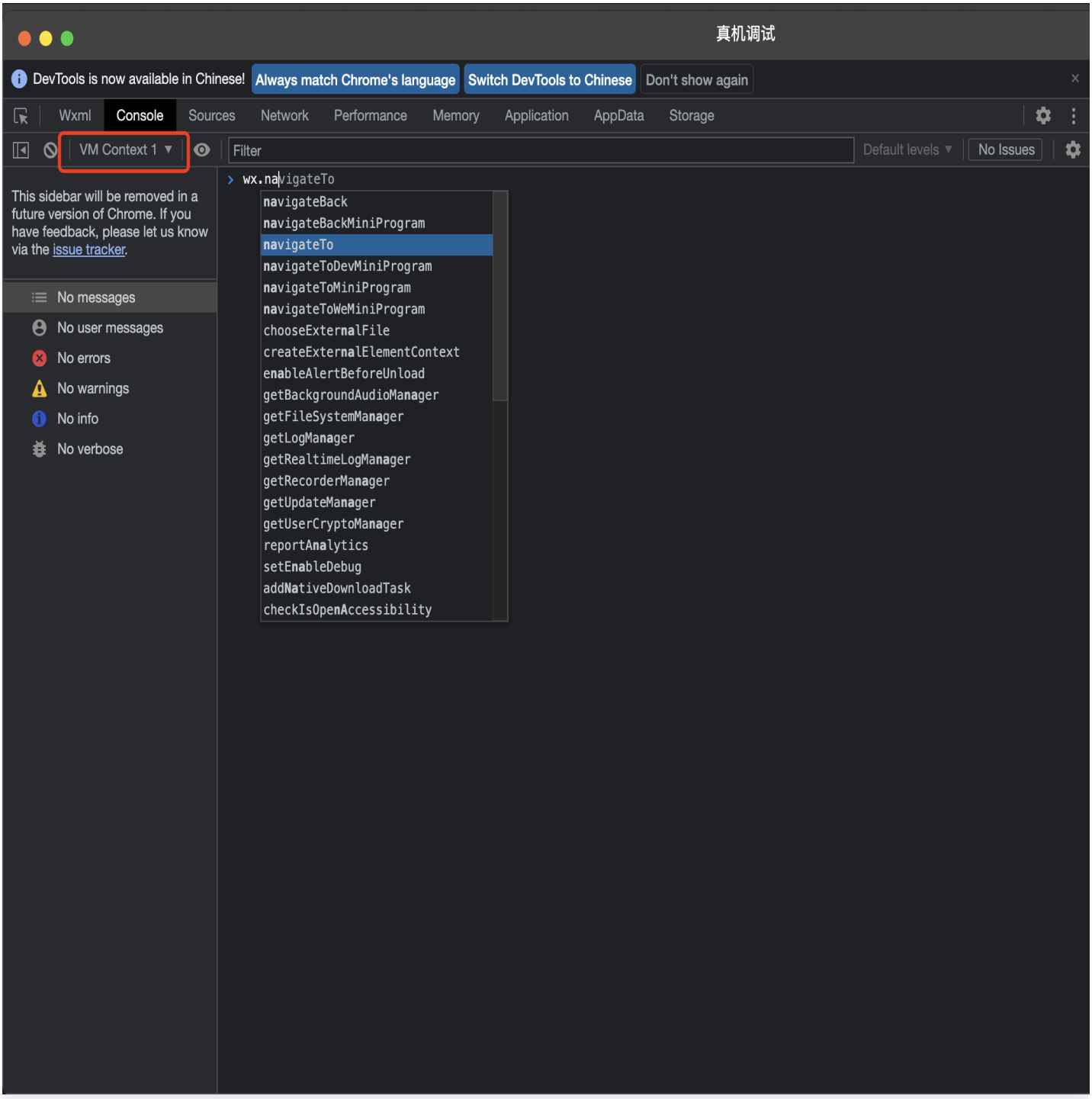
> **注意：**
> *使用真机调试 Android、iOS SDK版本需要大于 1.5.12。*

