ALIBABA CLOUD

阿里云

控制台

小程序开发文档

文档版本：20220519

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

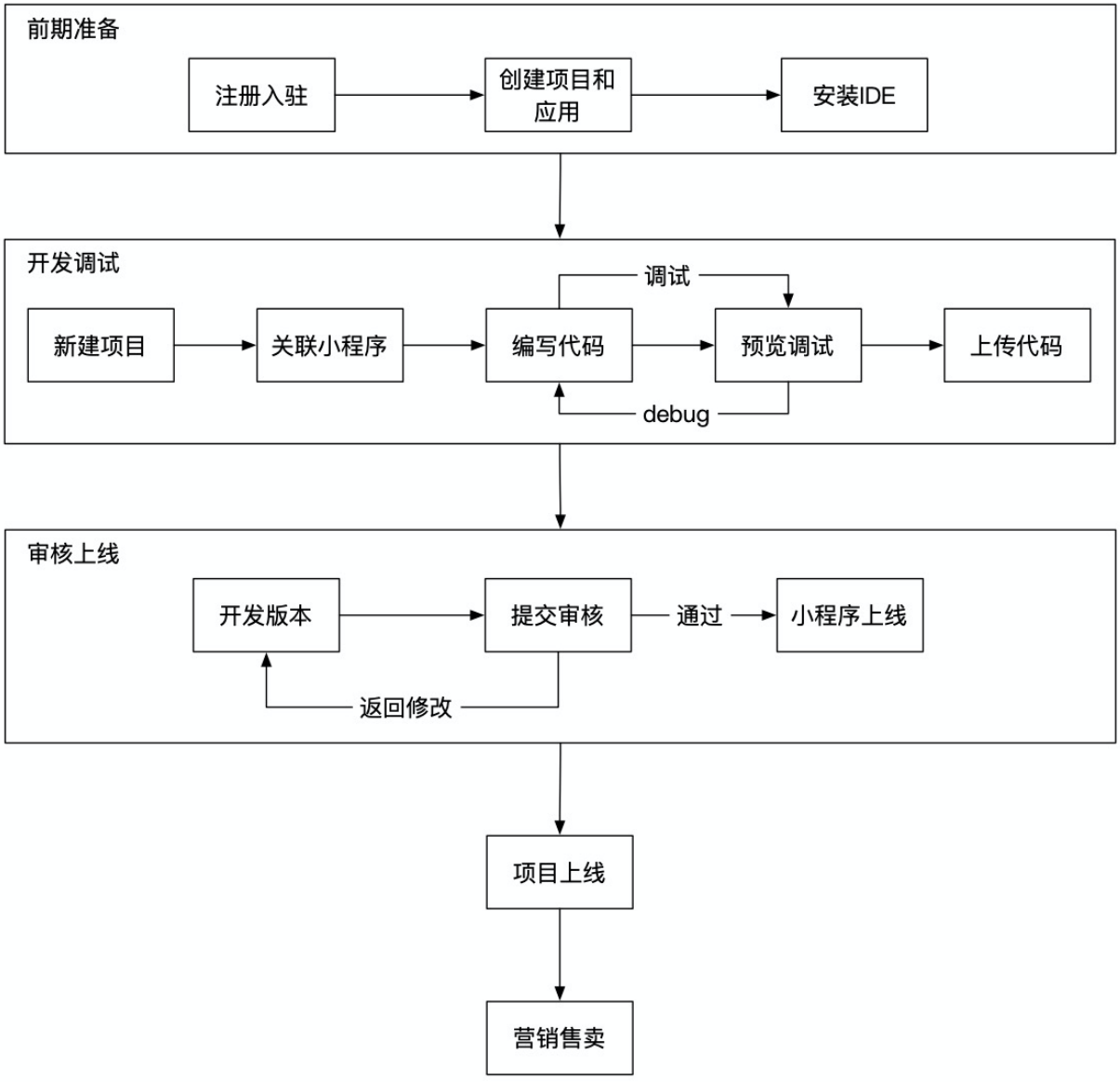
格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.整体介绍

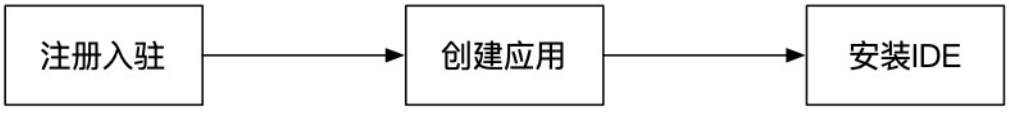
阿里云工具应用是一种全新的开放模式，让合作伙伴有机会分享阿里云多端流量和商业能力，为用户提供更好的体验，助力企业经营升级。

生命周期



前期准备

在开发工具应用之前，您需要完成下列准备工作：



1. 注册入驻

参看「入驻指南」

2. 创建项目以及工具应用

参看「项目的创建和管理」

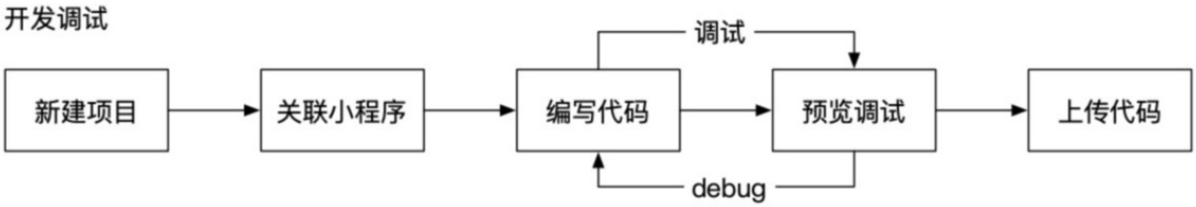
3. 安装开发工具

下载并安装 小程序开发者工具（简称 IDE）的最新版本（>= v1.14）

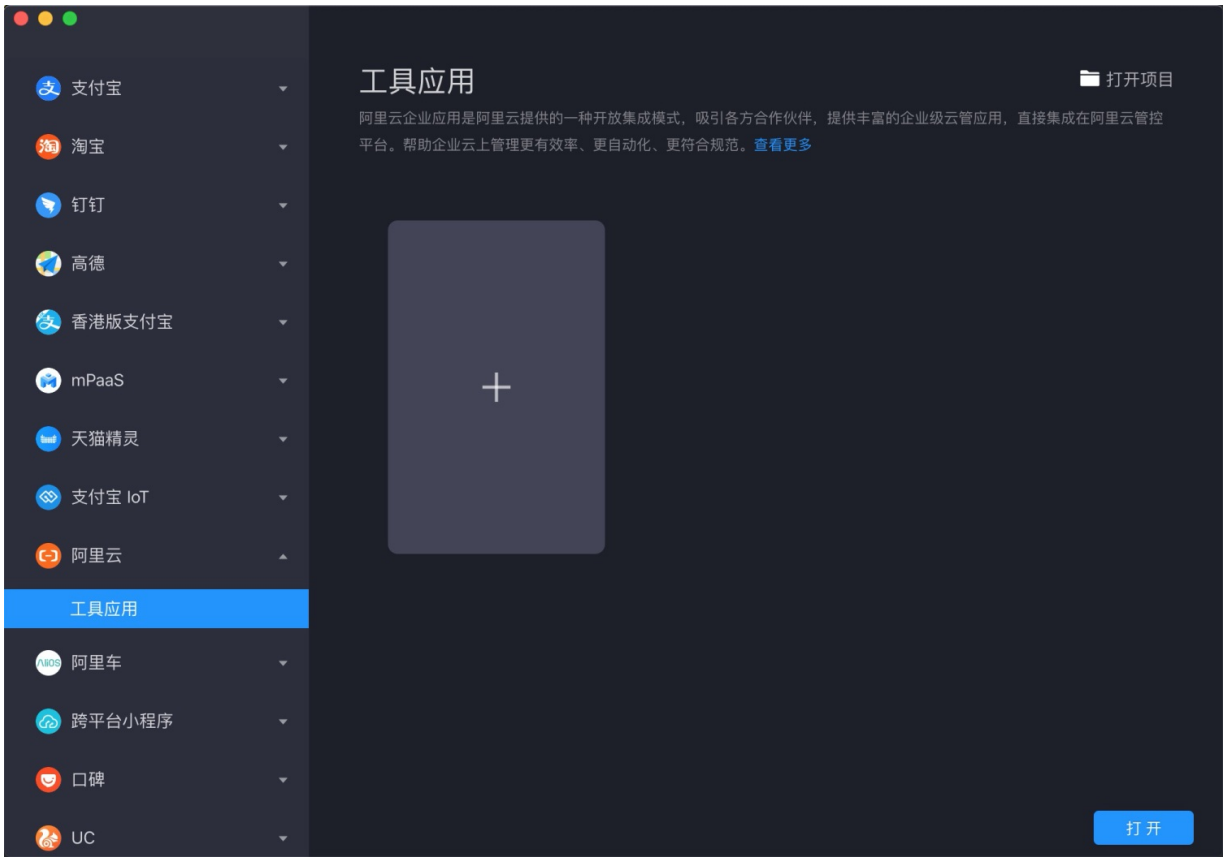
- Windows 7/8/10（64-bit）
- macOS

开发调试

工具应用的开发工作在 IDE 中展开。以下是开发环节的基本流程：



1. 新建工具应用
   i. 启动 IDE，在左侧边栏选择 阿里云 > 工具应用
   ii. 点击 ▯ 号，创建一个新的本地项目



2. 关联小程序

进入主界面后，IDE 通过弹窗提示 **扫码登录** 以及 **关联小程序**，按照提示操作即可。每个账号可以拥有多个小程序的开发权限。上传本地代码之前，需要先将本地代码关联到后台的小程序。

小程序可以在阿里云 App 4.20+ 版本运行，请使用最新阿里云 App 预览调试。

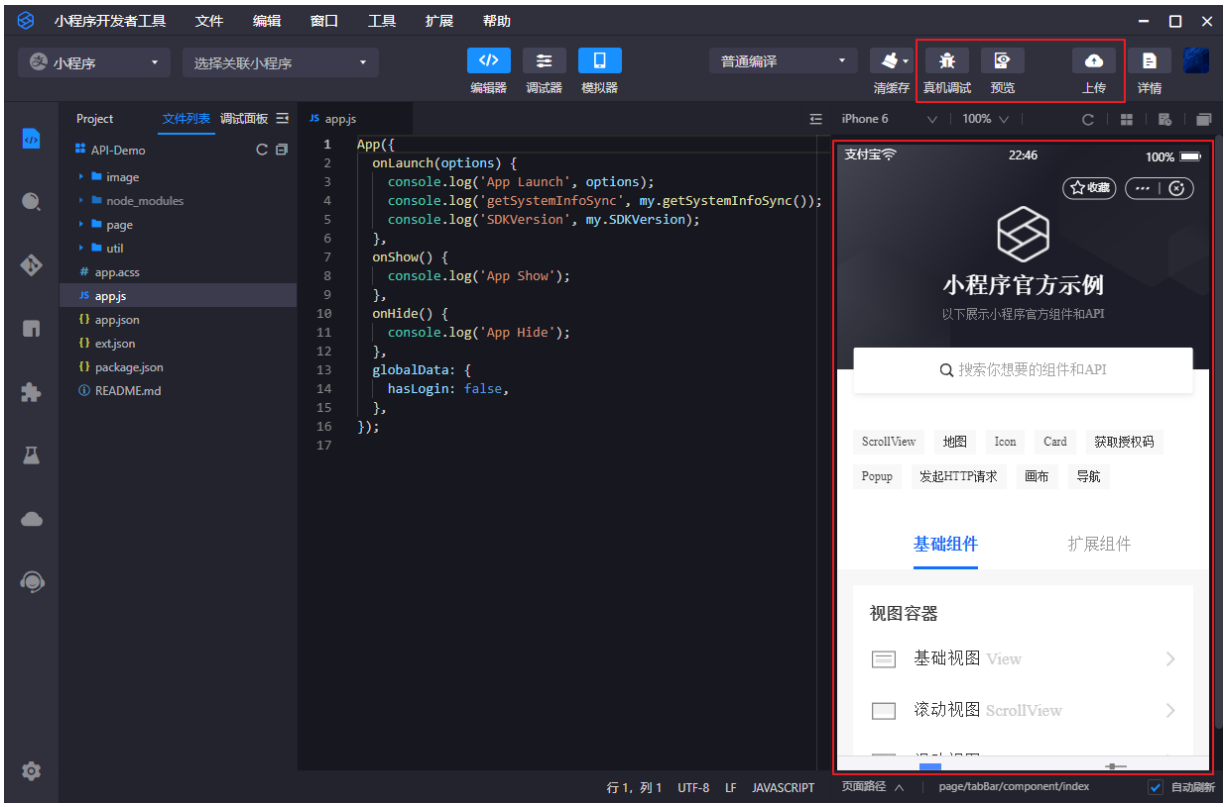
3. 编写代码

开发者需要熟悉 Web 前端基础知识。下列途径有助您了解如何开发小程序前端应用：

- 了解小程序的 框架、概览与 概览
- 了解小程序中如何调用后端服务，主要是官方托管以及三方自管的区别

4. 预览调试

在开发过程中，应用的运行效果会在主界面右侧的模拟器中显示。由于模拟器尚未支持全部 API，因此您还需要使用真机来预览与调试小程序的实际运行情况。



5. 上传代码

在完成开发之后，点击工具栏右侧的 **上传** 工具，在弹出菜单中再点击 **上传** 按钮，本地代码将会上传至后台。

6. 审核上线

应用代码成功上传之后，会对应一个开发版本，需要审核通过才能正式发布上线。

2.框架

2.1. 概述

文件结构

应用分为 `app` 和 `page` 两层。`app` 用来描述整个应用，`page` 用来描述各个页面。

`app` 由 3 个文件组成，必须放在项目的根目录。

文件	必需	作用
app.js	是	应用逻辑
app.json	是	应用全局设置
app.acss	否	应用全局样式表

每个页面 `page` 由 4 个文件组成，分别是：

文件类型	必需	作用
[page].js	是	页面逻辑
[page].axml	是	页面结构
[page].acss	否	页面样式表
[page].json	否	页面配置

为了方便组织您的代码，我们约定这 4 个文件必须具有相同的路径与文件名。

小程序的所有代码最终会被打包成一份 JavaScript 脚本，在应用启动的时候运行，在应用结束时销毁。

逻辑结构

应用的核心是一个响应式的数据绑定系统，分为视图层和逻辑层。这两层始终保持同步，只要在逻辑层修改数据，视图层就会相应的更新。

请看下面这个简单的例子。

```
<!-- 视图层 -->
<view> Hello {{name}}! </view>
<button onTap="changeName"> Click me! </button>
```

```
// 逻辑层
var initialData = {
  name: 'aliyun',
};
// 注册一个页面
Page({
  data: initialData,
  changeName(e) {
    // 改变数据
    this.setData({
      name: 'ali',
    });
  },
});
```

上面代码中，框架自动将逻辑层数据中的 `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 aliyun!` 。

用户点击按钮的时候，视图层会发送 `changeName` 的事件给逻辑层，逻辑层找到对应的事件处理函数。逻辑层执行了 `setData` 的操作，将 `name` 从 `aliyun` 变为 `ali` ，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 ali!` 。

注意：

逻辑层的 JS 可以用 ES2015 模块化语法组织代码：

```
import util from './util'; // 载入相对路径
import absolute from '/absolute'; // 载入项目根路径文件
```

小程序框架运行在非浏览器环境，所以 JavaScript 不能访问部分 Web 能力，如 `window` 、 `document` 等对象。这些内置对象名作 `保留字` 使用，以应对未来的不时之需。

有这些保留字：globalThis、global、AlipayJSBridge、fetch、self、window、document、location、XMLHttpRequest。

请不要使用这些保留字做模块名，否则会出现无法正常访问模块的现象。如：

```
import { window } from './myWindow'
console.log(window) // undefined
```

上述代码中，因为使用了保留字做模块名，使得引入的模块变成了 undefined 。正确的方法是不使用这些保留字命名模块，或者使用 as 关键字给模块重命名，例如：

```
import { window as myWindow } from './myWindow'
console.log(myWindow)
```

第三方 NPM 模块

应用支持引入第三方模块，需先在应用根目录下，打开命令行工具，执行如下命令安装该模块：

```
$ npm install lodash --save
```

引入后即可在逻辑层中直接使用：

```
import lodash from 'lodash'; // 载入第三方 npm 模块
```

注意：由于 node_modules 里第三方模块代码不会经过转换器，为了确保各个终端兼容，node_modules 下的代码需要转成 ES5 格式再引用，模块格式推荐使用 ES2015 的 import/export。类似前面的文档所述，浏览器部分 Web 能力无法使用。

2.2. 应用全局配置

2.2.1. 全局介绍

App 代表顶层应用，管理所有页面和全局数据，以及提供生命周期回调等。它也是一个构造方法，生成 App 实例。

一个应用就是一个 App 实例。

每个应用顶层一般包含三个文件。

- app.json：应用配置
- app.js：应用逻辑
- app.acss：应用样式（可选）

简单示例

一个简单的 app.json 代码如下：

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

这段代码配置指定应用包含两个页面（index 和 logs），以及应用窗口的默认标题设置为 “Demo”。

一个简单的 app.js 代码如下：

```
App({
  onLaunch(options) {
    // 第一次打开
  },
  onShow(options) {
    // 应用启动，或从后台被重新打开
  },
  onHide() {
    // 应用从前台进入后台
  },
  onError(msg) {
    // 应用发生脚本错误或 API 调用出现报错
    console.log(msg);
  },
  globalData: {
    // 全局数据
    name: 'aliyun',
  },
});
```

2.2.2. app.json 全局配置

`app.json` 是应用的全局配置，设置页面文件的路径、窗口表现、网络超时时间、多 tab 等。

以下是一个基本配置示例：

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

完整配置项如下：

属性	类型	是否必填	描述
pages	Array	是	设置页面路径
window	Object	否	设置默认页面的窗口表现
tabBar	Object	否	设置底部 tabbar 的表现

pages

app.json 中的 `pages` 为数组属性，数组中每一项都是字符串，用于指定应用的页面。在应用中新增或删除页面，都需要对 `pages` 数组进行修改。

`pages` 数组的每一项代表对应页面的路径信息，其中，第一项代表应用的首页。

页面路径不需要写任何后缀，框架会自动去加载同名的 `.json` 、 `.js` 、 `.axml` 、 `.acss` 文件。举例来说，如果开发目录为：

```
├── pages
│   ├──index
│   │   ├── index.json
│   │   ├── index.js
│   │   ├── index.axml
│   │   └── index.acss
│   ├──logs
│   │   ├── logs.json
│   │   ├── logs.js
│   │   └── logs.axml
├── app.json
├── app.js
└── app.acss
```

`app.json` 就要写成下面的样子：

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

window

属性	类型	是否必填	描述
defaultTitle	String	否	页面title标题，需在非首页使用。必须在navigationBarForceEnable设置开启才能展示。使用了tabbar的非首个tab不需要开navigationBarForceEnable也能展示。
pullRefresh	String	否	是否允许下拉刷新。默认NO, 备注：下拉刷新生效的前提是allowsBounceVertical 值为 YES

allowsBounceVertical	String	否	是否允许向下拉拽。默认 `YES`，支持 `YES` / `NO`
transparentTitle	String	否	导航栏透明设置。默认 `none`，支持 `always` 一直透明 / `auto` 滑动自适应 / `none` 不透明
navigationBarTextStyle	String	否	导航栏标题颜色，仅支持 black/white
titleImage	String	否	导航栏图片地址
titleBarColor	HexColor	否	导航栏背景色
showNavigationBarLogo	Boolean	否	是否显示首页导航栏上的 AppLogo，默认显示。
navigationBarForceEnable	Boolean	否	默认false，若开启，则设置的二级页头部标题可展示。

下面是一个例子：

```
{
  "window":{
    "defaultTitle": "接口功能演示"
  }
}
```

tabBar

如果你的应用是一个多 tab 应用（客户端窗口的底部栏可以切换页面），那么可以通过 `tabBar` 配置项指定 tab 栏的表现，以及 tab 切换时显示的对应页面。

注意：

- 通过页面跳转（ `my.navigateTo` ）或者页面重定向（ `my.redirectTo` ）所到达的页面，即使它是定义在 tabBar 配置中的页面，也不会显示底部的 tab 栏。

- `tabBar` 的第一个页面必须是首页。

tabBar 配置项有以下：

属性	类型	是否必填	描述
textColor	HexColor	否	文字颜色
selectedColor	HexColor	否	选中文字颜色
backgroundColor	HexColor	否	背景色
items	Array	是	每个 tab 配置

每个 item 配置：

属性	类型	是否必填	描述
pagePath	String	是	设置页面路径
name	String	是	名称
icon	String	否	平常图标路径
activeIcon	String	否	高亮图标路径

icon 图标推荐大小为 60×60 px 大小，系统会对传入的非推荐尺寸的图片进行非等比拉伸或缩放。

tabBar 示例如下：

```
{
  "tabBar": {
    "textColor": "#dddddd",
    "selectedColor": "#49a9ee",
    "backgroundColor": "#ffffff",
    "items": [
      {
        "pagePath": "pages/index/index",
        "name": "首页"
      },
      {
        "pagePath": "pages/logs/logs",
        "name": "日志"
      }
    ]
  }
}
```

2.2.3. app.acss 全局样式

`app.acss` 作为全局样式定义应用全局的样式，有关 acss 更详细的文档请参见 ACSS 语法参考。

2.2.4. app.js 全局逻辑

App(object: Object)

`App()` 用于注册应用，接受一个 Object 作为属性，用来配置应用的生命周期等。

`App()` 必须在 app.js 中调用，必须调用且只能调用一次。

object 属性说明

属性	类型	描述	触发时机
onLaunch	Function	生命周期回调：监听应用初始化	当应用初始化完成时触发，全局只触发一次
onShow	Function	生命周期回调：监听应用显示	当应用启动，或从后台进入前台显示时触发
onHide	Function	生命周期回调：监听应用隐藏	当应用从前台进入后台时触发
onError	Function	监听应用错误	当应用发生 js 错误时触发
onShareAppMessage	Function	全局分享配置	

前台/后台定义：

- 应用用户点击右上角关闭，或者按下设备 Home 键离开淘宝时，应用并不会直接销毁，而是进入后台。
- 当用户再次进入阿里云或再次打开应用时，应用会从后台进入前台。
- 只有当应用进入后台一定时间，或占用系统资源过高，才会被真正销毁。

onLaunch(object: Object) 及 onShow(object: Object)

object 属性说明：

属性	类型	描述
query	Object	当前应用的 query，从启动参数的 query 字段解析而来
path	String	当前应用的页面地址，从启动参数 page 字段解析而来，page 忽略时默认为首页
referrerInfo	Object	来源信息

比如，启动应用的 URL 如下：

```
https://m.duanqu.com/?_ariver_appid=20000067&query=number%3D1&path=x%2Fy%2Fz
```

参数解析如下：

```
query = decodeURIComponent('number%3D1');
// number=1
path = decodeURIComponent('x%2Fy%2Fz');
// x/y/z
```

- 应用首次启动时，`onLaunch` 方法可获取 `query` 、`path` 属性值。
- 应用在后台被用 schema 打开，也可从 `onShow` 方法中获取 `query` 、`path` 属性值。

```
App({
  onLaunch(options) {
    // 第一次打开
    console.log(options.query);
    // {number:1}
    console.log(options.path);
    // x/y/z
  },
  onShow(options) {
    // 从后台被 schema 重新打开
    console.log(options.query);
    // {number:1}
    console.log(options.path);
    // x/y/z
  },
});
```

referrerInfo 子属性说明：

属性	类型	描述	兼容性
appId	string	来源应用	
sourceServiceId	String	来源插件，当处于插件运行模式时可见	
extraData	Object	来源应用传过来的数据。	

注意：

- 不要在 `onShow` 中进行 `redirectTo` 或 `navigateTo` 等操作页面栈的行为。
- 不要在 `onLaunch` 里调用 getCurrentPages 方法，因为此时 page 还未生成。

onHide()

应用从前台进入后台时触发 `onHide()` 。

示例代码：

```
App({
  onHide() {
    // 进入后台时
    console.log('app hide');
  },
});
```

onError(error: String)

应用发生脚本错误或 API 调用报错时触发。

```
App({
  onError(error) {
    // 应用执行出错时
    console.log(error);
  },
});
```

onShareAppMessage(object: Object)

全局分享配置。当页面未设置 `page.onShareAppMessage` 时，调用分享会执行全局的分享设置，具体见分享。

globalData 全局数据

`App()` 中可以设置全局数据 `globalData` 。

下面是一个例子：

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

2.2.5. getApp 方法

应用提供了全局的 `getApp()` 方法，可获取当前应用实例，一般用于在子页面中获取顶层应用。

```
var app = getApp();
console.log(app.globalData); // 获取 globalData
```

使用过程中，请注意以下几点：

- `App()` 函数中不可以调用 `getApp()` ，可使用 `this` 可以获取当前应用实例。
- 通过 `getApp()` 获取实例后，请勿私自调用生命周期回调函数。
- 请区分全局变量及页面局部变量，比如：

```
// a.js
// localValue 只在 a.js 有效
var localValue = 'a';
// 获取 app 实例
var app = getApp();
// 拿到全局数据，并改变它
app.globalData++;
```

```
// b.js
// localValue 只在 b.js 有效
var localValue = 'b';
// 如果 a.js 先运行，globalData 会返回 2
console.log(getApp().globalData);
```

`a.js` 和 `b.js` 两个文件中都声明了变量 `localValue` ，但并不会互相影响，因为各个文件声明的局部变量和函数只在当前文件下有效。

2.3. 应用页面

2.3.1. 应用页面介绍

Page 代表应用的一个页面，负责页面展示和交互。每个页面对应一个子目录，一般有多少个页面，就有多少个子目录。它也是一个构造函数，用来生成页面实例。

每个小程序页面一般包含四个文件。

- `[page].js` ：页面逻辑
- `[page].axml` ：页面结构
- `[page].acss` ：页面样式（可选）
- `[page].json` ：页面配置（可选）

页面初始化时，提供数据。

```
Page({
  data: {
    title: 'Aliyun',
    array: [{user: 'li'}, {user: 'zhao'}],
  },
});
```

根据以上提供的数据，渲染页面内容。

```
<view>{{title}}</view>
<view>{{array[0].user}}</view>
```

定义交互行为时，需要指定响应函数。

```
<view onTap="handleTap">click me</view>
```

以上代码指定用户触摸按钮时，调用 `handleTap` 方法。

```
Page({
  handleTap() {
    console.log('yo! view tap!');
  },
});
```

页面重新渲染，需要在页面脚本里面调用 `this.setData` 方法。

```
<view>{{text}}</view>
<button onTap="changeText"> Change normal data </button>
```

以上代码指定用户触摸按钮时，调用 `changeText` 方法。

```
Page({
  data: {
    text: 'init data',
  },
  changeText() {
    this.setData({
      text: 'changed data',
    });
  },
});
```

上面代码中，`changeText` 方法里面调用 `this.setData` 方法，会导致页面重新渲染。

2.3.2. 页面配置

在 `/pages` 目录中的 .json 文件用于配置当前页面的窗口表现。页面配置比 `app.json` 全局配置简单得多，只能设置 `window` 相关配置项，但无需写 `window` 这个键。页面配置项会优先于全局配置项。

`window` 配置项同 app.json 全局配置，同时支持以下几点：

- 支持 `optionMenu` 配置导航图标，点击后触发 `onOptionMenuClick` 。但注意： `optionMenu` 配置将被废弃，建议使用 `my.setOptionMenu` 设置导航栏图标。
- 支持 `titlePenetrate` ，设置导航栏点击穿透。
- 支持 `barButtonTheme` ，设置导航栏图标主题。

完整配置项如下：

属性名	类型	必填	描述
optionMenu	Object	否	设置导航栏额外图标，目前支持设置属性 icon，值为图标 url（以 https/http 开头）或 base64 字符串，大小建议 30*30 px
titlePenetrate	BOOL	否	设置导航栏点击穿透
barButtonTheme	String	否	设置导航栏图标主题。“default”为蓝色图标，“light”为白色图标

以下为一个基本示例：

```
{
  "optionMenu": {
    "icon": "https://img.alicdn.com/tps/i3/T1OjaVFl4dXXa.JOZB-114-114.png"
  },
  "titlePenetrate": "YES",
  "barButtonTheme": "light"
}
```

2.3.3. 页面结构

在 `/pages` 目录中的 .axml 文件用于定义当前这个页面的结构，文件内容遵循 AXML 语法。

AXML 和 HTML 非常相似，但是也有很多不一样的地方，更详细的文档可以参考 AXML 介绍。

2.3.4. 页面样式

在 `/pages` 目录中的 .acss 文件用于定义页面样式。

每个页面中的根元素为 `page`，需要设置页面高度或背景色时，可按如下方式：

```
page {
  background-color: #fff;
}
```

有关 ACSS 更详细的文档请参见 ACSS 语法参考 。

2.3.5. 页面运行机制

Page(object: Object)

在 `/pages` 目录的 .js 文件中，定义 `Page()`，用于注册一个小程序页面，接受一个 object 作为属性，用来指定页面的初始数据、生命周期回调、事件处理等。

以下为一个基本的页面代码：

```
// pages/index/index.js
Page({
  data: {
    title: "Aliyun",
  },
  onLoad(query) {
    // 页面加载
  },
  onShow() {
    // 页面显示
  },
  onReady() {
    // 页面加载完成
  },
  onHide() {
    // 页面隐藏
  },
  onUnload() {
    // 页面被关闭
  },
  onTitleClick() {
    // 标题被点击
  },
  onPullDownRefresh() {
    // 页面被下拉
  },
  onReachBottom() {
    // 页面被拉到底部
  },
  onShareAppMessage() {
   // 返回自定义分享信息
  },
  // 事件处理函数对象
  events: {
    onBack() {
      console.log('onBack');
    },
  },
  // 自定义事件处理函数
  viewTap() {
    this.setData({
      text: 'Set data for update.',
    });
  },
  // 自定义事件处理函数
  go() {
    // 带参数的跳转，从 page/ui/index 的 onLoad 函数的 query 中读取 type
    my.navigateTo({url:'/page/ui/index?type=mini'});
  },
  // 自定义数据对象
  customData: {
    name: 'aliyun',
  },
});
```

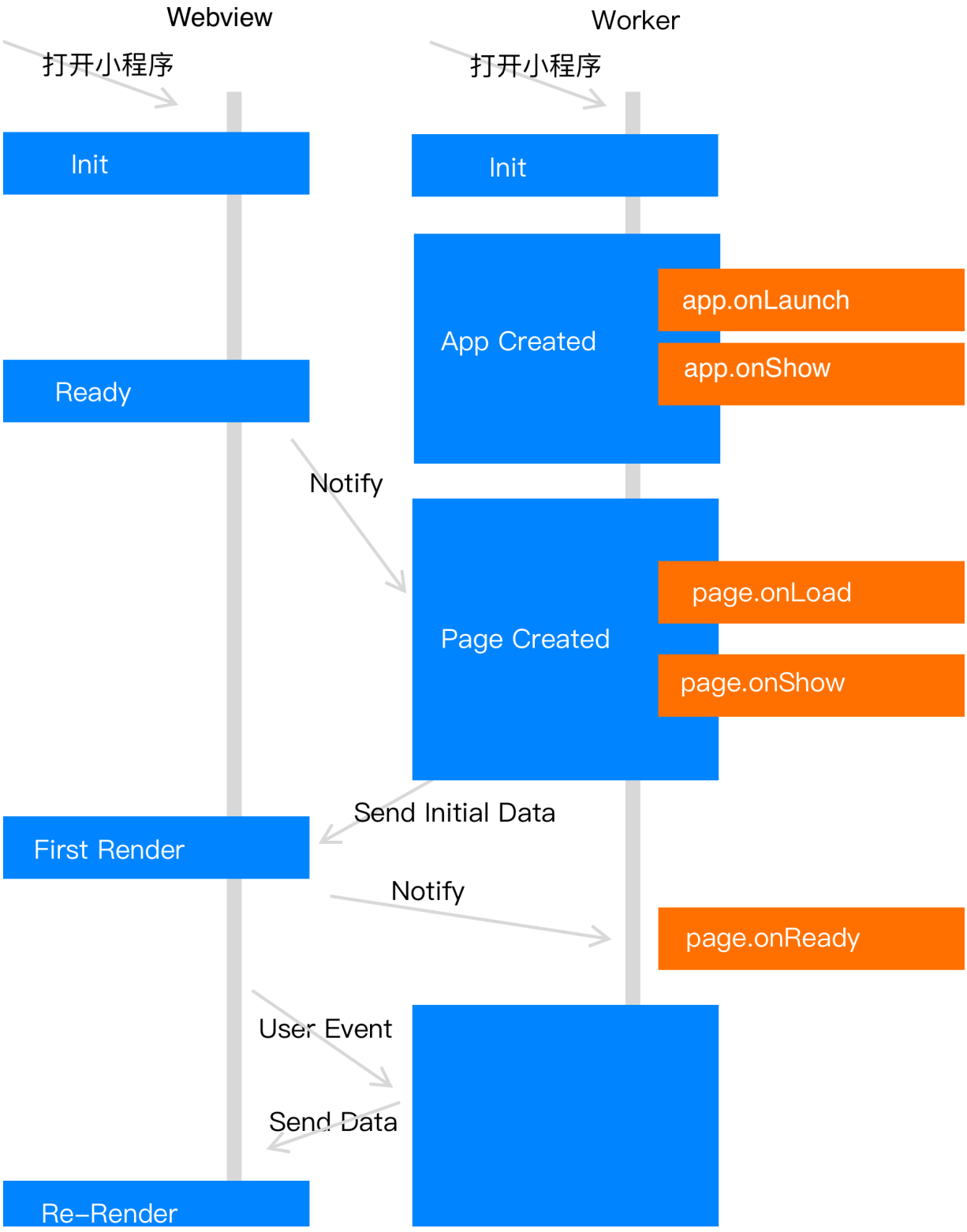
页面生命周期

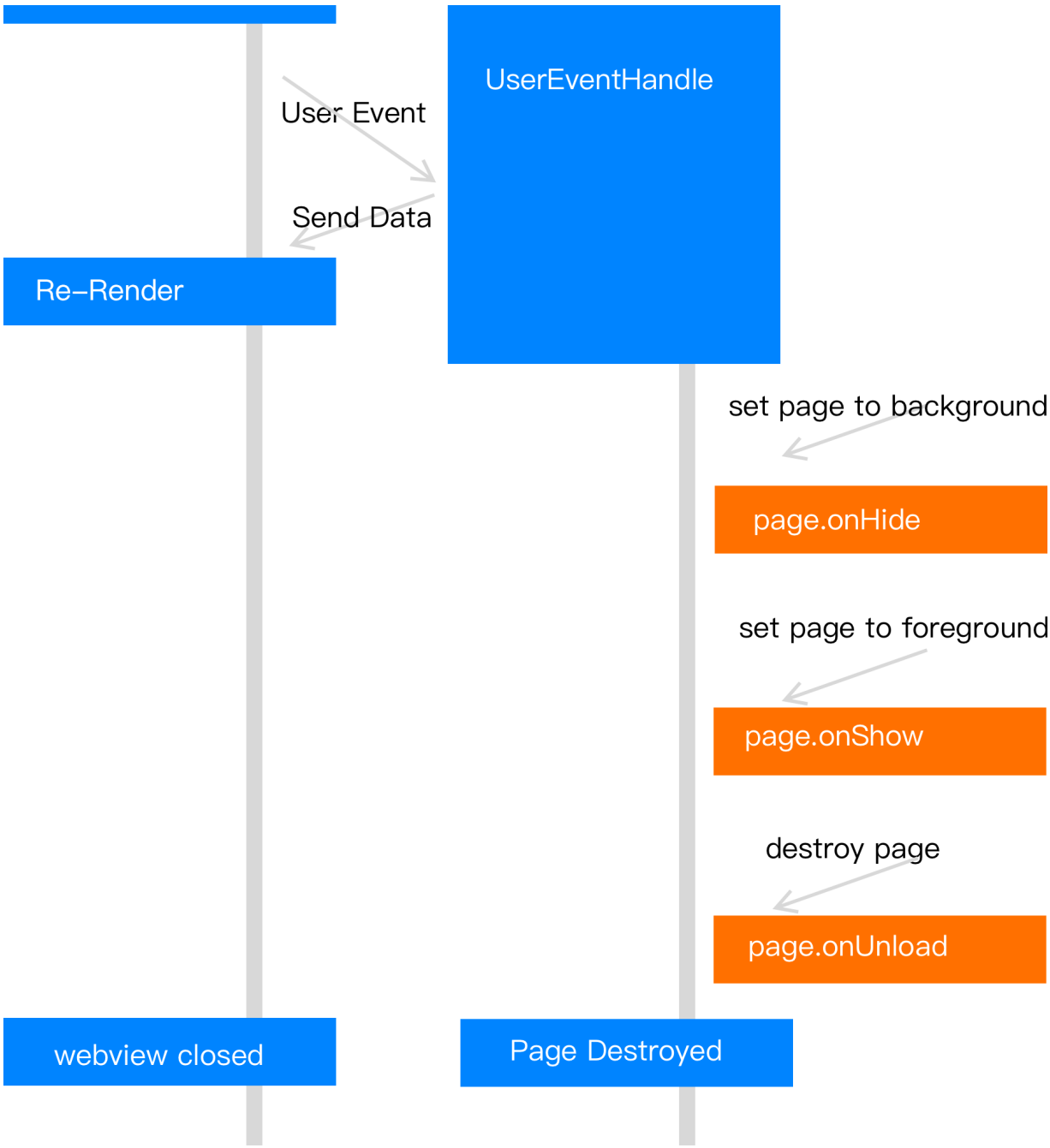
下图说明了页面 Page 对象的生命周期。

小程序主要靠视图线程（Webview）和应用服务线程（Worker）来控制管理。视图线程和应用服务线程同时运行。

- 应用服务线程启动后运行 app.onLauch 和 app.onShow 以完成 App 创建，再运行 page.onLoad 和 page.onShow 以完成 Page 创建，此时等待视图线程初始化完成通知。
- 视图线程初始化完成通知应用服务线程，应用服务线程将初始化数据发送给视图线程进行渲染，此时视图线程完成第一次数据渲染。
- 第一次渲染完成后视图线程进入就绪状态并通知应用服务线程，应用服务线程调用 page.onReady 函数并进入活动状态。

- 应用线程进入活动状态后每次数据修改将会通知视图线程进行渲染。当切换页面进入后台，应用线程调用page.onHide 函数后，进入存活状态；页面返回到前台将调用 page.onShow 函数，进入活动状态；当调用返回或重定向页面后将调用 page.onUnload 函数，进行页面销毁。





object 属性说明

属性	类型	描述
data	Object | Function	初始数据或返回初始化数据的函数。
events	Object	事件处理函数对象
onLoad	Function(query: Object)	页面加载时触发
onShow	Function	页面显示时触发

属性	类型	描述
onReady	Function	页面初次渲染完成时触发
onHide	Function	页面隐藏时触发
onUnload	Function	页面卸载时触发
onShareAppMessage	Function(options: Object)	点击右上角分享时触发
onTitleClick	Function	点击标题触发
onOptionMenuClick	Function	点击导航栏额外图标触发
onPopMenuClick	Function	点击右上角通用菜单中的自定义菜单按钮触发
onPullDownRefresh	Function({from: `manual` | `code` })	页面下拉时触发
onPullIntercept	Function	下拉中断时触发
onTabItemTap	Function	点击 `tabItem` 时触发
onPageScroll	Function({scrollTop})	页面滚动时触发
onReachBottom	Function	上拉触底时触发
其他	Any	开发者可以添加任意的函数或属性到 `object` 中，在页面的函数中可以用 `this` 来访问

页面数据对象 data

通过设置 `data` 指定页面的初始数据。当 `data` 为对象时，被所有页面共享。

即：当该页面回退后再次进入该页面时，会显示上次页面的数据，而非初始数据。这种情况，可以通过设置 `data` 为不可变数据或者变更 `data` 为页面独有数据两种方式来解决。

设置为不可变数据

```
Page({
 data: { arr:[] },
 doIt() {
   this.setData({arr: [...this.data.arr, 1]});
 },
});
```

设置页面独有数据（不推荐）

```
Page({
 data() { return { arr:[] }; },
 doIt() {
   this.setData({arr: [1, 2, 3]});
 },
});
```

注意：不要直接修改 `this.data`，无法改变页面的状态，还会造成数据不一致。

比如：

```
Page({
 data: { arr:[] },
 doIt() {
   this.data.arr.push(1); // 不要这么写!
   this.setData({arr: this.data.arr});
 }
});
```

生命周期函数

onLoad(query: Object)

页面初始化时触发。一个页面只会调用一次。

query 为 `my.navigateTo` 和 `my.redirectTo` 中传递的 query 对象。

query 内容格式为：“参数名=参数值&参数名=参数值...”。

属性	类型	描述
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。

对界面的设置，如 `my.setNavigationBar` 请在 `onReady` 之后设置。

onHide()

页面隐藏/切入后台时触发。

如 `my.navigateTo` 到其他页面或底部 tab 切换等。

onUnload()

页面卸载时触发。

如 `my.redirectTo` 或 `my.navigateBack` 到其他页面等。

页面事件处理函数

onShareAppMessage(options: Object)

点击右上角通用菜单中的分享按钮时或点击页面内分享按钮时触发。

onTitleClick()

点击标题时触发。

onOptionMenuClick()

点击右上角菜单按钮时触发。

onPopMenuClick()

点击右上角通用菜单按钮时触发。

onPullDownRefresh({from: `manual` | `code`})

下拉刷新时触发。

需要先在 app.json 全局配置 的 `window` 选项中开启 `pullRefresh` 。当处理完数据刷新后， `my.stopPullDownRefresh` 可以停止当前页面的下拉刷新。

onPullIntercept()

下拉中断时触发。

onTabItemTap(object: Object)

点击 `tabItem` 时触发。

属性	类型	描述
from	String	点击来源
pagePath	String	被点击tabItem的页面路径
text	String	被点击tabItem的按钮文字
index	Number	被点击tabItem的序号，从0开始

onPageScroll({scrollTop})

页面滚动时触发， `scrollTop` 为页面滚动距离。

onReachBottom()

上拉触底时触发。

events

为了使代码更加简洁，提供了新的事件处理对象 `events` 。

已有的页面处理事件函数跟直接在 page 实例上暴露的事件函数等价。

注意：

- `events`

从基础库

`1.13.7`

版本 开始支持。

- 请正确区分页面事件函数与

`events`

内同名事件函数的基础库版本要求。

以下是 events 支持的事件函数列表：

事件	类型	描述
onBack	Function	页面返回时触发
onKeyboardHeight	Function	键盘高度变化时触发
onOptionMenuClick	Function	点击右上角菜单按钮触发
onPopMenuClick	Function	点击右上角通用菜单按钮触发
onPullIntercept	Function	下拉截断时触发
onPullDownRefresh	Function({from: `manual` / `code` })	页面下拉时触发
onTitleClick	Function	点击标题触发
onTabItemTap	Function	点击非当前 `tabItem` 后触发
beforeTabItemTap	Function	点击非当前 `tabItem` 前触发
onResize	Function({size: {windowWidth: number, windowHeight: number}})	window尺寸改变时触发