使用接入 SDK 的 App 扫描此二维码，即可开始远程调试。
要结束调试，直接关闭此调试窗口，或单击右下角 “结束调试” 按钮即可。



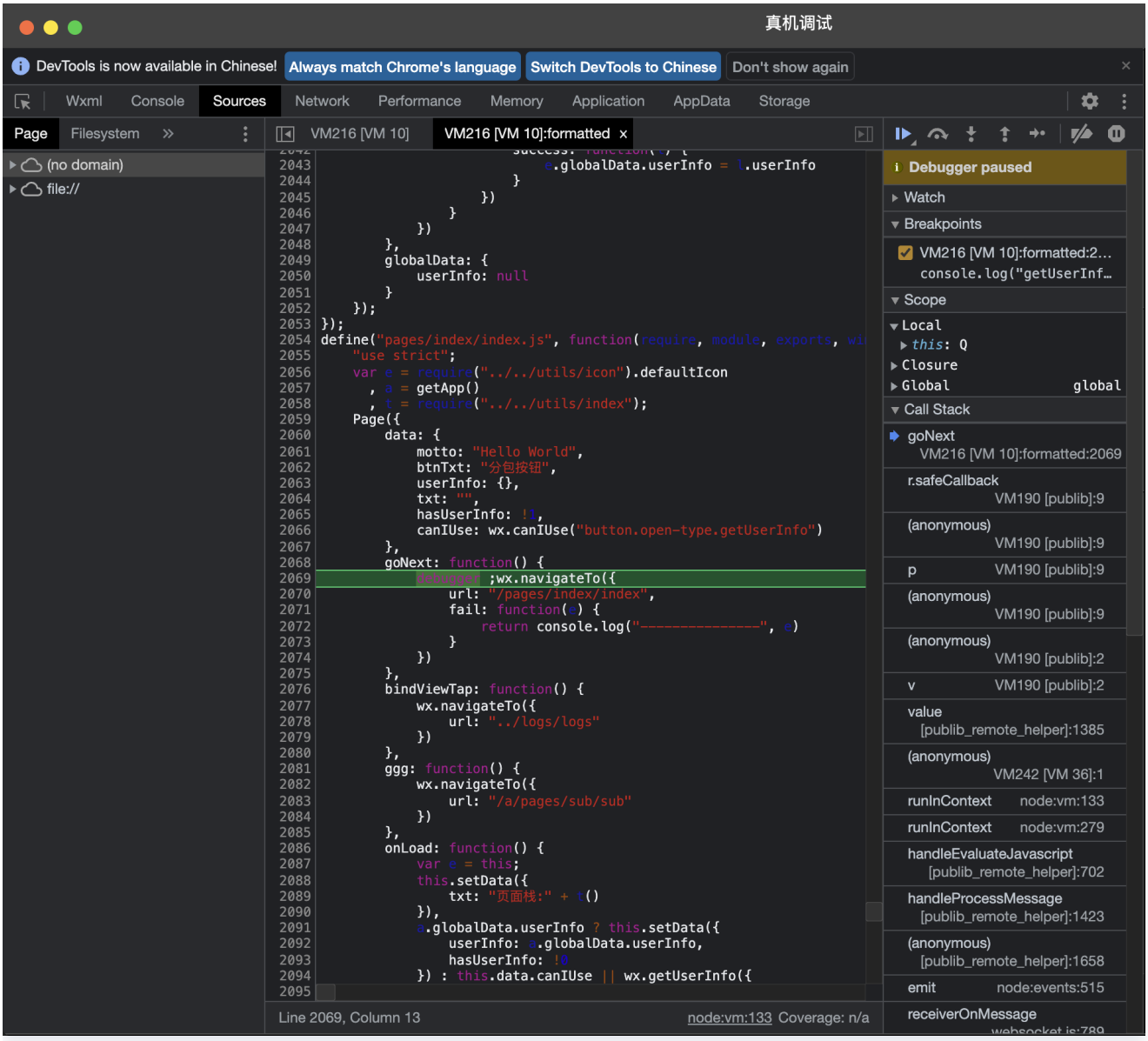
远程调试窗口分为两部分，分别是左侧的调试器视图、右侧的信息视图。开发者可以在调试器里直接进行代码的调试，并查看 Storage 情况；信息视图则可以查看目前与手机和服务器的连接情况，以及发生的错误信息等。

在远程调试的调试器里，开发者可以在 Console 面板里对代码进行调试，在 Sources 面板里查看小程序的源代码并进行断点单步调试，在 Storage 面板里查看小程序的 Storage 使用情况等。