示例代码：

```
// 特征检测
my.canIUse('page.events.onBack');
Page({
  data: {
    text: 'This is page data.'
  },
  onLoad(){
    // 页面加载时触发
  },
  events:{
    onBack(){
      // 页面返回时触发
    },
    onKeyboardHeight(e){
      // 键盘高度变化时触发
      console.log('键盘高度: ', e.height)
    },
    onOptionMenuClick(){
      // 点击右上角菜单按钮触发
    },
    onPopMenuClick(e){
      // 点击右上角通用菜单中的自定义菜单按钮触发
      console.log('用户点击自定义菜单的索引', e.index)
      console.log('用户点击自定义菜单的name', e.name)
      console.log('用户点击自定义菜单的menuIconUrl', e.menuIconUrl)
    },
    onPullIntercept(){
      // 下拉截断时触发
    },
    onPullDownRefresh(e){
      // 页面下拉时触发。e.from的值是"code"表示startPullDownRefresh触发的事件；值是"manual"表示用户下拉触发的下拉事件
      console.log('触发下拉刷新的类型', e.from)
      my.stopPullDownRefresh()
    },
    onTitleClick(){
      // 点击标题触发
    },
    onTabItemTap(e){
      // e.from是点击且切换tabItem后触发，值是"user"表示用户点击触发的事件；值是"api"表示switchTab触发的事件
      console.log('触发tab变化的类型', e.from)
      console.log('点击的tab对应页面的路径', e.pagePath)
      console.log('点击的tab的文字', e.text)
      console.log('点击的tab的索引', e.index)
    },
    beforeTabItemTap(){
      // 点击但切换tabItem前触发
    },
    onResize(e){
      // window尺寸改变时触发
      var {windowWidth, windowHeight} = e.size
      console.log('改变后window的宽度', windowWidth)
      console.log('改变后window的高度', windowHeight)
    },
  }
})
```

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

`setData` 会将数据从逻辑层发送到视图层，同时改变对应的 `this.data` 的值。

`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` 可以非常灵活，以数据路径的形式给出，如 `array[2].message`、`a.b.c.d`，可以不需要在 `this.data` 中预先定义。

使用过程中，需要注意以下几点：

1. 直接修改

`this.data`

无效，无法改变页面的状态，还会造成数据不一致。

2. 仅支持设置可 JSON 化的数据。
3. 请尽量避免一次设置过多的数据。
4. 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：

```
<view>{{text}}</view>
<button onTap="changeTitle"> Change normal data </button>
<view>{{array[0].text}}</view>
<button onTap="changeArray"> Change Array data </button>
<view>{{object.text}}</view>
<button onTap="changePlanetColor"> Change Object data </button>
<view>{{newField.text}}</view>
<button onTap="addNewKey"> Add new data </button>
<view>hello: {{name}}</view>
<button onTap="changeName"> Change name </button>
```

```
Page({
  data: {
    text: 'test',
    array: [{text: 'a'}],
    object: {
      text: 'blue',
    },
    name: 'ali',
  },
  changeTitle() {
    // 错误！不要直接去修改 data 里的数据
    // this.data.text = 'changed data'

    // 正确
    this.setData({
      text: 'ha',
    });
  },
  changeArray() {
    // 可以直接使用数据路径来修改数据
    this.setData({
      'array[0].text': 'b',
    });
  },
  changePlanetColor(){
    this.setData({
      'object.text': 'red',
    });
  },
  addNewKey() {
    this.setData({
      'newField.text': 'c',
    });
  },
  changeName() {
    this.setData({
      name: 'aliyun',
    }, () => { // 接受传递回调函数
      console.log(this); // this 当前页面实例
      this.setData({ name: this.data.name + ', ' + 'welcome!'});
    });
  },
});
```

参数说明：

事件	类型	描述	最低版本
data	Object	待改变的数据	-
callback	Function	回调函数，在页面渲染更新完成之后执行。	使用 `my.canIUse('page.setData.callback')` 做兼容性处理。详见1.7.0

Page.prototype.$spliceData(data: Object, callback: Function)

说明： `$spliceData` 自 1.7.2 之后才支持，可以使用 **my.canIUse('page.$spliceData')** 做兼容性处理。

`spliceData` 同样用于将数据从逻辑层发送到视图层，但是相比于 `setData`，在处理长列表的时候，其具有更高的性能。

`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` 可以非常灵活，以数据路径的形式给出，如 `array[2].message`、`a.b.c.d`，可以不需要在 `this.data` 中预先定义。`value` 为一个数组（格式：[start, deleteCount, ...items]），数组的第一个元素为操作的起始位置，第二个元素为删除的元素的个数，剩余的元素均为插入的数据。对应 `es5` 中数组的 `splice` 方法。

示例代码：

```
<!-- pages/index/index.axml -->
<view class="spliceData">
  <view a:for="{{a.b}}" key="{{item}}" style="border:1px solid red">
    {{item}}
  </view>
</view>
```

```
// pages/index/index.js
Page({
  data: {
    a: {
      b: [1,2,3,4],
    },
  },
  onLoad(){
    this.$spliceData({ 'a.b': [1, 0, 5, 6] });
  },
});
```

页面输出：

```
1
5
6
2
3
4
```

参数说明：

事件	类型	描述
data	Object	待改变的数据
callback	Function	回调函数，在页面渲染更新完成之后执行。

Page.prototype.$batchedUpdates(callback: Function)

批量更新数据。

说明 `$batchedUpdates` 自 1.14.0 之后才支持，可以使用 **my.canIUse('page.$batchedUpdates')** 做兼容性处理。

参数说明：

事件	类型	描述
callback	Function	在此回调函数中的数据操作会被批量更新。

下面是示例代码：

```
// pages/index/index.js
Page({
  data: {
    counter: 0,
  },
  plus() {
    setTimeout(() => {
      this.$batchedUpdates(() => {
        this.setData({
          counter: this.data.counter + 1,
        });
        this.setData({
          counter: this.data.counter + 1,
        });
      });
    }, 200);
  },
});
```

```
<!-- pages/index/index.axml -->
<view>{{counter}}</view>
<button onTap="plus">+2</button>
```

1. 本示例中每次点击按钮，页面的 `counter` 会加 2
2. 将 `setData` 放在 this.$batchedUpdates 中，这样尽管有多次 `setData` ，但是却只有一次数据的传输

Page.route

`Page` 路径，对应 app.json 中配置的路径值，类型为 `String` 。

注意：这是一个只读属性

```
Page({
  onShow() {
    // 对应 app.json 中配置的路径值
    console.log(this.route)
  }
})
```

2.3.6. getCurrentPages 方法

`getCurrentPages()` 方法用于获取当前页面栈的实例，返回页面数组栈。第一个元素为首页，最后一个元素为当前页面。

框架以栈的形式维护当前的所有页面。路由切换与页面栈的关系如下：

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	当前页面出栈，新页面入栈
页面返回	当前页面出栈
Tab 切换	页面全部出栈，只留下新的 Tab 页面

下面代码可以用于检测当前页面栈是否具有 5 层页面深度。

```
if (getCurrentPages().length === 5) {
      my.redirectTo({
        url: '/pages/logs/logs'
      });
    } else {
      my.navigateTo({
        url: '/pages/index/index'
      });
    }
```

注意： 不要尝试修改页面栈，会导致路由以及页面状态错误。

2.3.7. 页面常见问题

小程序怎么使用 cookie?

小程序中不建议使用 cookie，小程序针对服务端回设的 cookie 不会禁用掉，会设置到小程序进程中，下次小程序进行请求，会自动将已有的 cookie 带入到服务端请求中。前端获取不到 cookie，也不会对 cookie 做任何操作。小程序建议使用 缓存。

setDate 了，小程序页面数据怎么没刷新?

请检查是否有 this 对象值，或者代码执行逻辑是否有问题。

生活号跳转到小程序，小程序如何接收到传递的参数?

需要在小程序 app.js 文件 app() 里的 onLaunch(options) 使用 options.query 获取。

扫描小程序码 A 后再扫描小程序码 B，怎么在 onLaunch 获取不到码 B 携带的参数?

在 onShow 函数中获取。

第一次扫码进入小程序，切入后台。第二次扫码进入后，怎么在 onLaunch 函数中获取不到参数?

建议在 onLaunch 和 onShow 函数中都尝试获取参数。

小程序如何获取跳转链接中附带的参数?

使用小程序生命周期 onLaunch 监听小程序初始化，监听器中获取 query 值。

如何去掉启动日志?

在 app.js 中的 onLaunch、onShow 及页面的 js 文件中 onLoad 等小程序初始和页面初始化方法中去掉日志打印的代码。

2.4. AXML

2.4.1. AXML 介绍

AXML 是小程序框架设计的一套标签语言，用于描述小程序页面的结构。

AXML 语法可分为五个部分：数据绑定、条件渲染、列表渲染、模板、引用。

AXML 代码示例：

```
<!-- pages/index/index.axml -->
<view a:for="{{items}}"> {{item}} </view>
<view a:if="{{view == 'WEBVIEW'}}"> WEBVIEW </view>
<view a:elif="{{view == 'APP'}}"> APP </view>
<view a:else> alipay </view>
<view onTap="add"> {{count}} </view>
```

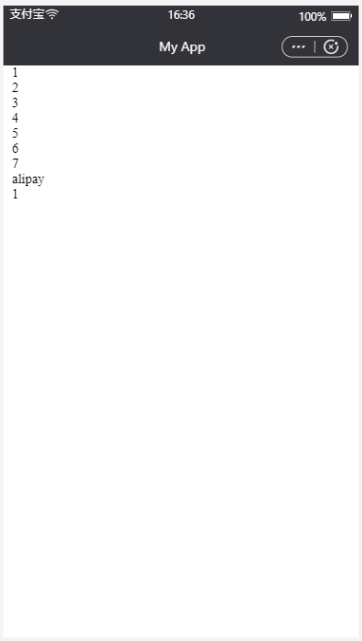
对应的 .js 文件示例：

```
// pages/index/index.js
Page({
  data: {
    items: [1, 2, 3, 4, 5, 6, 7],
    view: 'alipay',
    count: 1,
  },
  add(e) {
    this.setData({
      count: this.data.count + 1,
    });
  },
});
```

对应的 .acss 文件示例：

```
/* pages/index/index.acss */
view {
  padding-left: 10px;
}
```

效果示例：



2.4.2. 数据绑定

AXML 中的动态数据与对应的 `Page` 中 `data` 内容绑定。

简单绑定

数据绑定使用 Mustache 语法将变量用两对大括号（{{}}）封装，可以在多种语法场景下使用。

内容

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

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

组件属性

组件属性需使用双引号（""）封装。

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

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

控制属性

控制属性需使用双引号（""）封装。

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

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

关键字

关键字需使用双引号封装（""）。

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

```
<checkbox checked="{{false}}"> </checkbox>
```

注意: 不要直接写 `checked="false"` ，计算结果是一个字符串，转成布尔值类型后代表真值。

运算

可用两对大括号（{{}}）封装简单的运算。支持如下几种方式：

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,
  },
});
```

页面输出内容为 `3 + 3 + d`

逻辑判断

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

字符串运算

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

```
Page({
  data:{
    name: 'alipay',
  },
});
```

数据路径运算

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

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

组合

可在 Mustache 语法内直接进行组合，构成新的对象或者数组。

数组

```
<view a: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="{{foo: a, bar: b}}"></template>
```

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

最终组合成的对象是 `{foo: 1, bar: 2}` 。

也可用解构运算符 `...` 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 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'}` 。

注意：上面的方式可以随意组合，但是变量名相同时，后边的变量会覆盖前面的变量，比如：

```
<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}` 。

常见问题

Q：跳转页面时，怎么清除 data 数据中的数据？

A：无法清除，可以在跳转时覆盖之前的 data 值。

2.4.3. 条件渲染

a:if

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

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

也可以使用 `a:elif` 和 `a:else` 添加一个 **else** 块。

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

block a:if

因为 `a:if` 是控制属性，需要在标签中使用。如果要一次性判断多个组件标签，可以使用 `<block/>` 标签包装多个组件，并使用 `a:if` 来控制属性。

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

说明： `<block/>` 并不是一个组件，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。

对比 a:if 与 hidden

- `a:if` 中的模板可能包含数据绑定，所以当 `a:if` 的条件值切换时，框架有局部渲染的过程，用于确保条件块在切换时销毁或重新渲染。此外， `a:if` 在初始渲染条件为 false 时，不触发任何渲染动作，当条件第一次变成 true 时才开始局部渲染。
- `hidden` 控制显示与隐藏，组件始终会被渲染。一般来说， `a:if` 有更高的切换消耗而 `hidden` 有更高的初始渲染消耗。因此，在需要频繁切换的情景下，用 `hidden` 更好。如果在运行时条件改变不多则 `a:if` 较好。

2.4.4. 列表渲染

a:for

在组件上使用 `a:for` 属性可以绑定一个数组，即可使用数组中各项的数据重复渲染该组件。

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

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

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

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

`a:for` 支持嵌套。

以下是九九乘法表的嵌套示例代码。

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

block a:for

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

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

a:key

如果列表项位置会动态改变或者有新项目添加到列表中，同时希望列表项保持特征和状态（比如 `<input/>` 中的输入内容，`<switch/>` 的选中状态），需要使用 `a:key` 来指定列表项的唯一标识。

`a:key` 的值以两种形式来提供：

- 字符串：代表列表项某个属性，属性值需要是列表中唯一的字符串或数字，比如 ID，并且不能动态改变。
- 保留关键字 `*this`，代表列表项本身，并且它是唯一的字符串或者数字，比如当数据改变触发重新渲染时，会校正带有 `key` 的组件，框架会确保他们重新被排序，而不是重新创建，这可以使组件保持自身状态，提高列表渲染效率。

注意：

- 如不提供 `a:key`，会报一个 warning。
- 如果明确知道列表是静态，或者不用关注其顺序，则可以忽略。

下面是示例代码：

```
<view class="container">
  <view a:for="{{list}}" a:key="*this">
    <view onTap="bringToFront" data-value="{{item}}">
    {{item}}: click to bring to front
    </view>
  </view>
</view>
```

```
Page({
  data:{
    list:['1', '2', '3', '4'],
  },
  bringToFront(e) {
    const { value } = e.target.dataset;
    const list = this.data.list.concat();
    const index = list.indexOf(value);
    if (index !== -1) {
      list.splice(index, 1);
      list.unshift(value);
      this.setData({ list });
    }
  },
});
```

key

`key` 是比 `a:key` 更通用的写法，里面可以填充任意表达式和字符串。

注意：`key` 不能设置在 block 上。

下面是示例代码：

```
<view class="container">
  <view a:for="{{list}}" key="{{item}}">
    <view onTap="bringToFront" data-value="{{item}}">
    {{item}}: click to bring to front
    </view>
  </view>
</view>
```

```
Page({
  data:{
    list:['1', '2', '3', '4'],
  },
  bringToFront(e) {
    const { value } = e.target.dataset;
    const list = this.data.list.concat();
    const index = list.indexOf(value);
    if (index !== -1) {
      list.splice(index, 1);
      list.unshift(value);
      this.setData({ list });
    }
  },
});
```

2.4.5. 模板

axml 提供模板 `template`，可以在模板中定义代码片段，然后在不同地方调用。

建议使用 `template` 方式引入模版片段，因为 `template` 会指定其作用域，只使用 data 传入的数据，如果 `template` 的 `data` 没有改变，该片段 UI 不会重新渲染。

定义模板

使用 name 属性申明模板名，然后在 `<template/>` 内定义代码片段。

```
<!--
  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，声明需要的模板，然后将需要的 data 传入，比如：

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

```
Page({
  data: {
    item: {
      index: 0,
      msg: 'this is a template',
      time: '2019-04-19',
    },
  },
});
```

`is` 属性可以使用 Mustache 语法，来动态决定具体渲染哪个模板。

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

模板作用域

模板有其作用域，只能使用 data 传入的数据。除了直接由 data 传入数据外，也可以通过 onXX 事件绑定页面逻辑进行函数处理。如下代码所示：

```
<!-- templ.axml -->
<template name="msgItem">
    <view>
        <view>
            <text> {{index}}: {{msg}} </text>
            <text> Time: {{time}} </text>
        </view>
        <button onTap="onClickButton">onTap</button>
    </view>
</template>
```

```
<!-- index.axml -->
<import src="./templ.axml"/>
<template is="msgItem" data="{{...item}}"/>
```

```
Page({
  data: {
    item: {
      index: 0,
      msg: 'this is a template',
      time: '2019-04-22'
    }
  },
  onClickButton(e) {
    console.log('button clicked', e)
  },
});
```

2.4.6. 引用

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

import

import 可以加载已经定义好的 `template` 。

比如，在 item.axml 中定义了一个叫 `item` 的 `template` 。

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

在 index.axml 中引用 item.axml，就可以使用 `item` 模板。

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

import 有作用域的概念，只会 import 目标文件中定义的 template，而不会 import 目标文件 import 的 template。

比如，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 中定义的 template。

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

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

```
<!-- c.axml -->
<import src="./b.axml"/>
<template is="A"/>  <!-- 注意：不能使用 import A -->
<template is="B"/>
```

注意 template 的子节点只能是一个，例如：

允许的示例：

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

不允许的示例：

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

include

include 可以将目标文件除 `<template/>` 外整个代码引入，相当于是拷贝到 include 位置。

代码示例：

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

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

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

引入路径

模板引入路径支持相对路径、绝对路径，也支持从 node_modules 目录载入第三方模块。

```
<import src="./a.axml"/> <!-- 相对路径 -->
<import src="/a.axml"/> <!-- 项目绝对路径 -->
<import src="third-party/x.axml"/> <!--
```

2.5. SJS 语法参考

2.5.1. SJS 介绍

SJS（safe/subset javascript）是小程序一套自定义脚本语言，可以在 AXML 中使用其构建页面结构。

SJS 是 JavaScript 语言的子集，与 JavaScript 是不同的语言，其语法并不与 JavaScript 一致，请勿将其等同于 JavaScript。

使用方式

在 `.sjs` 文件中定义 SJS：

```
// pages/index/index.sjs
const message = 'hello aliyun';
const getMsg = x => x;
export default {
  message,
  getMsg,
};
```

```
// pages/index/index.js
Page({
  data: {
    msg: 'hello alibaba',
  },
});
```

```
<!-- pages/index/index.axml -->
<import-sjs name="m1" from="./index.sjs"/>
<view>{{m1.message}}</view>
<view>{{m1.getMsg(msg)}}</view>
```

页面输出：

```
hello aliyun
hello alibaba
```

注意：

- sjs 只能定义在 `.sjs` 文件中。然后在 axml 中使用 `<import-sjs>` 标签引入。
- sjs 可以调用其他 sjs 文件中定义的函数。
- sjs 是 JavaScript 语言的子集，请勿将其等同于 JavaScript。
- sjs 的运行环境和其他 JavaScript 代码是隔离的， sjs 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。
- sjs 函数不能作为组件事件回调。
- sjs 不依赖于基础库版本，可以在所有版本小程序中运行。

import-sjs 标签

示例代码：

```
// pages/index/index.js
Page({
  data: {
    msg: 'hello aliyun',
  },
});
```

```
// pages/index/index.sjs
function bar(prefix) {
  return prefix;
}
export default {
  foo: 'foo',
  bar: bar,
};
```

```
// pages/index/namedExport.sjs
export const x = 3;
export const y = 4;
```

```
<!-- pages/index/index.axml -->
<import-sjs from="./index.sjs" name="test"></import-sjs>
<!-- 也可以直接使用单标签闭合的写法
<import-sjs from="./index.sjs" name="test" />
-->
<!-- 调用 test 模块里面的 bar 函数，且参数为 test 模块里面的 foo -->
<view> {{test.bar(test.foo)}} </view>
<!-- 调用 test 模块里面的 bar 函数，且参数为 page.js 里面的 msg -->
<view> {{test.bar(msg)}} </view>
<!-- 支持命名导出（named export） -->
<import-sjs from="./namedExport.sjs" name="{x, y: z}" />
<view>{{x}}</view>
<view>{{z}}</view>
```

页面输出：

```
foo
hello aliyun
3
4
```

属性	类型	是否必填	说明
name	String	是	当前 `<import-sjs>` 标签的模块名
from	String	是	引用 .sjs 文件的相对路径

说明：

name 属性指定当前 `<import-sjs>` 标签的模块名。在单个 AXML 文件内，建议将 name 值设为唯一。若有重复模块名则按照先后顺序覆盖（后者覆盖前者）。不同 AXML 文件之间的 `<import-sjs>` 模块名不会相互覆盖。

name 属性可使用一个字符串表示默认模块名，也可使用 `{x}` 表示命名模块的导出。

注意：

- 引用时务必使用 “.sjs” 文件后缀。
- 若定义了一个 .sjs 模块，但从未引用，则该模块不会被解析与运行。

2.5.2. 变量

SJS 中的变量均为值的引用。

语法规则

- var 与 JavaScript 中表现一致，会有变量提升。
- 支持 const 与 let，与 JavaScript 表现一致。
- 没有声明的变量直接赋值使用，会被定义为全局变量。
- 只声明变量而不赋值，默认值为 `undefined` 。

```
var num = 1;
var str = "hello alipay";
var undef; // undef === undefined
const n = 2;
let s = 'string';
globalVar = 3;
```

变量名

命名规则

变量命名必须符合下面两个规则：

- 首字符必须是：字母（a-z,A-Z），下划线（_）
- 首字母以外的字符可以是：字母（a-z,A-Z），下划线（_），数字（0-9）

保留标识符

与 Javascript 语法规则一致，以下标识符不能作为变量名：

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

2.5.3. 注释

注释方法和 Javascript 一致，可以使用以下方法对 SJS 代码进行注释：

```
// page.sjs
// 方法一：这是一个单行注释
/*
方法二：这是一个多行注释
中间的内容都会被注释
*/
let h = 'hello';
const w = ' aliyun';
```

2.5.4. 运算符

算术运算符

```
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);
```

加法 `+` 运算符可用作字符串拼接。

```
var a = 'hello', b = ' alipay';
// 字符串拼接
console.log('hello alipay' === a + b);
```

比较运算符

```
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));
```

二元逻辑运算符

```
var a = 10, b = 20;
// 逻辑与
console.log(20 === (a && b));
// 逻辑或
console.log(10 === (a || b));
// 逻辑否，取反运算
console.log(false === !a);
```

位运算符

```
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));
```

赋值运算符

```
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);
```

一元运算符

```
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);
```

三元运算符

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

逗号运算符

```
var a = 10, b = 20;
// 逗号运算符
console.log(20 === (a, b));
```

运算符优先级

SJS 运算符的优先级与 Javascript 一致。

2.5.5. 语句

if 语句

在 .sjs 文件中，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;
}
```

- `default` 分支可以省略不写。

- `case` 关键词后面只能使用：`变量`，`数字`，`字符串`。

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 (var i = 0; i < 3; ++i) {
  console.log(i);
  if( i >= 1) break;
}
```

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)
```

- 当 `表达式` 为 true 时，循环执行 `语句` 或 `代码块`。
- 支持使用 `break`，`continue` 关键词。

2.5.6. 数据类型

sjs 目前支持如下数据类型：

- **string**: 字符串
- **boolean**: 布尔值
- **number**: 数值
- **object**: 对象
- **function**: 函数
- **array**: 数组
- **date**: 日期
- **regexp**: 正则表达式

判断数据类型

sjs 提供了 constructor 与 typeof 两种方式判断数据类型。

constructor

```
const number = 10;
console.log(number.constructor); // "Number"
const string = "str";
console.log(string.constructor); // "String"
const boolean = true;
console.log(boolean.constructor); // "Boolean"
const object = {};
console.log(object.constructor); // "Object"
const func = function(){};
console.log(func.constructor); // "Function"
const array = [];
console.log(array.constructor); // "Array"
const date = getDate();
console.log(date.constructor); // "Date"
const regexp = getRegExp();
console.log(regexp.constructor); // "RegExp"
```

typeof

```
const num = 100;
const bool = false;
const obj = {};
const func = function(){};
const array = [];
const date = getDate();
const regexp = getRegExp();
console.log(typeof num); // 'number'
console.log(typeof bool); // 'boolean'
console.log(typeof obj); // 'object'
console.log(typeof func); // 'function'
console.log(typeof array); // 'object'
console.log(typeof date); // 'object'
console.log(typeof regexp); // 'object'
console.log(typeof undefined); // 'undefined'
console.log(typeof null); // 'object'
```

string

语法

```
'hello aliyun';
"hello alibaba";
```

es6 语法

```
// 字符串模板
const a = 'hello';
const str = `${a} aliyun`;
```

属性

- `constructor` : 返回值 `"String"`
- `length`

> 说明
> 除 constructor 外属性的具体含义请参考 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
> 具体使用请参考 ES5 标准。

number

语法

```
const num = 10;
const PI = 3.141592653589793;
```

属性

- `constructor` : 返回值 `"Number"`

方法

- toString
- toLocaleString
- valueOf
- toFixed
- toExponential
- toPrecision

> 说明
>
> 具体使用请参考 ES5 标准。

boolean

布尔值只有两个特定的值：true 和 false。

语法

```
const a = true;
```

属性

- `constructor`：返回值 `"Boolean"`

方法

- toString
- valueOf

> 说明
>
> 具体使用请参考 ES5 标准。

object

语法

```
var o = {}; // 生成一个新的空对象
// 生成一个新的非空对象
o = {
  'str': "str",  // 对象的 key 可以是字符串
  constVar: 2,  // 对象的 key 也可以是符合变量定义规则的标识符
  val: {}, // 对象的 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.constVar);
// 对象属性的写操作
o['string']++;
o['string'] += 10;
o.constVar++;
o.constVar += 10;
// 对象属性的读操作
console.log(12 === o['string']);
console.log(13 === o.constVar);
```

es6 语法：

```
// 支持
let a = 2;
o = {
  a, // 对象属性
  b() {}, // 对象方法
};
const { a, b, c: d, e = 'default'} = {a: 1, b: 2, c: 3}; // 对象解构赋值 & default
const {a, ...other} = {a: 1, b: 2, c: 3}; // 对象解构赋值
const f = {...others}; // 对象解构
```

属性

- `constructor`：返回值 `"Object"`

```
console.log("Object" === {a:2,b:"5"}.constructor);
```

方法

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

function

语法

```
// 方法 1：函数声明
function a (x) {
  return x;
}
// 方法 2：函数表达式
var b = function (x) {
  return x;
};
// 方法 3：箭头函数
const double = x => x * 2;
function f(x = 2){} // 函数参数默认
function g({name: n = 'xiaoming', ...other} = {}) {} // 函数参数解构赋值
function h([a, b] = []) {} // 函数参数解构赋值
// 匿名函数、闭包
var c = function (x) {
  return function () { return x;}
};
var d = c(25);
console.log(25 === d());
```

function 中可以使用 `arguments` 关键字。

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

输出：

```
true
true
true
```

属性

- `constructor`：返回值 `"Function"`

- `length`：返回函数的形参个数

方法

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

示例

```
var f = function (a,b) { }
console.log("Function" === f.constructor);
console.log("[function Function]" === f.toString());
console.log(2 === f.length);
```

输出：

```
true
true
true
```

array

语法

```
var a = [];       // 空数组
a = [5,"5",{},function(){}];  // 非空数组，数组元素可以是任何类型
const [b, , c, d = 5] = [1,2,3]; // 数组解构赋值 & 默认值
const [e, ...other] = [1,2,3]; // 数组解构赋值
const f = [...other]; // 数组解构
```

属性

- `constructor`：返回值 `"Array"`
- `length`

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

方法

- toString
- concat
- join
- pop
- push
- reverse
- shift
- slice
- sort
- splice
- unshift
- indexOf
- lastIndexOf
- every
- some

- forEach
- map
- filter
- reduce
- reduceRight

> 说明
> 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`：返回值 `"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

> 说明
>
> 具体使用请参考 ES5 标准。

示例

```
let date = getDate(); //返回当前时间对象
date = getDate(1500000000000);
// Fri Jul 14 2017 10:40:00 GMT+0800 (中国标准时间)
date = getDate('2016-6-29');
// Fri June 29 2016 00:00:00 GMT+0800 (中国标准时间)
date = getDate(2017, 6, 14, 10, 40, 0, 0);
// Fri Jul 14 2017 10:40:00 GMT+0800 (中国标准时间)
```

regexp

语法

生成 regexp 对象需要使用 getRegExp 函数。

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

参数

- pattern: 正则的内容。
- flags：修饰符。只能包含以下字符：
  - `g`: global

- `i`: ignoreCase
- `m`: multiline

属性

- constructor：返回字符串 `"RegExp"` 。
- global
- ignoreCase
- lastIndex
- multiline
- source

> 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
> 具体使用请参考 ES5 标准。

示例

```
var reg = getRegExp("name", "img");
console.log("name" === reg.source);
console.log(true === reg.global);
console.log(true === reg.ignoreCase);
console.log(true === reg.multiline);
```

2.5.7. 基础类库

Global

注意：SJS 不支持 JavaScript 的大部分全局属性和方法。

属性

- Infinity
- NaN
- undefined

具体使用请参考 ES5 标准。

方法

- decodeURI
- decodeURIComponent
- encodeURI
- encodeURIComponent
- isNaN
- isFinite
- parseFloat

- parseInt

具体使用请参考 ES5 标准。

console

`console.log` 方法可在 console 窗口输出信息，可以接受多个参数，将多个参数结果连接起来输出。

Date

方法

- now
- parse
- UTC

具体使用请参考 ES5 标准。

Number

属性

- MAX_VALUE
- MIN_VALUE
- NEGATIVE_INFINITY
- POSITIVE_INFINITY

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

```
console.log(undefined === JSON.stringify());
console.log(undefined === JSON.stringify(undefined));
console.log("null"===JSON.stringify(null));
console.log("222"===JSON.stringify(222));
console.log('"222"'===JSON.stringify("222"));
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(222===JSON.parse(JSON.stringify(222)));
console.log("222"===JSON.parse(JSON.stringify("222")));
console.log(true===JSON.parse(JSON.stringify(true)));
console.log(undefined===JSON.parse(JSON.stringify(function(){})));
```

Math

属性

- E
- LN10
- LN2
- LOG2E
- LOG10E
- PI

- SQRT1_2
- SQRT2

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

具体使用请参考 ES5 标准。

2.5.8. esnext

SJS 支持部分 ES6 语法。

let & const

```
function test(){
  let a = 5;
  if (true) {
    let b = 6;
  }
  console.log(a); // 5
  console.log(b); // 引用错误：b 未定义
}
```

箭头函数

```
const a = [1,2,3];
const double = x => x * 2; // 箭头函数
console.log(a.map(double));
var bob = {
  _name: "Bob",
  _friends: [],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " knows " + f));
  }
};
console.log(bob.printFriends());
```

更简洁的对象字面量（enhanced object literal）

```
var handler = 1;
var obj = {
  handler, // 对象属性
  toString() { // 对象方法
    return "string";
  },
};
```

注意： 不支持 `super` 关键字，不能在对象方法中使用 `super`

模板字符串（template string）

```
const h = 'hello';
const msg = `${h} alipay`;
```

解构赋值（Destructuring）

```
// array 解构赋值
var [a, ,b] = [1,2,3];
a === 1;
b === 3;
// 对象解构赋值
var { op: a, lhs: { op: b }, rhs: c }
       = getASTNode();
// 对象解构赋值简写
var {op, lhs, rhs} = getASTNode();
// 函数参数解构赋值
function g({name: x}) {
  console.log(x);
}
g({name: 5});
// 解构赋值默认值
var [a = 1] = [];
a === 1;
// 函数参数：解构赋值 + 默认值
function r({x, y, w = 10, h = 10}) {
  return x + y + w + h;
}
r({x:1, y:2}) === 23;
```

Default + Rest + Spread

```
// 函数参数默认值
function f(x, y=12) {
  // 如果不给y传值，或者传值为undefied，则y的值为12
  return x + y;
}
f(3) == 15;
function f(x, ...y) {
  // y 是一个数组
  return x * y.length;
}
f(3, "hello", true) == 6;
function f(x, y, z) {
  return x + y + z;
}
f(...[1,2,3]) == 6; // 数组解构
const [a, ...b] = [1,2,3]; // 数组解构赋值，b = [2, 3]
const {c, ...other} = {c: 1, d: 2, e: 3}; // 对象解构赋值，other = {d: 2, e: 3}
const d = {...other}; // 对象解构
```

2.6. ACSS 语法参考

ACSS 是一套样式语言，用于描述 AXML 的组件样式，决定 AXML 的组件的显示效果。

为适应广大前端开发者，ACSS 和 CSS 规则完全一致，100% 可以用。同时为更适合开发小程序，对 CSS 进行了扩充，ACSS 支持 px，rpx，vh，vw 等单位。

ACSS 已经帮开发者做了不同手机端的样式兼容性处理。

rpx

rpx（responsive pixel）可以根据屏幕宽度进行自适应，规定屏幕宽为 750rpx。以 Apple iPhone6 为例，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。

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

样式导入

使用 `@import` 语句可以导入外联样式表，`@import` 后需要加上外联样式表相对路径，用 `;` 表示结束。

示例代码：

```
/** button.acss **/
.sm-button {
  padding: 5px;
}
```

```
/** app.acss **/
@import "./button.acss";
.md-button {
  padding: 15px;
}
```

导入路径支持从 node_modules 目录载入第三方模块，例如 page.acss:

```
@import "./button.acss"; /*相对路径*/
@import "/button.acss"; /*项目绝对路径*/
@import "third-party/page.acss"; /*第三方 npm 包路径*/
```

内联样式

组件上支持使用 `style`、`class` 属性来控制样式。

style 属性

用于接收动态样式，样式在运行时会进行解析。行内样式不支持 `!important` 优先级规则。

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

class 属性

用于接收静态样式，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 `.`，多个类名间以空格分隔。请静态样式写进 class 中，避免将静态样式写进 style 中，以免影响渲染速度。

```
<view class="my-awesome-view" />
```

选择器

同 CSS3 保持一致。

注意：

- `.a-`，`.am-` 开头的类选择器为系统组件占用，不可使用。
- 不支持属性选择器。

全局样式与局部样式

- app.acss 中的样式为全局样式，作用于每一个页面。
- 页面文件夹内的 .acss 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.acss 中相同的选择器。

本地资源引用

ACSS 文件里的本地资源引用请使用绝对路径的方式，不支持相对路径引用。例如：

```
/* 支持 */
background-image: url('/images/ant.png');
/* 不支持 */
background-image: url('./images/ant.png');
```

常见问题

Q：一个 axml 引用多个自定义组件或 template 模板、include 等，造成样式之间相互影响、样式污染怎么办？

A：请开发者自行通过名空间处理样式隔离。

Q：给页面设高度100%为什么没用？

A：添加一个绝对定位就可以了，不添加的话，会根据您的页面的内容去自适应的。

相关文档

app.json 全局配置

3.组件

3.1. 基础组件

3.1.1. 概览

扫码体验

视图容器

名称	功能说明
view	视图容器
swiper	滑块视图容器
scroll-view	可滚动视图区域
cover-view	覆盖在原生组件之上的文本视图
cover-image	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
movable-view	可移动的视图容器
movable-area	movable-view 的可移动区域

基础内容

名称	功能说明
text	文本
icon	图标
progress	进度条
rich-text	富文本组件

表单组件

名称	功能说明
button	按钮
form	表单
label	用来改进表单组件的可用性
input	输入框
textarea	多行输入框
radio	单选按钮
checkbox	多项选择器
switch	单选开关
slider	滑动选择器
picker-view	嵌入页面的滚动选择器
picker	从底部弹起的滚动选择器

导航

名称	功能说明
navigator	页面链接

媒体组件

名称	功能说明
image	图片
video	视频

画布

名称	功能说明
canvas	画布

3.1.2. 使用说明

小程序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业务开发。

共有属性

所有的组件都包含以下属性：

属性名	类型	描述	注解
id	String	组件的唯一标识	
class	String	样式类	
style	String	内联样式	
data-*	Any	自定义属性	当事件触发时，会将自定义属性传递给事件处理函数
on* / catch*	EventHandle	事件绑定，遵循驼峰命名规范，例如 `onTap`	参见事件

属性类型

每个组件都有一系列的属性配置，每个属性值都有类型要求。

类型	描述	注释
Boolean	布尔值	
Number	数字	
String	字符串	
Array	数组	
Object	对象	
EventHandle	事件处理函数	需在 Page 中定义事件处理函数名对应的实现
any	任意类型	

数据绑定

通过 Mustache 语法两对大括号（{{}}）绑定动态数据，参见 数据绑定。

3.1.3. 视图组件

3.1.3.1. view 视图容器

视图容器。相当于 web 的 div 或者 react-native 的 view。如果需要使用滚动视图，请使用 scroll-view 可滚动视图区域。

扫码体验

示例代码

```
<!-- API-DEMO page/component/view.axml -->
<view class="page">
  <view>
    <button a:if="{{returnIndex}}" onTap="returnIndex">回到首页</button>
  </view>
  <view class="page-description">视图容器，相当于 web 的 div 或者 react-native 的 View。</view>
  <view class="page-section">
    <view class="page-section-demo">
      <view class="stream">
        <view class="post">
          <view class="postUser">
            <view class="postUser__name">Chris</view>
          </view>
          <view class="postBody">
            <view class="postBody__content">
              欢迎使用支付宝小程序！！！
            </view>
            <view class="postBody__date">
              May 20
            </view>
          </view>
        </view>
        <view class="post">
          <view class="postUser">
            <view class="postUser__name">Jack</view>
          </view>
          <view class="postBody">
            <view class="postBody__content">
              @Chris 我该如何上手?
            </view>
            <view class="postBody__date">
              May 21
            </view>
          </view>
        </view>
        <view class="post">
          <view class="postUser">
            <view class="postUser__name">Chris</view>
          </view>
          <view class="postBody">
            <view class="postBody__content">
              你可以查看 Demo，对小程序有一个简单的了解；然后下载我们的 IDE 进行开发。
            </view>
            <view class="postBody__date">
              May 22
            </view>
          </view>
        </view>
        <view class="post">
          <view class="postUser">
            <view class="postUser  name">Jessie</view>
```

```
          </view>
          <!-- hover red -->
          <view class="postBody" hover-class="red">
            <view class="postBody__content">
              赞!
            </view>
            <view class="postBody__date" hidden>
              June 1
            </view>
          </view>
        </view>
        <view class="post" hidden>
          <view class="postUser">
            <view class="postUser__name">Jessie</view>
          </view>
          <view class="postBody">
            <view class="postBody__content">
              赞! +1
            </view>
            <view class="postBody__date">
              June 1
            </view>
          </view>
        </view>
      </view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/view.js
Page({
  data: {
    pageName: 'component/view',
  },
  onLoad() {
    this.setData({
      returnIndex: getCurrentPages().length === 1,
    })
  },
  returnIndex() {
    my.switchTab({ url: '/page/component/index' });
  },
});
```

```
/* API-DEMO page/component/view.acss */
.post + .post {
  margin-top: 10rpx;
}
.post {
  display: flex;
}
.postUser {
  flex: 0 1 auto;
  padding-bottom: 20rpx;
}
.postUser__name {
  width: 180rpx;
  color: #57727C;
  font-size: 24rpx;
  font-weight: 700;
  line-height: 1;
  text-align: center;
  margin-top: 60rpx;
}
.postBody {
  flex: 1 1 0%;
  position: relative;
  padding: 30rpx;
  border: 2rpx solid #CAD0D2;
  border-radius: 8rpx;
}
.postBody:after,
.postBody:before {
  right: 100%;
  top: 70rpx;
  border: solid transparent;
  content: " ";
  height: 0;
  width: 0;
  position: absolute;
  pointer-events: none;
}
.postBody:after {
  border-color: transparent;
  border-right-color: #ffffff;
  border-width: 16rpx;
  margin-top: -16rpx;
}
.postBody:before {
  border-color: transparent;
  border-right-color: #CAD0D2;
  border-width: 18rpx;
  margin-top: -18rpx;
}
.postBody__content {
  color: #57727C;
  font-size: 24rpx;
}
.postBody__date {
  margin-top: 10rpx;
  color: #86969C;
  font-size: 20rpx;
  text-transform: uppercase;
  letter-spacing: 2rpx;
}
```

属性

属性名	类型	默认值	描述

disable-scroll	Boolean	false	是否阻止区域内滚动页面 **说明：如果 view 中嵌套 view，外层 view 设置 disable-scroll 为 true 时将阻止内部的滚动属性。**
hover-class	String	-	触摸时添加的样式类
hover-start-time	Number	-	按住多久后出现点击状态，单位毫秒
hover-stay-time	Number	-	松开后点击状态保留时间，单位毫秒
hidden	boolean	false	是否隐藏
class	String	-	自定义样式名
style	String	-	内联样式
animation	Object	{}	用于动画，详见 my.createAnimation。使用 `my.createAnimation` 生成的动画是通过过渡（Transition）实现的，只会触发 `onTransitionEnd`，不会触发 `onAnimationStart`，`onAnimationIteration`，`onAnimationEnd`。
hover-stop-propagation	Boolean	false	是否阻止当前元素的祖先元素出现点击态
onTap	EventHandle	-	点击
onTouchStart	EventHandle	-	触摸动作开始
onTouchMove	EventHandle	-	触摸后移动
onTouchEnd	EventHandle	-	触摸动作结束

onTouchCancel	EventHandle	-	触摸动作被打断，如来电提醒，弹窗
onLongTap	EventHandle	-	长按 500ms 之后触发，触发了长按事件后进行移动将不会触发屏幕的滚动
onTransitionEnd	EventHandle	-	过渡（Transition）结束时触发
onAnimationIteration	EventHandle	-	每开启一次新的动画过程时触发。（第一次不触发）
onAnimationStart	EventHandle	-	动画开始时触发
onAnimationEnd	EventHandle	-	动画结束时触发
onAppear	EventHandle	-	当前元素可见时触发。
onDisappear	EventHandle	-	当view标签隐藏到2/3时会触发。
onFirstAppear	EventHandle	-	当前元素首次可见时触发。
role	-	-	表示组件的语义角色。设置为 img 时，组件聚焦后读屏软件会朗读出**图像**；设置为 button 时，聚焦后读屏软件会朗读出**按钮**。

常见问题

如何改变 view 的展示顺序？

将这两个模块嵌入到一个循环里面，每一个循环的小模块加一个类型值进行标识。

页面滚动时在不同屏幕滚动到的位置不同，如何解决？

1.按照屏幕的比例计算，换算跳转位置 。

2.固定一个滑动位置，在该位置写入一个 view，通过计算这个 view 到顶部距离，进行跳转。

3.1.3.2. swiper 滑块视图容器

滑块容器

扫码体验

子组件

swiper-item

仅可放置在 swiper 中，宽高自动撑满。

示例代码

```
<template>
  <swiper
    :indicator-dots="showDots"
    :autoplay="autoplay"
    :interval="interval"
    class="swiper-container"
    indicator-color="rgba(255,255,255,0.6)"
    indicator-active-color="#00653E"
    :circular="circular"
  >
    <swiper-item>
      <view class="swiper-item">
        <image class='swipe-pic' src="https://gw.alicdn.com/bao/uploaded/TB12bMPGHSYBuNjSspiXXXNzpXa-60-36.png" />
      </view>
    </swiper-item>
    <swiper-item>
      <view class="swiper-item">
        <image class='swipe-pic' src="https://gw.alicdn.com/bao/uploaded/TB12bMPGHSYBuNjSspiXXXNzpXa-60-36.png" />
      </view>
    </swiper-item>
    <swiper-item>
      <view class="swiper-item">
        <image class='swipe-pic' src="https://gw.alicdn.com/bao/uploaded/TB12bMPGHSYBuNjSspiXXXNzpXa-60-36.png" />
      </view>
    </swiper-item>
  </swiper>
</template>
<script>
  export default {
    data() {
      return {
        showDots: true,
        autoplay: true,
        interval: 3000,
        circular: false,
      }
    }
  }
</script>
```