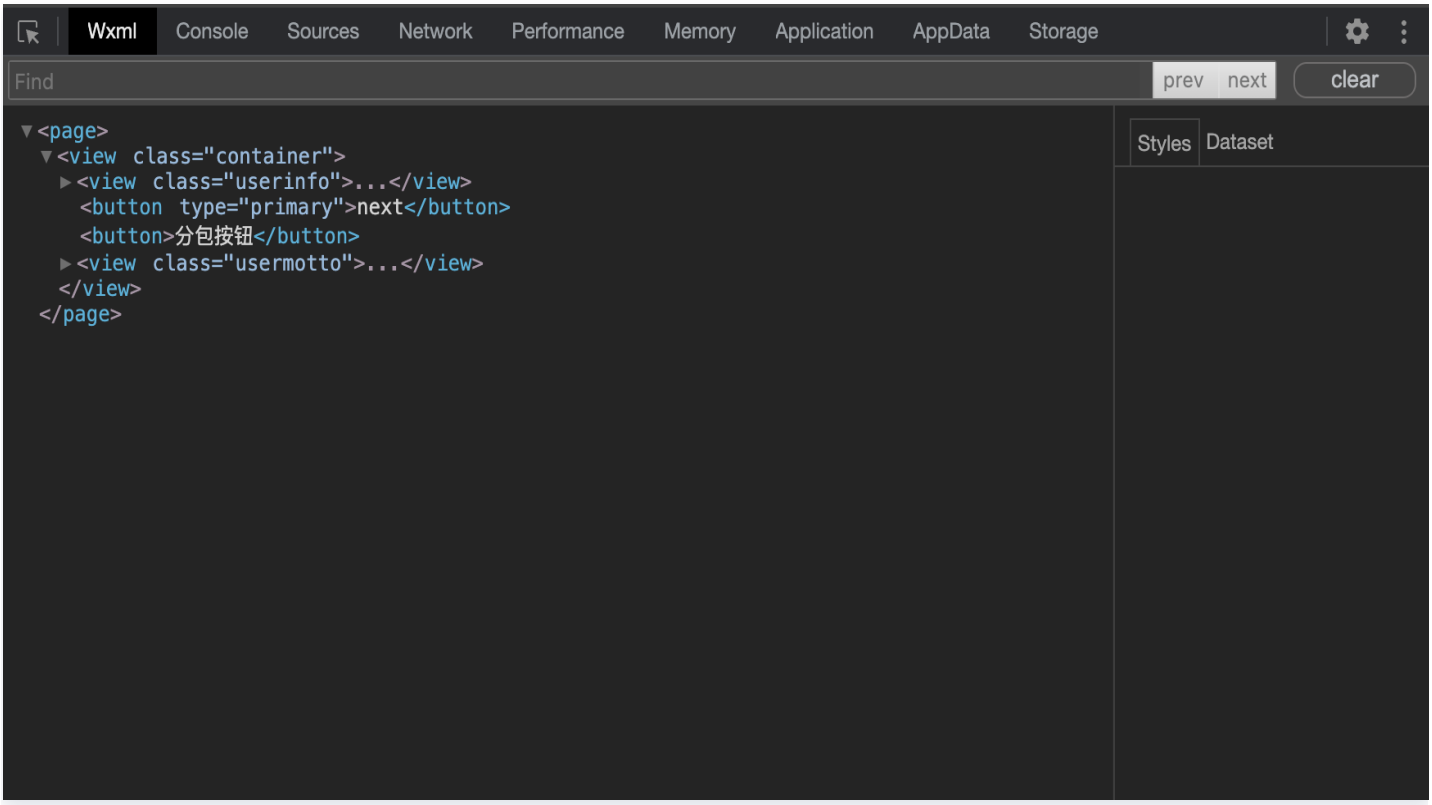


对于 JS 的断点调试，在代码里使用 debugger。

```
goNext(){
  debugger;
  wx.navigateTo({
    url:'/pages/index/index',
    fail: err => console.log("---------------",err),
  })
},
// event handler function
```



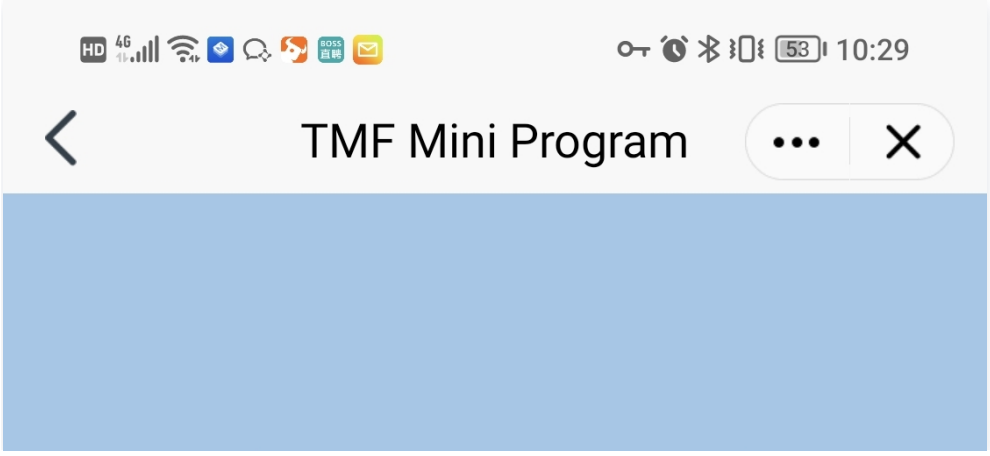
wxml 面板：



右侧的信息视图展示了手机、网络连接的信息。手机信息展示手机的型号、系统、名称、宿主 App 版本等信息，以及通信延时。通信延时越小，与手机的通信越流畅，打开节点审查模式，可以在真机按钮上点击元素查看元素详情。


远程设备信息
手机名称 HNLGE
手机型号 LGE-AN00
运行系统 31
宿主App版本 8.2.0
连接方式 Wifi
连接信息
连接状态 连接出错
收到信息 171
发出信息 400
警告和错误
类型: 真机断开连接 10:30:27
设置
节点审查模式
不校验合法域名、web-view（业务域名）、TLS 版本以及 HTTPS 证书
结束调试

手机端展示：


10:29
TMF Mini Program


getUserInfo
next
分包按钮
已连接
待确认 0
已发送 170
已接收 166
结束
收起
orld
page
377.02 x 547.15
vConsole