属性

属性名	类型	默认值	说明
vertical	Boolean	false	滑动方向是否为纵向
duration	Number	500	滑动动画时长，单位毫秒
circular	Boolean	false	是否开启循环滑动
autoplay	Boolean	false	是否自动切换
current	Number	0	当前页面的索引
interval	Number	5000	自动切换时间间隔，单位毫秒
indicator-dots	Boolean	false	是否显示指示点
indicator-color	Color	rgba(0, 0, 0, .3)	指示点颜色
indicator-active-color	Color	#000	当前选中的指示点颜色
change	Function		current 改变时会触发，`event.detail = {current: current}`

3.1.3.3. scroll-view 可滚动视图区域

scroll-view 可滚动视图组件，用于包裹可滚动元素，支持横向滚动和纵向滚动。

扫码体验：

示例代码

```
<style>
  .red {
    background-color: #f56264;
  }
  .green {
    background-color: #1fb822;
  }
  .blue {
    background-color: #24b0fc;
  }
  .yellow {
    background-color: #F0E68C;
  }
  .scroll-view-item {
    width: 100%;
    height: 100px;
    color: #fff;
  }
  .scroll-view-item_H {
    display: inline-block;
    width: 200px;
    height: 200px;
    color: #fff;
  }
  .scroll-view_H {
    white-space: nowrap;
  }
</style>
<template>
  <view class="container">
    <view class="page-section">
      <view class="page-section-title">vertical scroll</view>
      <view class="page-section-demo">
        <scroll-view
          style="height: 200px;"
          scroll-with-animation

          :scroll-y="true"
          :scroll-into-view="toView"
          :scroll-top="scrollTop"
          @scrolltoupper="upper"
          @scrolltolower="lower"
          @scroll="scroll"
        >
          <view id="blue" class="scroll-view-item blue"></view>
          <view id="red" class="scroll-view-item red"></view>
          <view id="yellow" class="scroll-view-item yellow"></view>
```

```
          <view id="yellow" class="scroll-view-item yellow"></view>
          <view id="green" class="scroll-view-item green"></view>
        </scroll-view>
      </view>
      <view class="page-section-btns">
        <button @click="tap">next</button>
        <button @click="tapMove">move</button>
        <button @click="tapMoveTop">scrollToTop</button>
      </view>
    </view>
    <view class="page-section">
      <view class="page-section-title">horizontal scroll</view>
      <view class="page-section-demo">
        <scroll-view class="scroll-view_H" :scroll-x="true" style="width: 100%">
          <view id="blue2" class="scroll-view-item_H blue"></view>
          <view id="red2" class="scroll-view-item_H red"></view>
          <view id="yellow2" class="scroll-view-item_H yellow"></view>
          <view id="green2" class="scroll-view-item_H green"></view>
        </scroll-view>
      </view>
    </view>
  </view>
</template>
<script>
  export default {
    data() {
      return {
        toView: 'red',
        scrollTop: 100,
      };
    },
    methods: {
      scroll(evt) {
        console.log(evt)
      },
      upper(evt) {
        console.log(evt)
      },
      lower(evt) {
        console.log(evt)
      },
      tap(evt) {
        var order = ['red', 'yellow', 'blue', 'green', 'red'];
        for (var i = 0; i < order.length; ++i) {
          if (order[i] === this.toView) {
            this.toView = order[i + 1];
            break;
          }
        }
      },
      tapMove(evt) {
        this.scrollTop = this.scrollTop + 80;
        if (this.scrollTop > 200) {
          this.scrollTop = 0;
        }
      },
      tapMoveTop() {
        this.scrollTop = 0;
      }
    }
  };
</script>
```

属性

属性名	类型	默认值	说明
class	String	''	外部样式名
style	String	''	内联样式名
scroll-y	Boolean	false	允许纵向滚动
scroll-x	Boolean	false	允许横向滚动
upper-threshold	Number	50	距顶部/左边多远时（单位px），触发 scrolltoupper 事件
lower-threshold	Number	50	距底部/右边多远时（单位px），触发 scrolltolower 事件
lower-emit-once	Boolean	false	每次滚动到范围内，只触发一次 scrolltolower 事件
upper-emit-once	Boolean	false	每次滚动到范围内，只触发一次 scrolltoupper 事件
scroll-top	Number	0	设置竖向滚动条位置
scroll-left	Number	0	设置横向滚动条位置
scroll-into-view	String	''	值应为某子元素id，则滚动到该元素，元素顶部对齐滚动区域顶部
scroll-with-animation	Boolean	false	在设置滚动条位置时使用动画过渡
scrolltoupper	EventHandler	noop	滚动到顶部/左边，会触发 scrolltoupper 事件
scrolltolower	EventHandler	noop	滚动到底部/右边，会触发 scrolltolower 事件
scroll	EventHandler	noop	滚动时触发，`event.detail = { scrollLeft, scrollTop, scrollHeight, scrollWidth }`

3.1.3.4. cover-view 文本视图

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、canvas。

注意：实际效果请以真机为准。

扫码体验

示例代码

```
<!-- API-DEMO page/component/cover-view.axml -->
<view class="page">
  <view class="page-description">cover-view</view>
  <view class="page-section">
    <view class="page-section-demo" style="position: relative;">
      <map
        longitude="{{longitude}}"
        latitude="{{latitude}}"
        scale="{{scale}}"
        style="width: 100%; height: 200px;"
        include-points="{{includePoints}}"
      />
      <cover-view class="cover-view">
        <cover-view class="cover-view-item cover-view-item-1"></cover-view>
        <cover-view class="cover-view-item cover-view-item-2"></cover-view>
        <cover-view class="cover-view-item cover-view-item-3"></cover-view>
      </cover-view>
      <cover-image style="" src="/image/ant.png" />
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/cover-view.js
Page({
  data: {
    scale: 14,
    longitude: 120.10675,
    latitude: 30.266786,
    includePoints: [{
      latitude: 30.266786,
      longitude: 120.10675,
    }],
  }
});
```

```
/* API-DEMO page/component/cover-view.acss */
cover-image {
  position: absolute;
  left: 20px;
  top: 100px;
  height: 50px;
  width: 50px;
}
.cover-view {
  position: absolute;
  top: calc(50% - 75rpx);
  left: calc(50% - 150rpx);
  display:flex;
  flex-direction:row;
  background-color: rgba(0, 0, 0, 0);
}
.cover-view-item{
  width: 100rpx;
  height: 150rpx;
  font-size: 26rpx;
}
.cover-view-item-1 {
  background-color: rgba(26, 173, 25, 0.7);
}
.cover-view-item-2 {
  background-color: rgba(39, 130, 215, 0.7);
}
.cover-view-item-3 {
  background-color: rgba(255, 255, 255, 0.7);
}
```

属性

属性名	类型	默认值	描述
onTap	EventHandle	-	点击事件回调

常见问题

cover-view 默认背景白色如何取消？

不支持更改背景色，建议更改字体颜色。

cover-view是否支持圆角和阴影？

小程序 acss 支持圆角和阴影，示例代码：

圆角：border-radius: 15%;

阴影：box-shadow: 10px 10px 5px #888888;

3.1.3.5. cover-image 图片视图

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 文本视图 一致。

扫码体验

示例代码

```
<!-- API-DEMO page/component/cover-view.axml -->
<view class="page">
  <view class="page-description">cover-view</view>
  <view class="page-section">
    <view class="page-section-demo" style="position: relative;">
      <map
        longitude="{{longitude}}"
        latitude="{{latitude}}"
        scale="{{scale}}"
        style="width: 100%; height: 200px;"
        include-points="{{includePoints}}"
      />
      <cover-view class="cover-view">
        <cover-view class="cover-view-item cover-view-item-1"></cover-view>
        <cover-view class="cover-view-item cover-view-item-2"></cover-view>
        <cover-view class="cover-view-item cover-view-item-3"></cover-view>
      </cover-view>
      <cover-image style="" src="/image/ant.png" />
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/cover-view.js
Page({
  data: {
    scale: 14,
    longitude: 120.10675,
    latitude: 30.266786,
    includePoints: [{
      latitude: 30.266786,
      longitude: 120.10675,
    }],
  }
});
```

```
/* API-DEMO page/component/cover-view.acss */
cover-image {
  position: absolute;
  left: 20px;
  top: 100px;
  height: 50px;
  width: 50px;
}
.cover-view {
  position: absolute;
  top: calc(50% - 75rpx);
  left: calc(50% - 150rpx);
  display:flex;
  flex-direction:row;
  background-color: rgba(0, 0, 0, 0);
}
.cover-view-item{
  width: 100rpx;
  height: 150rpx;
  font-size: 26rpx;
}
.cover-view-item-1 {
  background-color: rgba(26, 173, 25, 0.7);
}
.cover-view-item-2 {
  background-color: rgba(39, 130, 215, 0.7);
}
.cover-view-item-3 {
  background-color: rgba(255, 255, 255, 0.7);
}
```

属性

属性名	类型	默认值	描述
src	String	-	图片地址，支持的地址格式同 image 一致
onTap	EventHandle	-	点击事件回调

3.1.3.6. movable-view 可移动视图容器

可移动的视图容器，在页面中可以拖拽滑动。movable-view 必须在 movable-area 可移动视图区域 组件中，并且必须是直接子节点，否则不能移动。

注意：

- 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(请不要修改)，top 和 left 属性为 0px。

- 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）。

扫码体验

示例代码

```
<!-- API-DEMO page/component/movable-view.axml -->
<view class="page">
  <view class="page-description">可移动视图</view>
  <view class="page-section">
    <view class="page-section-title">movable-view区域小于movable-area</view>
    <view class="page-section-demo">
      <movable-area>
        <movable-view x="{{x}}" y="{{y}}" direction="all">movable-view</movable-view>
      </movable-area>
    </view>
    <button style="margin-left: 10px; mrigin-right: 10px;" type="primary" onTap="onButtonTap">点击移动到 (30px, 30px)
</button>
  </view>
  <view class="page-section">
    <view class="page-section-title">movable-view区域大于movable-area</view>
    <view class="page-section-demo">
      <movable-area>
        <movable-view class="max" direction="all">movable-view</movable-view>
      </movable-area>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">只可以横向移动</view>
    <view class="page-section-demo">
     <movable-area>
        <movable-view direction="horizontal">
          movable-view
        </movable-view>
      </movable-area>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">只可以纵向移动</view>
    <view class="page-section-demo">
     <movable-area>
        <movable-view direction="vertical">
          movable-view
        </movable-view>
      </movable-area>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/movable-view.js
Page({
  data: {
    x: 0,
    y: 0,
  },
  onButtonTap() {
    const { x, y } = this.data;
    if (x === 30) {
      this.setData({
        x: x + 1,
        y: y + 1,
      });
    } else {
      this.setData({
        x: 30,
        y: 30
      });
    }
  },
});
```

```
// API-DEMO page/component/movable-view.json
{
  "allowsBounceVertical": "NO"
}
```

```
/* API-DEMO page/component/movable-view.acss */
movable-area {
  height: 400rpx;
  width: 400rpx;
  margin: 50rpx 0rpx 0 50rpx;
  background-color: #ccc;
  overflow: hidden;
}
movable-view {
  display: flex;
  align-items: center;
  justify-content: center;
  height: 200rpx;
  width: 200rpx;
  background: #108ee9;
  color: #fff;
}
.max {
  width: 600rpx;
  height: 600rpx;
}
```

属性

属性名	类型	默认值	描述
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	0	定义 x 轴方向的偏移，会换算为 left 属性，如果 x 的值不在可移动范围内，会自动移动到可移动范围
y	Number	0	定义 y 轴方向的偏移，会换算为 top 属性，如果 y 的值不在可移动范围内，会自动移动到可移动范围
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	false	是否使用动画
onTouchStart	EventHandle	-	触摸动作开始，事件会向父节点传递。
catchTouchStart	EventHandle	-	触摸动作开始，事件仅作用于组件，不向父节点传递。
onTouchMove	EventHandle	-	触摸移动事件，事件会向父节点传递。
catchTouchMove	EventHandle	-	触摸移动事件，事件仅作用于组件，不向父节点传递。
onTouchEnd	EventHandle	-	触摸动作结束，事件会向父节点传递。

catchTouchEnd	EventHandle	-	触摸动作结束，事件仅作用于组件，不向父节点传递。
onTouchCancel	EventHandle	-	触摸动作被打断，如来电提醒，弹窗
onChange	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）
onChangeEnd	EventHandle	-	拖动结束触发的事件，`event.detail = {x: x, y: y}`
onScale	EventHandle	-	缩放过程中触发的事件，event.detail = {x, y, scale}

onChange返回值detail.source

source字段表示产生移动的原因

值	说明
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
空字符串	setData

说明：冒泡事件，请查看 事件介绍 中的 **事件类型**。

3.1.3.7. movable-area 可移动视图区域

movable-view 可移动视图容器 的可移动区域。movable-area 必须设置 width 和 height 属性，不设置默认为 10px。

扫码体验

示例代码

```
<!-- API-DEMO page/component/movable-view.axml -->
<view class="page">
  <view class="page-description">可移动视图</view>
  <view class="page-section">
    <view class="page-section-title">movable-view区域小于movable-area</view>
    <view class="page-section-demo">
      <movable-area>
        <movable-view x="{{x}}" y="{{y}}" direction="all">movable-view</movable-view>
      </movable-area>
    </view>
    <button style="margin-left: 10px; mrigin-right: 10px;" type="primary" onTap="onButtonTap">点击移动到 (30px, 30px)
</button>
  </view>
  <view class="page-section">
    <view class="page-section-title">movable-view区域大于movable-area</view>
    <view class="page-section-demo">
      <movable-area>
        <movable-view class="max" direction="all">movable-view</movable-view>
      </movable-area>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">只可以横向移动</view>
    <view class="page-section-demo">
     <movable-area>
        <movable-view direction="horizontal">
          movable-view
        </movable-view>
      </movable-area>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">只可以纵向移动</view>
    <view class="page-section-demo">
     <movable-area>
        <movable-view direction="vertical">
          movable-view
        </movable-view>
      </movable-area>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/movable-view.js
Page({
  data: {
    x: 0,
    y: 0,
  },
  onButtonTap() {
    const { x, y } = this.data;
    if (x === 30) {
      this.setData({
        x: x + 1,
        y: y + 1,
      });
    } else {
      this.setData({
        x: 30,
        y: 30
      });
    }
  },
});
```

```
// API-DEMO page/component/movable-view.json
{
  "allowsBounceVertical": "NO"
}
```

```
/* API-DEMO page/component/movable-view.acss */
movable-area {
  height: 400rpx;
  width: 400rpx;
  margin: 50rpx 0rpx 0 50rpx;
  background-color: #ccc;
  overflow: hidden;
}
movable-view {
  display: flex;
  align-items: center;
  justify-content: center;
  height: 200rpx;
  width: 200rpx;
  background: #108ee9;
  color: #fff;
}
.max {
  width: 600rpx;
  height: 600rpx;
}
```

属性

属性名	类型	默认值	必填	描述
scale-area	Boolean	false	否	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

3.1.4. 基础内容

3.1.4.1. text 文本

扫码体验

示例代码

```
<!-- API-DEMO page/component/text.axml -->
<view class="page">
  <view class="page-description">
    <view class="text-demo-title">
      <text class="text-demo-text">这是一段文本。\n<text>\</text><text>n</text> 可以换行。</text>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-demo">
      <text>{{text}}</text>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/text.js
Page({
  data: {
    text: `支付宝是一个大型生活服务类的平台，用户群非常广泛，上至五六十岁，下至十几岁。
      这里不仅有官方自营应用，还有第三方接入应用，用户的选择很多。
      只有你的产品做得足够简单，才能让更多的用户使用。而更多人的使用，也意味着你更大的收益。\n\n:)
    `,
  },
});
```

```
/* API-DEMO page/component/text.acss */
.page {
  padding: 0;
}
.text-demo-title {
  margin-left: 30rpx;
  margin-top: 30rpx;
}
.text-demo-text {
  font-size: 36rpx;
}
```

属性

属性名	类型	默认值	描述
selectable	Boolean	false	是否可选
space	String	-	以何种方式显示连续空格

decode	Boolean	false	是否解码。为 true 时表示对文本内容进行解码，可解析的 HTML 实体字符有：` &lt;&gt;&amp;&apos;  `
number-of-lines	number	-	多行省略，值须大于等于1，表现同 css 的 `-webkit-line-clamp` 属性一致

space 有效值

值	说明
nbsp	根据字体设置的空格大小
ensp	中文字符空格一半大小
emsp	中文字符空格大小

说明： 各个操作系统的空格标准并不一致。

3.1.4.2. icon 图标

扫码体验

示例代码

```
<!-- API-DEMO page/component/icon.axml -->
<view class="page">
  <view class="page-description">图标</view>
  <view class="page-section">
    <view class="page-section-title">Type</view>
    <view class="page-section-demo icon-list">
      <block a:for="{{iconType}}">
        <view class="item">
          <icon type="{{item}}" size="45"/>
          <text>{{item}}</text>
        </view>
      </block>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">Size</view>
    <view class="page-section-demo icon-list">
      <block a:for="{{iconSize}}">
        <view class="item">
          <icon type="success" size="{{item}}"/>
          <text>{{item}}</text>
        </view>
      </block>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">Color</view>
    <view class="page-section-demo icon-list">
      <block a:for="{{iconColor}}">
        <view class="item">
          <icon type="success" size="45" color="{{item}}"/>
          <text style="color:{{item}}">{{item}}</text>
        </view>
      </block>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/icon.js
Page({
  data: {
    iconSize: [20, 30, 40, 50, 60],
    iconColor: [
      'red', 'yellow', 'blue', 'green',
    ],
    iconType: [
      'success',
      'info',
      'warn',
      'waiting',
      'clear',
      'success_no_circle',
      'download',
      'cancel',
      'search',
    ],
  },
});
```

```
/* API-DEMO page/component/icon.acss */
.icon-list {
  display: -webkit-flex;
  display: flex;
  -webkit-flex-wrap: wrap;
  flex-wrap: wrap;
}
.item {
  display: -webkit-flex;
  display: flex;
  flex-direction: column;
  -webkit-flex-direction: column;
  margin-bottom: 10px;
  margin-right: 10px;
  align-items: center;
  -webkit-align-items: center;
}
```

属性

属性名	类型	默认值	描述
type	String	-	icon 类型，有效值： info, warn, waiting, cancel, download, search, clear, success, success_no_circle,loading
size	Number	23	icon 大小，单位 px
color	String	-	icon 颜色，同 CSS 色值

3.1.4.3. progress 进度条

扫码体验

示例代码

```
<!-- API-DEMO page/component/progress.axml -->
<view class="page">
  <view class="page-description">进度条</view>
  <view class="page-section">
    <view class="page-section-demo">
      <progress percent="20" show-info/>
      <progress percent="40" active/>
      <progress percent="60" stroke-width="10"/>
      <progress percent="80" color="#10AEFF"/>
      <progress percent="80" activeColor="#6abf47" backgroundColor="#f4333c" />
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/progress.js
Page({});
```

```
/* API-DEMO page/component/progress.acss */
progress{
  margin-bottom: 60rpx;
}
```

属性

属性名	类型	默认值	描述
percent	Float	-	百分比(0~100)
show-info	Boolean	false	在右侧显示百分比值
stroke-width	Number	6	线的粗细，单位 px
active-color	Color	`#09BB07`	已选择的进度条颜色
background-color	Color	-	未选择的进度条颜色
active	Boolean	false	是否添加从0%开始加载的入场动画

3.1.4.4. rich-text 富文本

富文本。

注意：

- 富文本里面写 js 不支持事件执行。
- rich-text 支持 a 标签，不支持超链接。

扫码体验

示例代码

```
<!-- API-DEMO page/component/rich-text.axml -->
<view>
  <rich-text nodes="{{nodes}}" onTap="tap"></rich-text>
</view>
```

```
// API-DEMO page/component/rich-text.js
Page({
  data: {
    nodes: [{
      name: 'div',
      attrs: {
        class: 'wrapper',
        style: 'color: orange;',
      },
      children: [{
        type: 'text',
        text: 'Hello World!',
      }],
    }],
  },
  tap() {
    console.log('tap');
  },
});
```

```
/* API-DEMO page/component/rich-text.acss */
.wrapper {
  padding: 20rpx;
}
```

属性

属性名	类型	默认值	描述
nodes	Array	[]	节点列表

注意：

- nodes 属性只支持使用 Array 类型。如果需要支持 HTML String，则需要自己将 HTML String 转化为 nodes 数组，可使用 mini-html-parser 转换。

支持如下默认事件：

- tap
- touchstart
- touchmove
- touchcancel
- touchend
- longtap

nodes 属性

现支持两种节点：元素节点和文本节点，通过 type 来区分。默认是元素节点，在富文本区域里显示的 HTML 节点。

元素节点

属性	类型	说明	必填	备注
type	String	节点类型	否	默认值为 node
name	String	标签名	是	支持部分受信任的 HTML 节点
attrs	Object	属性	否	支持部分受信任的属性，遵循 Pascal 命名法
children	Array	子节点列表	否	结构和 nodes 相同

受信任的 HTML 节点及属性

支持 class 和 style 属性，不支持 id 属性。

节点	额外支持的属性
a	
abbr	
b	
blockquote	
br	
code	
col	span, width
colgroup	span, width
dd	
del	
div	
dl	
dt	
em	
fieldset	
h1	
h2	
h3	
h4	
h5	
h6	
hr	
i	
img	alt, src, height, width
ins	
label	
legend	
li	
ol	start, type
p	
q	
span	
strong	

sub	
sup	
table	width
tbody	
td	colspan, height, rowspan, width
tfoot	
th	colspan, height, rowspan, width
thead	
tr	
ul	

注意: 仅支持如下字符实体。其他字符实体会导致组件无法渲染。

显示结果	描述	实体名称
	空格	
<	小于号	&lt;
>	大于号	&gt;
&	和号	&amp;
"	引号	&quot;
'	撇号	&apos;

文本节点

属性名	类型	说明	必填	备注
type	String	节点类型	是	type 为 text
text	String	文本	是	-

3.1.5. 表单组件

3.1.5.1. button 按钮

需要重点强调该操作并且引导用户去点击的入口通过按钮表达。

扫码体验

示例代码

```
<!-- API-DEMO page/component/button/button.axml -->
<view class="page">
  <view class="page-description">按钮</view>
  <view class="page-section">
    <view class="page-section-title">type-primary/ghost</view>
    <view class="page-section-demo">
      <button type="primary">主要操作 Normal</button>
      <button type="primary" loading>操作</button>
      <button type="primary" disabled>主要操作 Disable</button>
      <button type="ghost">ghost操作</button>
      <button type="ghost" loading>ghost操作</button>
      <button type="ghost" disabled>ghost操作 Disable</button>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">type-default</view>
    <view class="page-section-demo">
      <button data-aspm-click="xxx">辅助操作 Normal</button>
      <button disabled>辅助操作 Disable</button>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">type-warn</view>
    <view class="page-section-demo">
      <button type="warn">警告类操作 Normal</button>
      <button type="warn" disabled>警告类操作 Disable</button>
      <button type="warn" hover-class="red">hover-red</button>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">Size</view>
    <view class="page-section-demo">
      <button size="mini" loading>提交</button>
      <button style="margin-left: 10px;" type="primary" size="mini">选项</button>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">open</view>
    <view class="page-section-demo">
      <button open-type="share">share</button>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">Form</view>
    <view class="page-section-demo">
      <form onSubmit="onSubmit" onReset="onReset">
        <button form-type="submit" type="primary">submit</button>
        <button form-type="reset">reset</button>
      </form>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/button/button.js
Page({
  data: {},
  onSubmit() {
    my.alert({ title: 'You click submit' });
  },
  onReset() {
    my.alert({ title: 'You click reset' });
  },
});
```

```
/* API-DEMO page/component/button/button.acss */
.red {
  background-color: red;
  border-color: red;
  color: #fff;
}
button + button {
  margin-top: 32rpx;
}
```

属性

		属性	类型	描述
size	String	default	有效值 default, mini（小尺寸）。	
type	String	default	按钮的样式类型，有效值 primary，default,，warn。	
plain	Boolean	false	是否镂空（ghost 与 plain 等效）	
disabled	Boolean	false	是否禁用	
loading	Boolean	false	按钮文字前是否带 loading 图标。	
hover-class	String	button-hover	按钮按下去的样式类。button-hover 默认为 `{background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;}` ，hover-class="none" 时表示没有被点击效果。	
hover-start-time	Number	20	按住后多少时间后出现点击状态，单位毫秒。	
hover-stay-time	Number	70	手指松开后点击状态保留时间，单位毫秒。	
hover-stop-propagation	Boolean	false	是否阻止当前元素的祖先元素出现被点击样式。	
form-type	String	-	有效值：submit，reset，用于 form 表单 组件，点击分别会触发 submit/reset 事件。	
open-type	String	-	开放能力	

scope	String	-	当 open-type 为 getAuthorize 时有效。	
onTap	EventHandle	-	点击	
app-parameter	String	-	打开 APP 时，向 APP 传递的参数。	
public-id	String	-	生活号 id，必须是当前小程序同主体且已关联的生活号，open-type="lifestyle" 时有效。	

open-type 有效值

值	说明
share	触发 自定义分享，可使用 my.canIUse('button.open-type.share') 判断
getAuthorize	支持小程序授权，可使用 my.canIUse('button.open-type.getAuthorize') 判断
contactShare	分享到通讯录好友，可使用 my.canIUse('button.open-type.contactShare') 判断
lifestyle	关注生活号，可使用 my.canIUse('button.open-type.lifestyle') 判断

scope 有效值

当 open-type 为 getAuthorize 时，可以设置 scope 为以下值：

值	说明
phoneNumber	唤起授权界面，用户可以授权小程序获取用户手机号。
getOpenUserInfo	唤起授权界面，用户可以授权小程序获取支付宝会员的基础信息。

常见问题

使用 button 点击授权获取手机号，服务端要怎么解密？

请参见文档 内容加密接入指引。

button 如何去除默认边框？

设置下 class 改为：border: 0; padding: 0;

如何实现自定义分享中的 button: 页面分享按钮触发？

通过给 button 组件设置属性 open-type="share"，可以在用户点击按钮后触发。

相关文档

- contact-button 智能客服
- my.canIUse

- my.getPhoneNumber
- my.getOpenUserInfo

3.1.5.2. form 表单

表单。用于将组件内的用户输入的 textarea、 switch、 input 、checkbox、slider、radio、picker 等组件提交。

当点击 form 表单中 form-type 为 submit 的 button 组件时，会将表单组件值进行提交，需要在表单组件中加上 name 来作为 key。

说明：

- 预览效果建议以真机为准。
- 目前还不支持form表单渲染。

扫码体验

示例代码

```
<!-- API-DEMO page/component/form/form.axml -->
<view class="page">
  <view class="page-description">表单</view>
  <form onSubmit="onSubmit" onReset="onReset">
    <view class="page-section">
      <view class="page-section-title">Slider</view>
      <view class="page-section-demo">
        <slider value="80" name="slider" show-value />
      </view>
    </view>
    <view class="page-section">
      <view class="form-row">
        <view class="form-row-label">Switch</view>
        <view class="form-row-content" style="text-align: right">
          <switch name="switch" />
        </view>
      </view>
      <view class="form-line" />
      <view class="form-row">
        <view class="form-row-label">Input</view>
        <view class="form-row-content">
          <input name="input" class="input" placeholder="input something" />
        </view>
      </view>
    </view>
    <view class="page-section">
      <view class="page-section-title">Radio</view>
      <view class="page-section-demo">
        <radio-group name="radio-group">
          <label><radio value="radio1" />radio1</label>
          <label><radio value="radio2" />radio2</label>
        </radio-group>
      </view>
    </view>
    <view class="page-section">
      <view class="page-section-title">Checkbox</view>
      <view class="page-section-demo">
        <checkbox-group name="checkbox">
          <label><checkbox value="checkbox1" />checkbox1</label>
          <label><checkbox value="checkbox2" />checkbox2</label>
        </checkbox-group>
      </view>
      <view class="page-section-btns">
        <view><button type="ghost" size="mini" formType="reset">Reset</button></view>
        <view><button type="primary" size="mini" data-id="121" formType="submit">Submit</button></view>
      </view>
    </view>
  </form>
</view>
```

```
// API-DEMO page/component/form/form.js
Page({
  data: {},
  onSubmit(e) {
    my.alert({
      content: `数据: ${JSON.stringify(e.detail.value)}`,
    });
  },
  onReset() {
  },
});
```

```
/* API-DEMO page/component/form/form.acss */
button + button {
  margin-top: 32rpx;
}
```

属性

属性	类型	默认值	描述
report-submit	boolean	-	onSubmit 回调是否返回 formId，用于发送 模板消息，使用前可使用 canIUse('form.report-submit')判断是否支持
onSubmit	EventHandle	-	携带 form 中的数据触发 submit 事件，`event.detail = {value : {'slider': '80'}, buttonTarget: {'dataset': 'buttonDataset'} }`(可以在 submit 按钮上添加自定义参数)
onReset	EventHandle	-	表单重置时会触发 reset 事件

常见问题

formId 返回值是否可以自定义？

formId 返回值不支持自定义，设置完成对应属性 report-submit 后支付宝返回。

支付宝小程序消息推送获取的 formId 有效期是多久？用一次会失效一次吗？

formId 有效期是7天，可在 7 天内向用户推送消息。一个 formId 可发送三次。

小程序接入，付款后调用消息下发返回 formId 为何显示不合法？

商户的模板是表单类型，表单类的模板消息只允许使用表单组件生成的 formId 发送。

小程序 form 表单静默触发吗？

不支持，需要用户点击触发。

3.1.5.3. label 标签

简介

用于改进表单组件的可用性。使用 for 属性找到对应组件的 id，或者将组件放在该标签下。当点击时，就会聚焦对应的组件。for 优先级高于内部组件，内部有多个组件的时候默认触发第一个组件。

使用限制

目前可以绑定的组件有：checkbox , radio ,input, textarea。

扫码体验

示例代码

```
<!-- API-DEMO page/component/label/label.axml -->
<view class="page">
  <view class="page-section">
    <view class="page-section-title">Checkbox</view>
    <view class="page-section-demo">
      <checkbox-group>
        <view>
          <label>
            <checkbox value="AngularJS" />
            <text> AngularJS</text>
          </label>
        </view>
        <view>
          <label>
            <checkbox value="React" />
            <text> React</text>
          </label>
        </view>
      </checkbox-group>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">Radio</view>
    <view class="page-section-demo">
      <radio-group>
        <view>
          <radio id="AngularJS" value="AngularJS" />
          <label for="AngularJS">AngularJS</label>
        </view>
        <view>
          <radio id="React" value="React" />
          <label for="React">React</label>
        </view>
      </radio-group>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">label中有多个 Checkbox 时，点击 label 关联的元素选择第一个，例如点击的“Click Me”</vi
ew>
    <view class="page-section-demo">
      <label>
        <checkbox>选中我</checkbox>
        <checkbox>选不中</checkbox>
        <checkbox>选不中</checkbox>
        <checkbox>选不中</checkbox>
        <view>
          <text>Click Me</text>
        </view>
      </label>
    </view>
  </view>
</view>
```

```
/* API-DEMO page/component/label/label.acss */
checkbox-group > view,
radio-group > view {
  margin-bottom: 12rpx;
}
```

属性	类型	描述
for	String	绑定组件的 ID。

3.1.5.4. input 输入框

输入框，可设置输入内容的类型、长度、显示形式等。当用户需要输入文字内容时点击文本框，它将自动打开键盘。使用文本字段来请求少量信息。

注意：

- iOS系统阿里云客户端版本不支持 focus=true 自动唤起。
- 小程序中 input 如果父类是 position: fixed，可以加上 enableNative="{{false}}"，解决输入框错位/光标上移问题。

扫码体验

示例代码

```
<!-- API-DEMO page/component/input/input.axml -->
<view class="page">
  <view class="page-description">输入框</view>
  <view class="page-section">
    <view class="form-row">
      <view class="form-row-label">受控聚焦</view>
      <view class="form-row-content">
        <input class="input" focus="{{focus}}" onFocus="onFocus" onBlur="onBlur" placeholder="input something" />
      </view>
    </view>
    <view class="page-section-btns">
      <button size="mini" onTap="bindButtonTap">聚焦</button>
    </view>
  </view>
  <view class="page-section">
    <view class="form-row">
      <view class="form-row-label"><label for="controlled">显示输入</label></view>
      <view class="form-row-content">
        <input class="input" id="controlled" onInput="bindKeyInput" placeholder="show input content" />
      </view>
    </view>
    <view class="extra-info">你输入的是：{{inputValue}}</view>
  </view>
  <view class="page-section">
    <view class="form-row">
      <view class="form-row-label">最大长度</view>
      <view class="form-row-content">
        <input class="input" maxlength="10" placeholder="maxlength 10" />
```

```
        <input class="input" maxlength="10" placeholder="maxlength 10" />
      </view>
    </view>
    <view class="form-line" />
    <view class="form-row">
      <view class="form-row-label">收起键盘</view>
      <view class="form-row-content">
        <input class="input" onInput="bindHideKeyboard" placeholder="输入 123 自动收起键盘" />
      </view>
    </view>
    <view class="form-line" />
    <view class="form-row">
      <view class="form-row-label">输入密码</view>
      <view class="form-row-content">
        <input class="input" password type="text" placeholder="密码输入框" />
      </view>
    </view>
    <view class="form-line" />
    <view class="form-row">
      <view class="form-row-label">输入数字</view>
      <view class="form-row-content">
        <input class="input" type="number" placeholder="数字输入框" />
      </view>
    </view>
    <view class="form-line" />
    <view class="form-row">
      <view class="form-row-label">小数点键盘</view>
      <view class="form-row-content">
        <input class="input" type="digit" placeholder="带小数点的数字键盘" />
      </view>
    </view>
    <view class="form-line" />
    <view class="form-row">
      <view class="form-row-label">身份证键盘</view>
      <view class="form-row-content">
        <input class="input" type="idcard" placeholder="身份证输入键盘" />
      </view>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">搜索框</view>
    <view class="page-section-demo">
      <view class="search-outer">
        <input
          class="search-input"
          placeholder="搜索"
          value="{{search}}"
          onConfirm="doneSearch"
          onInput="handleSearch"
        />
        <text class="search-cancel" onTap="clearSearch">取消</text>
      </view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/input/input.js
Page({
  data: {
    focus: false,
    inputValue: '',
  },
  bindButtonTap() {
    // blur 事件和这个冲突
    setTimeout(() => {
      this.onFocus();
    }, 100);
  },
  onFocus() {
    this.setData({
      focus: true,
    });
  },
  onBlur() {
    this.setData({
      focus: false,
    });
  },
  bindKeyInput(e) {
    this.setData({
      inputValue: e.detail.value,
    });
  },
  bindHideKeyboard(e) {
    if (e.detail.value === '123') {
      // 收起键盘
      my.hideKeyboard();
    }
  },
  handleSearch(e) {
    console.log('search', e.detail.value);
    this.setData({
      search: e.detail.value,
    });
  },
  doneSearch() {
    console.log('doneSearch', this.data.search);
    my.hideKeyboard();
  },
  clearSearch() {
    console.log('clear search', this.data.search);
    this.setData({
      search: '',
    });
  },
});
```

```
/* API-DEMO page/component/input/input.acss */
.extra-info {
  border-top: 1px solid #ddd;
  margin-left: 30rpx;
  padding: 20rpx 0;
  overflow: auto;
}
.search-outer {
  box-sizing: border-box;
  display:flex;
  height:40px;
  overflow:hidden;
  padding: 8px;
  border-bottom: 1px solid #ddd;
  background-color: #efeff4;
}
.search-outer * {
  box-sizing: border-box;
}
.search-input {
  flex:1;
  text-align: left;
  display: block;
  color: #000;
  height: 24px;
  font-size: 15px;
  background-color: #fff;
  border-color: transparent;
}
.search-input:focus + .search-cancel {
  margin-right:0;
  opacity: 1;
}
.search-cancel {
  margin-right:-40px;
  display: inline-block;
  opacity: 0;
  padding-left: 8px;
  height: 24px;
  line-height: 24px;
  font-size: 16px;
  color: #108ee9;
  text-align: right;
  transition: margin-right .3s,opacity .3s;
  transition-delay: .1s;
}
```

属性

属性	类型	默认值	描述
value	String	-	初始内容
name	String	-	组件名字，用于表单提交获取数据
type	String	text	input 的类型，有效值：`text`、`number`、`idcard`、`digit`(可以唤起带有小数点的数字键盘)、`numberpad`、`digitpad`、`idcardpad`。
password	Boolean	false	是否是密码类型
placeholder	String	-	占位符
placeholder-style	String	-	指定 placeholder 的样式，可设置间距

placeholder-class	String	-	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大长度
focus	Boolean	false	获取焦点（iOS无效）
confirm-type	String	done	设置键盘右下角按钮的文字，有效值：done（显示“完成”）、go（显示“前往”）、next（显示“下一个”）、search（显示“搜索”）、send（显示“发送”），平台不同显示的文字略有差异。**注意：只有在 type=text 时有效**
confirm-hold	Boolean	false	点击键盘右下角按钮时是否保持键盘不收起状态
cursor	Number	-	指定 focus 时的光标位置
selection-start	Number	-1	获取光标时，选中文本对应的焦点光标起始位置，需要和 selection-end 配合使用
selection-end	Number	-1	获取光标时，选中文本对应的焦点光标结束位置，需要和 selection-start 配合使用
randomNumber	Boolean	false	当 type 为 number, digit, idcard 数字键盘是否随机排列
controlled	Boolean	false	是否为受控组件。为 true 时，value 内容会完全受 setData 控制
onInput	EventHandle	-	键盘输入时触发 input 事件，`event.detail = {value: value}`
onConfirm	EventHandle	-	点击键盘完成时触发，`event.detail = {value: value}`
onFocus	EventHandle	-	聚焦时触发，`event.detail = {value: value}`
onBlur	EventHandle	-	失去焦点时触发（仅支持真机），`event.detail = {value: value}`

常见问题

为何 input 输入框聚焦的时候出现白屏，只有键盘弹出来？

因为使用定位导致键盘把页面 input 内容顶上去了，建议使用 SearchBar 搜索框。

为何 input 输入的内容没有在输入框显示？

商户有使用 fixed 定位 建议更换 fixed 定位使用相对或者绝对定位。

小程序 input 输入框获取焦点时会向上推输入框，能否固定？

暂不支持 ，input 是在聚焦的时候弹起 ，失去焦点的时候收起，若让键盘处于一直弹起状态，就要保证 input 一直聚焦。

input 输入框弹起键盘有遮挡，影响其他标签控件触发点击事件？

建议修改自定义 view 样式。

input 输入框是否支持点击事件，比如 click、tap、touchstart？

暂时不支持，可以考虑外嵌一层 view，利用 view 的 onTap 事件实现。

input 如何用 js 代码清空数据？

需要添加属性 controlled="{{true}}，也可以在 onInput 事件里把输入的值通过 setData 再赋值给 value，再去 setData 设置 value。

```
/axml<br><input class="internet_input" value="{{textValue}}" onInput="keyNum" controlled={{true}} type="text"  />
//input怎么用js清空
keyNum() {
      this.setData({
        textValue:''
    })
 }
```

input 如何进行监听，如果出现不能监听问题如何解决？

可以使用 input 的 onInput 事件监听输入值，通过 e.detail.value 打印出输入值进行正则表达式匹配校验。详情请见**示例代码**。

如何判断 input 的 value 值是不是符合正则表示式？

使用 var reg = new RegExp("\w+\s", "g"); getRegExp() 需要在 sjs 中使用。 sjs 脚本不能直接在 js 中引入调用。

父组件如何调用子组件的 input 事件？

请参见 组件对象。

input 内容跳动、延迟如何处理？

可以使用防抖动，示例代码如下：

```
var timer = null
element.input = function () {
    clearTimeout(timer) // 每次进来的时候都将之前的清除掉，如果还没到一秒的时候就将之前的清除掉，这样就不会触发之前setTimeout绑定的事件， 如果超过一秒，之前的事件就会被触发下次进来的时候同样清除之前的timer
    timer = setTimeout(function () {
        // 在这里进行我们的操作  这样就不会频繁的进行我们这里面的操作了
    }, 1000)
}
```

3.1.5.5. textarea 多行输入框

多行输入框，可输入多行内容。

注意：

- 无法通过 textarea 获取键盘高度。
- iOS 系统阿里云客户端不支持 focus=true 自动唤起。
- 可以使用 my.hideKeyboard 隐藏键盘。
- 添加属性 controlled="{{true}}" 表示 value 内容会完全受 setData 控制
- 可以在 input 组件中加上 enableNative="{{false}}"，避免 textarea 弹出键盘后内容被顶起。
- 加上 enableNative="{{false}}" 解决安卓系统下 textarea 获取焦点的时候文字消失问题。

扫码体验

示例代码

```
<!-- API-DEMO page/component/textarea/textarea.axml -->
<view class="page">
  <view class="page-description">文本框</view>
  <view class="page-section">
    <view class="page-section-title">受控聚焦</view>
    <view class="page-section-demo">
      <textarea focus="{{focus}}" onFocus="onFocus" onBlur="onBlur" placeholder="Please input something" />
    </view>
    <view class="page-section-btns">
      <button type="default" size="mini" onTap="bindButtonTap">聚焦</button>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">自适应高度</view>
    <view class="page-section-demo">
      <textarea onBlur="bindTextAreaBlur" auto-height placeholder="Please input something" />
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">结合表单</view>
    <form onSubmit="bindFormSubmit">
      <view class="page-section-demo">
        <textarea name="textarea" placeholder="Please input something"  />
      </view>
      <view class="page-section-btns">
        <button form-type="submit" size="mini" type="primary">提交</button>
      </view>
    </form>
  </view>
</view>
```

```
// API-DEMO page/component/textarea/textarea.js
Page({
  data: {
    height: 20,
    focus: false,
  },
  bindButtonTap() {
    this.onFocus();
  },
  onFocus() {
    this.setData({
      focus: true,
    });
  },
  onBlur() {
    this.setData({
      focus: false,
    });
  },
  bindTextAreaBlur(e) {
    console.log(e.detail.value);
  },
  bindFormSubmit(e) {
    my.alert({
      content: e.detail.value.textarea,
    });
  },
});
```

属性

属性	类型	默认值	描述

name	String	-	组件名字，用于表单提交获取数据
value	String	-	初始内容
placeholder	String	-	占位符
placeholder-style	String	-	指定 placeholder 的样式
placeholder-class	String	-	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大长度，当设置为-1时不限制最大长度
focus	Boolean	false	获取焦点（iOS无效）
auto-height	Boolean	false	是否自动增高
show-count	Boolean	true	是否渲染字数统计功能（**是否删除默认计数器/是否显示字数统计**）
controlled	Boolean	false	是否为受控组件。为 true 时，value 内容会完全受 setData 控制
onInput	EventHandle	-	键盘输入时触发，`event.detail = {value: value}`，可以直接 return 一个字符串以替换输入框的内容
onFocus	EventHandle	-	输入框聚焦时触发 `event.detail = {value: value}`
onBlur	EventHandle	-	输入框失去焦点时触发，`event.detail = {value: value}`
onConfirm	EventHandle	-	点击完成时触发，`event.detail = {value: value}`

3.1.5.6. radio 单选按钮

单选按钮。

注意：

- 不支持修改 radio 选中后的宽高。
- radio 按钮不能嵌套 text 标签，支持平行关系。

扫码体验

示例代码

```
<!-- API-DEMO page/component/radio/radio.axml -->
<view class="page">
  <view class="page-description">单选框</view>
  <view class="page-section">
    <view class="section section_gap">
      <form onSubmit="onSubmit" onReset="onReset">
        <view class="page-section-demo">
          <radio-group class="radio-group" onChange="radioChange" name="lib">
            <label class="radio" a:for="{{items}}" key="label-{{index}}">
              <radio value="{{item.name}}" checked="{{item.checked}}" disabled="{{item.disabled}}" />
              <text class="radio-text">{{item.value}}</text>
            </label>
          </radio-group>
        </view>
        <view class="page-section-btns">
          <view><button size="mini" type="ghost" formType="reset">reset</button></view>
          <view><button size="mini" type="primary" formType="submit">submit</button></view>
        </view>
      </form>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/radio/radio.js
Page({
  data: {
    items: [
      { name: 'angular', value: 'AngularJS' },
      { name: 'react', value: 'React', checked: true },
      { name: 'polymer', value: 'Polymer' },
      { name: 'vue', value: 'Vue.js' },
      { name: 'ember', value: 'Ember.js' },
      { name: 'backbone', value: 'Backbone.js', disabled: true },
    ],
  },
  onSubmit(e) {
    my.alert({
      content: e.detail.value.lib,
    });
    console.log('onSubmit', e.detail);
  },
  onReset(e) {
    console.log('onReset', e);
  },
  radioChange(e) {
    console.log('你选择的框架是: ', e.detail.value);
  },
});
```

```
/* API-DEMO page/component/radio/radio.acss */
.radio {
  display: block;
  margin-bottom: 20rpx;
}
.radio-text {
  line-height: 1.8;
}
```

属性

属性	类型	默认值	描述
value	String	-	组件值，选中时 change 事件会携带的 value
checked	Boolean	false	当前是否选中
disabled	Boolean	false	是否禁用
color	String	-	radio 的颜色，同 CSS 色值

3.1.5.7. checkbox 多项选择器

多选项目。

注意：

- checkbox 没有源码。
- checkbox 不支持修改选中的背景颜色样式。

扫码体验

示例代码

```
<!-- API-DEMO page/component/checkbox/checkbox.axml -->
<view class="page">
  <view class="page-description">多项选择器</view>
  <form onSubmit="onSubmit" onReset="onReset">
    <view class="page-section">
      <view class="page-section-title">选择你用过的框架：</view>
      <view class="page-section-demo">
        <checkbox-group onChange="onChange" name="libs">
          <label class="checkbox" a:for="{{items}}" key="label-{{index}}">
            <checkbox value="{{item.name}}" checked="{{item.checked}}" disabled="{{item.disabled}}" />
            <text class="checkbox-text">{{item.value}}</text>
          </label>
        </checkbox-group>
      </view>
      <view class="page-section-btns">
        <view><button type="ghost" size="mini" formType="reset">reset</button></view>
        <view><button type="primary" size="mini" formType="submit">submit</button></view>
      </view>
    </view>
  </form>
</view>
```

```
// API-DEMO page/component/checkbox/checkbox.js
Page({
  data: {
    items: [
      { name: 'angular', value: 'AngularJS' },
      { name: 'react', value: 'React', checked: true },
      { name: 'polymer', value: 'Polymer' },
      { name: 'vue', value: 'Vue.js' },
      { name: 'ember', value: 'Ember.js' },
      { name: 'backbone', value: 'Backbone.js', disabled: true },
    ],
  },
  onSubmit(e) {
    console.log('onSubmit', e);
    my.alert({
      content: `你选择的框架是 ${e.detail.value.libs.join(', ')}`,
    });
  },
  onReset(e) {
    console.log('onReset', e);
  },
  onChange(e) {
    console.log(e);
  },
});
```

```
/* API-DEMO page/component/checkbox/checkbox.acss */
.checkbox {
  display: block;
  margin-bottom: 20rpx;
}
button + button {
  margin-top: 32rpx;
}
.checkbox-text {
  font-size:34rpx;
  line-height: 1.2;
}
```

属性

属性	类型	默认值	描述
value	String	-	组件值，选中时 change 事件会携带的 value
checked	Boolean	false	当前是否选中，可用来设置默认选中
disabled	Boolean	false	是否禁用
onChange	EventHandle	-	组件发生改变时触发，`detail = {value: 该 checkbox 是否 checked }`
color	String	-	checkbox 的颜色，同 CSS 色值

3.1.5.8. switch 单选开关

单选开关。

注意：

- iOS 和安卓展现样式有所差异。iOS 单选开关为圆形；安卓单选开关为方形。
- 不支持自定义 switch 样式，如大小等。

扫码体验

示例代码

```
<!-- API-DEMO page/component/switch/switch.axml -->
<view class="page">
  <view class="page-description">开关</view>
  <view class="page-section">
    <view class="page-section-demo switch-list">
      <view class="switch-item">
        <switch checked onChange="switch1Change" aria-label="{{switch1 ? 'switch opened' : 'switch closed'}}" />
      </view>
      <view class="switch-item">
        <switch onChange="switch2Change"/>
      </view>
      <view class="switch-item">
        <switch color="red" checked />
      </view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/switch/switch.js
Page({
  data: {
    switch1: true,
  },
  switch1Change(e) {
    console.log('switch1 发生 change 事件，携带值为', e.detail.value);
    this.setData({
      switch1: e.detail.value,
    });
  },
  switch2Change(e){
    console.log('switch2 发生 change 事件，携带值为', e.detail.value);
  },
});
```

```
/* API-DEMO page/component/switch/switch.acss */
.switch-item + .switch-item {
  margin-top: 20rpx;
}
```

属性

属性	类型	默认值	描述
name	String	-	组件名字，用于表单提交获取数据
checked	Boolean	-	是否选中
disabled	Boolean	-	是否禁用
color	String	-	组件颜色，同 CSS 色值
onChange	EventHandle	-	checked 改变时触发，`event.detail={ value:checked}`
controlled	Boolean	false	是否为受控组件，为 true 时，checked 会完全受 setData 控制

3.1.5.9. slider 滑动选择器

滑动选择器。

扫码体验

示例代码

```
<!-- API-DEMO page/component/slider/slider.axml -->
<view class="page">
  <view class="page-description">滑块</view>
  <view class="page-section">
    <view class="page-section-title">设置step</view>
    <view class="page-section-demo">
      <slider value="5" onChange="slider2change" step="5"/>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">设置最小/最大值范围</view>
    <view class="page-section-demo">
      <slider value="33" onChange="slider4change" min="25" max="50" show-value/>
    </view>
  </view>
  <view class="page-section">
    <view class="page-section-title">自定义样式</view>
    <view class="page-section-demo">
      <slider value="33" onChange="slider4change" min="25" max="50" show-value
      backgroundColor="#FFAA00" activeColor="#00aaee" trackSize="2" handleSize="6" handleColor="blue" />
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/slider/slider.js
const pageData = {};
for (let i = 1; i < 5; ++i) {
  (function (index) {
    pageData['slider' + index + 'change'] = function (e) {
      console.log('slider' + index + '发生 change 事件，携带值为', e.detail.value);
    };
  })(i);
}
Page(pageData);
```

属性

属性	类型	默认值	描述
name	String	-	组件名字，用于表单提交获取数据
min	Number	0	最小值
max	Number	100	最大值
step	Number	1	步长，值必须大于 0，并可被(max - min)整除
disabled	Boolean	false	是否禁用

value	Number	0	当前取值
show-value	Boolean	false	是否显示当前 value
active-color	String	#108ee9	已选择的颜色，同 CSS 色值
background-color	String	#ddd	背景条颜色，同 CSS 色值
track-size	Number	4	轨道线条高度
handle-size	Number	22	滑块大小
handle-color	String	#fff	滑块填充色，同 CSS 色值
onChange	EventHandle	-	完成一次拖动后触发，`event.detail = {value: value}`
onChanging	EventHandle	-	拖动过程中触发的事件，`event.detail = {value: value}`

3.1.5.10. picker-view 滚动选择器

嵌入页面的滚动选择器。 其中只可放置 picker-view-column 组件，其它节点不会显示。

注意：

- 该组件内部请勿放入 hidden 或 display none 的节点，需要隐藏请用 a:if 切换。
- 不支持背景色设置成透明，可以修改颜色。
- 可以先获取索引 index 然后通过 index 获取数组中值。

扫码体验

示例代码

```
<!-- API-DEMO page/component/picker-view/picker-view.axml -->
<view class="page">
  <view class="page-description">嵌入页面的滚动选择器</view>
  <view class="page-section">
    <view class="page-section-demo">
      <picker-view value="{{value}}" onChange="onChange" class="my-picker">
        <picker-view-column>
          <view>2011</view>
          <view>2012</view>
          <view>2013</view>
          <view>2014</view>
          <view>2015</view>
          <view>2016</view>
          <view>2017</view>
          <view>2018</view>
        </picker-view-column>
        <picker-view-column>
          <view>春</view>
          <view>夏</view>
          <view>秋</view>
          <view>冬</view>
        </picker-view-column>
      </picker-view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/picker-view/picker-view.js
Page({
  data: {},
  onChange(e) {
    console.log(e.detail.value);
    this.setData({
      value: e.detail.value,
    });
  },
});
```

```
/* API-DEMO page/component/picker-view/picker-view.acss */
.my-picker {
  background: #EFEFF4;
}
```

picker-view-column 滚动选择器子项

滚动选择器子项。仅可放置于 picker-view 中，其孩子节点的高度会自动设置成与 picker-view 的选中框的高度一致。 **说明：** 该组件内部请勿放入 hidden 或 display none 的节点，需要隐藏请用 `a:if` 切换，即： 推荐：

```
<view a:if="{{xx}}"><picker-view/></view>
```

不要：

```
<view hidden><picker-view/></view>
```

属性

属性	类型	默认值	描述
value	Number Array	-	数字表示 picker-view-column 中对应的 index （从 0 开始）
indicator-style	String	-	选中框样式

indicator-class	String	-	选中框的类名
mask-style	String	-	蒙层的样式
mask-class	String	-	蒙层的类名
onChange	EventHandle	-	滚动选择 value 改变时触发，`event.detail = {value: value}` value为数组，表示 picker-view 内的 picker-view-column index 索引，从 0 开始

3.1.5.11. picker 底部弹起的滚动选择器

选择器包括一个或多个不同值的可滚动列表，每个值可以在视图的中心以较暗的文本形式显示。当用户正在编辑字段或点击菜单时，选择器通常会从屏幕底部弹起(iOS)。

注意：

- picker 组件在 iOS 系统中从底部弹出，在安卓系统中从中间弹出。
- picker 组件不支持多列选择，可以使用 picker-view 组件。
- 通过 my.multiLevelSelect 调用级联选择。
- 可以通过 my.datePicker 打开日期选择列表。

扫码体验

示例代码

```
<!-- API-DEMO page/component/picker/picker.axml -->
<view class="page">
  <view class="page-description">选择器</view>
  <view class="page-section">
    <picker onChange="bindPickerChange" value="{{index}}" range="{{array}}">
      <view class="row">
        <view class="row-title">地区选择器</view>
        <view class="row-extra">当前选择：{{array[index]}}</view>
        <image class="row-arrow" src="/image/arrowright.png" mode="aspectFill" />
      </view>
    </picker>
  </view>
  <view class="page-section">
    <picker onChange="bindObjPickerChange" value="{{arrIndex}}" range="{{objectArray}}" range-key="name">
      <view class="row">
        <view class="row-title">ObjectArray</view>
        <view class="row-extra">当前选择：{{objectArray[arrIndex].name}}</view>
        <image class="row-arrow" src="/image/arrowright.png" mode="aspectFill" />
      </view>
    </picker>
  </view>
</view>
```

```
// API-DEMO page/component/picker/picker.js
Page({
  data: {
    array: ['中国', '美国', '巴西', '日本'],
    objectArray: [
      {
        id: 0,
        name: '美国',
      },
      {
        id: 1,
        name: '中国',
      },
      {
        id: 2,
        name: '巴西',
      },
      {
        id: 3,
        name: '日本',
      },
    ],
    arrIndex: 0,
    index: 0
  },
  bindPickerChange(e) {
    console.log('picker发送选择改变，携带值为', e.detail.value);
    this.setData({
      index: e.detail.value,
    });
  },
  bindObjPickerChange(e) {
    console.log('picker发送选择改变，携带值为', e.detail.value);
    this.setData({
      arrIndex: e.detail.value,
    });
  },
});
```

```
/* API-DEMO page/component/picker/picker.acss */
.date-radio {
  padding: 26rpx;
}
.date-radio label + label {
  margin-left: 20rpx;
}
.row {
  display: flex;
  align-items: center;
  padding: 0 30rpx;
}
.row-title {
  flex: 1;
  padding-top: 28rpx;
  padding-bottom: 28rpx;
  font-size: 34rpx;
  color: #000;
}
.row-extra {
  flex-basis: initial;
  font-size: 32rpx;
  color: #888;
}
.row-arrow {
  width: 32rpx;
  height: 32rpx;
  margin-left: 16rpx;
}
```

属性

属性	类型	默认值	描述
range	String[] / Object[]	[]	String[] 时表示可选择的字符串列表；Object[] 时需指定 range-key 表示可选择的字段
range-key	String	-	当 range 是一个 Object[] 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示内容
value	Number	-	表示选择了 range 中的第几个（下标从 0 开始）。
onChange	EventHandle	-	value 改变时触发，`event.detail = {value: value}`
disabled	Boolean	false	是否禁用

相关文档

- 选择日期 my.datePicker
- picker-view 滚动选择器
- 级联选择 my.multiLevelSelect

3.1.6. 导航

3.1.6.1. navigator 页面链接

页面链接。

注意：

- navigator 组件不支持 ontap 事件。
- 导航栏相关设置请参见 my.setNavigationBar。

扫码体验

示例代码

```
<!-- API-DEMO page/component/navigator/navigator.axml -->
<view class="page">
  <view class="page-description">导航栏</view>
  <navigator open-type="navigate" url="./navigate" hover-class="navigator-hover">跳转到新页面</navigator>
  <navigator open-type="redirect" url="./redirect" hover-class="navigator-hover">在当前页打开</navigator>
  <navigator open-type="switchTab" url="/page/API/index/index" hover-class="navigator-hover">跳转到另外一个 Tab - API
</navigator>
  <navigator open-type="reLaunch" url="/page/component/index" hover-class="navigator-hover">重新打开</navigator>
  <navigator open-type="navigateBack" hover-class="navigator-hover">返回上一页面</navigator>
</view>
```

```
// API-DEMO page/component/navigator/navigator.js
Page({});
```

```
/* API-DEMO page/component/navigator/navigator.acss */
navigator {
  background-color: lightcoral;
  color: #fff;
  margin-bottom: 10rpx;
  padding: 20rpx;
  text-align: center;
}
.navigator-hover {
  background-color: lightskyblue;
  color: #fff;
}
```

属性

属性名	类型	默认值	描述
open-type	String	navigate	跳转方式

hover-class	String	none	点击后的样式类
hover-start-time	Number	-	按住后多少时间后出现点击状态，单位ms
hover-stay-time	Number	-	手指松开后点击状态保留时间，单位ms
url	String	-	当前小程序内的跳转链接

open-type 有效值

属性名	描述
navigate	对应 my.navigateTo 的功能
redirect	对应 my.redirectTo 的功能
switchTab	对应 my.switchTab 的功能
navigateBack	对应 my.navigateBack 的功能
reLaunch	对应 my.reLaunch 的功能

3.1.7. 媒体组件

3.1.7.1. image 图片

图片。

注意：

使用 webview 嵌套 H5 时，若遇到图片资源不显示问题，可参考 配置 H5 白名单流程 获取 H5 页面中所有的域名地址（含图片静态资源的地址），全部加入域名白名单中。

扫码体验

示例代码

```
<!-- API-DEMO page/component/image/image.axml -->
<view class="page">
  <view class="page-description">图片</view>
  <view class="page-section" a:for="{{array}}" a:for-item="item">
    <view class="page-section-title">{{item.text}}</view>
    <view class="page-section-demo" onTap="onTap">
      <image class="image"
        data-name="{{item.mode}}"
        onTap="onTap"
        mode="{{item.mode}}" src="{{src}}" onError="imageError" onLoad="imageLoad" />
    </view>
  </view>
</view>
```

```
// API-DEMO page/component/image/image.js
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: 'widthFix',
      text: 'widthFix：宽度不变，高度自动变化，保持原图宽高比不变',
    }, {
      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: '/image/ant.png',
  },
  imageError(e) {
    console.log('image 发生 error 事件，携带值为', e.detail.errMsg);
  },
  onTap(e) {
    console.log('image 发生 tap 事件', e);
  },
  imageLoad(e) {
    console.log('image 加载成功', e);
  },
});
```

```
/* API-DEMO page/component/image/image.acss */
.page-section-demo {
  display: flex;
  justify-content: space-around;
}
.image {
  background-color: red;
  width: 100px;
  height: 100px;
}
```

属性

属性	类型	默认值	描述
src	String	-	图片地址
mode	String	scaleToFill	图片模式
class	String	外部样式	-
style	String	内联样式	-
lazy-load	Boolean	false	支持图片懒加载，不支持通过 css 来控制 image 展示隐藏的场景。
default-source	String	-	默认图片地址，若设置默认图片地址，会先显示默认图片，等 src 对应的图片加载成功后，再渲染对应的图片。
onLoad	EventHandle	-	图片载入完毕时触发，事件对象 `event.detail = {height: '图片高度px', width: '图片宽度px'}`
onError	EventHandle	-	当图片加载错误时触发，事件对象 `event.detail = {errMsg: 'something wrong'}`
onTap	EventHandle	-	点击图片时触发
catchTap	EventHandle	-	点击图片时触发，阻止事件冒泡

注意：image 组件默认宽度 300px、高度 225px

mode

> *mode 有 13 种模式，其中 4 种是缩放模式，9 种是裁剪模式。*

缩放模式

属性	描述
scaleToFill	不保持纵横比缩放，使图片的宽高完全拉伸至填满 image 元素
aspectFit	保持纵横比缩放，使图片的长边能完全显示出来。也就是说，可以完整地将图片显示出来
aspectFill	保持纵横比缩放，只保证图片的短边能完全显示出来。也就是说，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
widthFix	宽度不变，高度自动变化，保持原图宽高比不变

裁剪模式

属性	描述
top	不缩放图片，只显示顶部区域
bottom	不缩放图片，只显示底部区域
center	不缩放图片，只显示中间区域
left	不缩放图片，只显示左边区域
right	不缩放图片，只显示右边区域
top left	不缩放图片，只显示左上边区域
top right	不缩放图片，只显示右上边区域

bottom left	不缩放图片，只显示左下边区域
bottom right	不缩放图片，只显示右下边区域

说明：图片高度不能设置为 auto，如果需要图片高度为 auto，直接设置 mode 为 widthFix。

相关文档

- my.chooseImage
- my.compressImage
- my.getImageInfo
- my.previewImage
- my.saveImage

3.1.7.2. video 视频

用户可通过 video 组件播放视频。相关 API：my.createVideoContext。

注意：

- css 动画对 video 组件无效。
- 自定义竖屏观看视频时两边出现的白色填充：
  - 如果是因为视频的宽高比跟 video 组件的宽高比不一致，则由 object-fit 来控制；
  - 如果是由于 poster 实际的宽高比跟容器的宽高比不一致，则由 poster-size 来控制。

扫码体验

示例代码

```
// API-DEMO page/component/video/video.axml
<view>
  <video id="myVideo"
    src="{{video.src}}"
    controls="{{video.showAllControls}}"
    loop="{{video.isLooping}}"
    muted="{{video.muteWhenPlaying}}"
    show-fullscreen-btn="{{video.showFullScreenButton}}"
    show-play-btn="{{video.showPlayButton}}"
    show-center-play-btn="{{video.showCenterButton}}"
    object-fit="{{video.objectFit}}"
    autoplay="{{video.autoPlay}}"
    direction="{{video.directionWhenFullScreen}}"
    initial-time="{{video.initTime}}"
    mobilenet_hint_type="{{video.mobilenet_hint_type}}"
    onPlay="onPlay"
    onPause="onPause"
    onEnded="onEnded"
    onError="onPlayError"
    onTimeUpdate="onTimeUpdate"
   />
</view>
```

```
// API-DEMO page/component/video/video.js
Page({
  data: {
       status: "inited",
    time: "0",
    video: {
      src: "XNDM0OTQzMDc2OA==",
      showAllControls: false,
      showPlayButton: false,
      showCenterButton: true,
      showFullScreenButton: true,
      isLooping: false,
      muteWhenPlaying: false,
      initTime: 0,
      objectFit: "contain",
      autoPlay: false,
      directionWhenFullScreen: 90,
      mobilenet_hint_type: 2,
    },
  },
    onPlay(e) {
    console.log('onPlay');
  },
  onPause(e) {
    console.log('onPause');
  },
  onEnded(e) {
    console.log('onEnded');
  },
  onPlayError(e) {
    console.log('onPlayError, e=' + JSON.stringify(e));
  },
  onTimeUpdate(e) {
    console.log('onTimeUpdate:', e.detail.currentTime);
  },
});
```

属性

属性名	类型	默认值	说明
style	String	-	内联样式
class	String	-	外部样式名
src	String	-	要播放的视频资源地址，支持优酷视频编码（阿里云客户端10.1.75），优酷视频上传流程请见 小程序视频播放。src 支持的协议如下：vid/showId:XMzg2Mzc5MzMwMA== apFilePath: https://resource/xxx.video
initial-time	Number	-	指定视频初始播放位置，单位：s
duration	Number	-	指定视频时长，单位s，默认读取视频本身市场信息
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
page-gesture	Boolean	false	非全屏状态下是否支持手势调节音量和亮度
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），90（屏幕逆时针90度），-90（屏幕顺时针90
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-play-btn	Boolean	true	是否显示视频中间的播放按钮
enable-progress-gesture	Boolean	true	是否开启左右滑动控制进度的手势
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（播放后生效，如果视频初始化(未播放)状态，显示视频自动适应宽度），cover：覆盖video标签认宽度300px、高度225px，设置宽高可以通过 style 设置 width 和 height。
poster	String	-	视频封面图的 URL，支持jpg、png等图片，如 **_https://*.jpg。 如果不传的话，默认取视频的首帧图作为封面图。
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）
mobilenet_hint_type	Number	-	移动网络提醒样式：0-不提醒；1-tip提醒；2-阻塞提醒(无消耗流量大小)；3-阻塞提醒(有消耗流量大小)
poster-size	String	contain	当 poster 高宽比跟视频高宽不匹配时，如何显示 poster，设置规则同 background-size 一致
onPlay	EventHandle	-	当开始/继续播放时触发 play 事件
onPause	EventHandle	-	当暂停播放时触发 pause 事件
onEnded	EventHandle	-	当播放到末尾时触发 ended 事件
onTimeUpdate	EventHandle	-	播放进度变化时触发，event.detail = {currentTime: '当前播放时间',userPlayDuration:'用户实际观看时长',videoDuration:'视频总时长'} 。触发频率应该在 250ms 一次
onLoading	EventHandle	-	视频资源加载中
onStop	EventHandle	-	视频播放终止
onRenderStart	EventHandle	-	当视频加载完真正开始播放时触发
onError	EventHandle	-	视频播放出错时触发(errorCode 见下面错误码表)
onFullScreenChange	EventHandle	-	视频进入和退出全屏时触发，event.detail = {fullScreen, direction}，direction的值可能为 vertical 或 horizontal
onTap	EventHandle	-	点击视频 view 时触发，event.detail = {ptInView:{x:0,y:0}}
onUserAction	EventHandle	-	用户操作事件，event.detail = {tag:"mute", value:0}，tag为用户操作的元素，目前支持的 tag 有：play(底部播放按钮)、centerplay(中心播放按钮)、mute(静音按钮)、fullscreen(全屏按钮)、retry(重试按钮)、mobilenetplay(网络提醒的播放按钮)

错误码

错误码	大类	详细说明
1	loading、playing 过程中都可能抛出	未知错误
1002	loading、playing 过程中都可能抛出	播放器内部错误
1005	loading、playing 过程中都可能抛出	网络连接失败
1006	loading 异常	数据源错误
1007	loading 异常	播放器准备失败
1008	loading 异常	网络错误

1009	loading 异常	搜索视频出错（源出错的一种）
1010	loading 异常	准备超时，也可认为是网络太慢或数据源太慢导致的播放失败。
400	loading 异常	读 ups 信息超时
3001	loading 异常	audio 渲染出错
3002	loading 异常	硬解码错误
2004	playing 过程中可能抛出	播放过程中加载时间超时
1023	playing 过程中可能抛出	播放中内部错误（ffmpeg 内错误）

支持的视频封装格式

iOS、Android 支持以下视频封装格式： mp4、mov、m4v、3gp、m3u8、flv

支持的编码格式

iOS、Android 支持以下编码格式： H.264、AAC

常见问题

video 组件中播放的视频，当用户加载观看视频一次后，再次进行观看的时候是拉取的缓存，还是再次使用网络重新加载的？

目前的缓存策略是如果视频是循环播放的，再次观看是拉取的缓存，如果不是循环播放，每次都是网络重新加载。 主要是针对一些循环播放的短视频场景提供缓存能力。

缓存中的视频什么时候会清除掉 ？

页面销毁或者关闭小程序会清除掉。

小程序怎么获取视频时长？

在视频组件onTimeUpdate方法中获取。

video组件，把 loop 字段设置为循环播放，播放一次后，在播放第二次的时候，把视频资源删除，发现无法播放了，和你们文档上的缓存策略好像不一样 ？

虽然再次播放拉取的是缓存中的视频，但是还是会校验视频资源的。

3.1.8. 画布

3.1.8.1. canvas 画布

画布。画布是一个矩形区域，用于在页面上绘制图形，开发者可以控制其每一像素。 canvas 拥有多种绘制路径、矩形、圆形、字符以及添加图像的方法。

相关 API：my.createCanvasContext。

扫码体验

示例代码

```
<!-- API-DEMO page/component/canvas/canvas.axml -->
<view class="page">
  <view class="canvas-view">
    <canvas
      id="canvas"
      width="610"
      height="610"
      class="canvas"
      onTouchStart="log"
      onTouchMove="log"
      onTouchEnd="log"
    />
  </view>
</view>
```

```
// API-DEMO page/component/canvas/canvas.js
Page({
  onReady() {
    this.point = {
      x: Math.random() * 590,
      y: Math.random() * 590,
      dx: Math.random() * 10,
      dy: Math.random() * 10,
      r: Math.round(Math.random() * 255 | 0),
      g: Math.round(Math.random() * 255 | 0),
      b: Math.round(Math.random() * 255 | 0),
    };
    this.interval = setInterval(this.draw.bind(this), 17);
    this.ctx = my.createCanvasContext('canvas');
  },
  draw() {
    const { ctx } = this;
    ctx.setFillStyle('#FFF');
    ctx.fillRect(0, 0, 610, 610);
    ctx.beginPath();
    ctx.arc(this.point.x, this.point.y, 20, 0, 2 * Math.PI);
    ctx.setFillStyle('rgb(' + this.point.r + ', ' + this.point.g + ', ' + this.point.b + ')');
    ctx.fill();
    ctx.draw();
    this.point.x += this.point.dx;
    this.point.y += this.point.dy;
    if (this.point.x <= 10 || this.point.x >= 590) {
      this.point.dx = -this.point.dx;
      this.point.r = Math.round(Math.random() * 255 | 0);
      this.point.g = Math.round(Math.random() * 255 | 0);
      this.point.b = Math.round(Math.random() * 255 | 0);
    }
    if (this.point.y <= 10 || this.point.y >= 590) {
      this.point.dy = -this.point.dy;
      this.point.r = Math.round(Math.random() * 255 | 0);
      this.point.g = Math.round(Math.random() * 255 | 0);
      this.point.b = Math.round(Math.random() * 255 | 0);
    }
  },
  drawBall() {
  },
  log(e) {
    if (e.touches && e.touches[0]) {
      console.log(e.type, e.touches[0].x, e.touches[0].y);
    } else {
      console.log(e.type);
    }
  },
  onUnload() {
    clearInterval(this.interval);
  },
});
```

```
/* API-DEMO page/component/canvas/canvas.acss */
.canvas-view {
  display: flex;
  justify-content: center;
}
.canvas {
  width: 305px;
  height: 305px;
  background-color: #fff;
}
```

属性

属性名	类型	默认值	描述
id	String	-	组件唯一标识符
style	String	-	-
class	String	-	-
width	String	canvas width attribute	-
height	String	canvas height attribute	-
disable-scroll	Boolean	false	禁止屏幕滚动以及下拉刷新
onTap	EventHandle	-	点击
onTouchStart	EventHandle	-	触摸动作开始
onTouchMove	EventHandle	-	触摸后移动
onTouchEnd	EventHandle	-	触摸动作结束
onTouchCancel	EventHandle	-	触摸动作被打断，如来电提醒，弹窗
onLongTap	EventHandle	-	长按 500ms 之后触发，触发了长按事件后进行移动将不会触发屏幕的滚动

注意：

- canvas 标签默认宽度 300px、高度 225px；
- 同一页面中的 id 不可重复；
- 如果需要在高 dpr 下取得更细腻的显示，需要先将 canvas 用属性设置放大，用样式缩小，例如

```
<!-- getSystemInfoSync().pixelRatio === 2 -->
<canvas width="200" height="200" style="width:100px;height:100px;"/>
```

相关文档

- CanvasContext 概览
- my.createCanvasContext

3.2. 扩展组件

3.2.1. 扩展组件概述

应用扩展组件库是基础组件库的重要补充，是基于应用自定义组件规范 开发的一套开源UI组件库，供应用开发者快速复用。

安装

打开 shell，切换到对应的应用所在目录

```
npm install mini-a-ui --save
```

使用

在响应页面的 .json 文件中进行注册，如card组件的注册如下所示：

```
{
  "usingComponents": {
    "card": "mini-antui/es/card/index",
  }
}
```

在 .axml 文件中进行调用：

```
<card
  thumb="{{thumb}}"
  title="卡片标题2"
  subTitle="副标题非必填2"
  onClick="onCardClick"
  info="点击了第二个card"
/>
```

3.2.2. 图表组件

> 说明
>
> 引自：https://github.com/ant-mini-program/mini-chart

小程序图表库

安装

打开 shell，切换到对应的应用所在目录

```
npm install mini-chart --save
```

使用

在响应页面的 .json 文件中进行注册，如card组件的注册如下所示：

```
{
  "usingComponents": {
    "line": "mini-chart/es/line/index"
  }
}
```

在 .js 文件中进行数据获取和设置

```
Page({
  data: {
    cn: 'line',
    categories: ['2017-06-05', '2017-06-06', '2017-06-07', '2017-06-08', '2017-06-09', '2017-06-10', '2017-06-11',
'2017-06-12', '2017-06-13', '2017-06-14', '2017-06-15', '2017-06-16', '2017-06-17', '2017-06-18', '2017-06-19', '20
17-06-20', '2017-06-21', '2017-06-22', '2017-06-23', '2017-06-24', '2017-06-25', '2017-06-26', '2017-06-27', '2017-
06-28', '2017-06-29', '2017-06-30', '2017-07-01', '2017-07-02', '2017-07-03', '2017-07-04', '2017-07-05', '2017-07-
06', '2017-07-07', '2017-07-08', '2017-07-09', '2017-07-10', '2017-07-11', '2017-07-12', '2017-07-13', '2017-07-14'
, '2017-07-15', '2017-07-16', '2017-07-17', '2017-07-18', '2017-07-19', '2017-07-20', '2017-07-21', '2017-07-22', '
2017-07-23', '2017-07-24'],
    series: [
      {
        type: 'a',
        style: 'dash',
        point: {
          size: 3,
          stroke: '#F00',
          lineWidth: 1,
        },
        data: [116, 129, 135, 86, 73, 85, 73, 68, 92, 130, 245, 139, 115, 111, 309, 206, 137, 128, 85, 94, 71, 106,
84, 93, 85, 73, 83, 125, 107, 82, 44, 72, 106, 107, 66, 91, 92, 113, 107, 131, 111, 64, 69, 88, 77, 83, 111, 57, 55
, 60],
      },
      {
        type: 'b',
        color: '#93F',
        style: 'smooth',
        data: [60, 55, 57, 111, 83, 77, 88, 69, 64, 111, 131, 107, 113, 92, 91, 66, 107, 106, 72, 44, 82, 107, 125,
83, 73, 85, 93, 84, 106, 71, 94, 85, 128, 137, 206, 309, 111, 115, 139, 245, 130, 92, 68, 73, 85, 73, 86, 135, 129,
116],
      },
    ],
    xAxis: {
      tickCount: 3,
    },
    yAxis: {
      tickCount: 3,
    },
    legend: {
      position: 'top',
      offsetY: 5
    },
    tooltip: {
      showTitle: true,
      showCrosshairs: true,
    },
  }
});
```

在 .axml 文件中进行调用：

```
<view class="{{cn}}">
  <line
    categories="{{categories}}"
    series="{{series}}"
    xAxis="{{xAxis}}"
    yAxis="{{yAxis}}"
    legend="{{legend}}"
    tooltip="{{tooltip}}"
  />
</view>
```

3.2.3. 布局导航

3.2.3.1. List 列表

列表将数据呈现为可以分为平铺和分组形式。使用列表以清单的形式干净，高效地显示大量或少量信息。一般来说，列表是基于文本内容的理想选择，也可以在列表中加入图标、按钮、箭头等其他元素扩展场景。

扫码体验

示例代码

```
// API-DEMO page/component/list/list.json
{
  "defaultTitle": "List",
  "usingComponents":{
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index"
  }
}
```

```
<!-- API-DEMO page/component/list/list.axml -->
<view>
    <scroll-view style="height: 100vh;" scroll-y onScrollToLower="onScrollToLower">
        <list>
            <view slot="header">
                列表头部
            </view>
            <block a:for="{{items}}">
                <list-item
                    thumb="{{item.thumb}}"
                    arrow="{{item.arrow}}"
                    align="{{item.align}}"
                    index="{{index}}"
                    onClick="onItemClick"
                    key="items-{{index}}"
                    last="{{index === (items.length - 1)}}"
                >
                    {{item.title}}
                    <view class="am-list-brief">{{item.brief}}</view>
                    <view slot="extra">
                        {{item.extra}}
                    </view>
             </list-item>
            </block>
            <view slot="footer">
                列表尾部
            </view>
        </list>
        <list>
            <view slot="header">
                列表头部
            </view>
            <block a:for="{{items2}}">
                <list-item
                    thumb="{{item.thumb}}"
                    arrow="{{item.arrow}}"
```

```
                arrow="{{item.arrow}}"
                onClick="onItemClick"
                index="items2-{{index}}"
                key="items2-{{index}}"
                last="{{index === (items2.length - 1)}}"
            >
                {{item.title}}
                <view class="am-list-brief">{{item.brief}}</view>
                <view a:if="{{item.extra}}" slot="extra">
                    {{item.extra}}
                </view>
            </list-item>
        </block>
        <view slot="footer">
            列表尾部
        </view>
    </list>
    <list>
        <view slot="header">
            列表头部
        </view>
        <block a:for="{{items3}}">
            <list-item
                thumb="{{item.thumb}}"
                arrow="{{item.arrow}}"
                index="items3-{{index}}"
                onClick="onItemClick"
                key="items3-{{index}}"
                last="{{index === (items3.length - 1)}}"
                multipleLine="{{true}}"
            >
                {{item.title}}
                <view class="am-list-brief">{{item.brief}}</view>
                <view a:if="{{item.extra}}" slot="extra">
                    {{item.extra}}
                </view>
            </list-item>
        </block>
        <view slot="footer">
            列表尾部
        </view>
    </list>
    <list>
        <view slot="header">
            列表头部
        </view>
        <block a:for="{{items4}}">
            <list-item
                thumb="{{item.thumb}}"
                arrow="{{item.arrow}}"
                onClick="onItemClick"
                index="items4-{{index}}"
                last="{{index === (items4.length - 1)}}"
                key="items4-{{index}}"
                multipleLine="{{true}}"
            >
                {{item.title}}
                <view class="am-list-brief">{{item.brief}}</view>
                <view a:if="{{item.extra}}" slot="extra">
                    {{item.extra}}
                </view>
            </list-item>
        </block>
        <view slot="footer">
            列表尾部
        </view>
    </list>
    <list>
```

```
            <view slot="header">
                小图文列表
            </view>
            <block a:for="{{itemsThumb}}">
                <list-item
                    thumb="{{item.thumb}}"
                    arrow="{{item.arrow}}"
                    onClick="onItemClick"
                    index="itemsThumb-{{index}}"
                    last="{{index === (itemsThumb.length - 1)}}"
                    key="itemsThumb-{{index}}"
                >
                    {{item.title}}
                    <view class="am-list-brief">{{item.brief}}</view>
                    <view a:if="{{item.extra}}" slot="extra">
                        {{item.extra}}
                    </view>
                </list-item>
            </block>
        </list>
        <list>
            <view slot="header">
                小图文双行列表
            </view>
            <block a:for="{{itemsThumbMultiple}}">
                <list-item
                    thumb="{{item.thumb}}"
                    arrow="{{item.arrow}}"
                    onClick="onItemClick"
                    index="items-multiple-{{index}}"
                    last="{{index === (itemsThumbMultiple.length - 1)}}"
                    key="items-multiple-{{index}}"
                    multipleLine="{{true}}"
                >
                    {{item.title}}
                    <view class="am-list-brief">{{item.brief}}</view>
                    <view a:if="{{item.extra}}" slot="extra">
                        {{item.extra}}
                    </view>
                </list-item>
            </block>
        </list>
        <list >
            <view slot="header">
                无限滚动列表
            </view>
            <block a:for="{{items5}}">
                <list-item
                    className="{{item.sticky ? 'am-list-sticky' : ''}}"
                    thumb="{{item.thumb}}"
                    arrow="{{item.arrow}}"
                    align="{{item.align}}"
                    last="{{index === (items5.length - 1)}}"
                    index="{{index}}"
                    key="items5-{{index}}"
                    onClick="onItemClick"
                    disabled="{{item.sticky}}"
                    wrap="{{true}}"
                >
                    {{item.title}}{{index}}
                    <view a:if="{{item.extra}}" slot="extra">
                        {{item.extra}}
                    </view>
                </list-item>
            </block>
            <view slot="footer">
                列表尾部
            </view>
```

```
      </list>
    </scroll-view>
</view>
```

```
// API-DEMO page/component/list/list.js
const newitems = [
  {
    thumb: 'https://gw.alipayobjects.com/zos/rmsportal/KXDIRejMrRdKlSEcLseB.png',
    title: '固定到头部',
    arrow: true,
    sticky: true,
  },
  {
    title: '标题文字不换行很长很长很长很长很长很长很长很长很长很长',
    arrow: true,
  },
  {
    title: '标题文字换行很长很长很长很长很长很长很长很长很长很长',
    arrow: true,
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '没有箭头',
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '子元素垂直对齐',
    textMode: 'wrap',
    align: 'top',
  },
  {
    title: '标题文字换行很长很长很长很长很长很长很长很长很长很长',
    arrow: true,
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '没有箭头',
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '子元素垂直对齐',
    textMode: 'wrap',
    align: 'top',
  },
  {
    title: '标题文字换行很长很长很长很长很长很长很长很长很长很长',
    arrow: true,
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '没有箭头',
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '子元素垂直对齐',
    textMode: 'wrap',
    align: 'top',
  },
  {
    title: '标题文字换行很长很长很长很长很长很长很长很长很长很长',
    arrow: true,
```

```
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '没有箭头',
    textMode: 'wrap',
  },
  {
    title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
    extra: '子元素垂直对齐',
    textMode: 'wrap',
    align: 'top',
  },
];
Page({
  data: {
    items: [
      {
        title: '单行列表',
        extra: '详细信息',
      },
    ],
    items2: [
      {
        title: '多行列表',
        arrow: true,
      },
      {
        title: '多行列表',
        arrow: 'up',
      },
      {
        title: '多行列表',
        arrow: 'down',
      },
      {
        title: '多行列表',
        arrow: 'empty',
      },
      {
        title: '多行列表',
      },
    ],
    items3: [
      {
        title: '双行列表',
        brief: '描述信息',
        arrow: true,
      },
    ],
    items4: [
      {
        title: '双行列表',
        brief: '描述信息',
        arrow: true,
      },
      {
        title: '双行列表',
        brief: '描述信息',
        arrow: true,
      },
      {
        title: '双行列表',
        brief: '描述信息',
        arrow: true,
      },
    ],
    itemsThumb: [
```

```
    {
      thumb: 'https://tfsimg.alipay.com/images/partner/T12rhxXkxcXXXXXXXX',
      title: '标题文字',
      extra: '描述文字',
      arrow: true,
    },
    {
      thumb: 'https://tfsimg.alipay.com/images/partner/T12rhxXkxcXXXXXXXX',
      title: '标题文字',
      arrow: true,
    },
    {
      thumb: 'https://tfsimg.alipay.com/images/partner/T12rhxXkxcXXXXXXXX',
      title: '标题文字',
      arrow: true,
    },
  ],
  itemsThumbMultiple: [
    {
      thumb: 'https://tfsimg.alipay.com/images/partner/T12rhxXkxcXXXXXXXX',
      title: '标题文字',
      brief: '描述信息',
    },
    {
      thumb: 'https://tfsimg.alipay.com/images/partner/T12rhxXkxcXXXXXXXX',
      title: '标题文字',
    },
    {
      thumb: 'https://tfsimg.alipay.com/images/partner/T12rhxXkxcXXXXXXXX',
      title: '标题文字',
    },
  ],
  items5: [
    {
      thumb: 'https://gw.alipayobjects.com/zos/rmsportal/KXDIRejMrRdKlSEcLseB.png',
      title: '固定到头部',
      brief: '描述信息',
      arrow: true,
      sticky: true,
    },
    {
      title: '标题文字不换行很长很长很长很长很长很长很长很长很长很长',
      arrow: true,
      align: 'middle',
    },
    {
      title: '标题文字换行很长很长很长很长很长很长很长很长很长很长',
      arrow: true,
      align: 'top',
    },
    {
      title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
      extra: '没有箭头',
      align: 'bottom',
    },
    {
      title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
      extra: '子元素垂直对齐',
      align: 'top',
    },
    {
      title: '标题文字换行很长很长很长很长很长很长很长很长很长很长',
      arrow: true,
    },
    {
      title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
      extra: '没有箭头',
```

```
      },
      {
        title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
        extra: '子元素垂直对齐',
        align: 'top',
      },
      {
        title: '标题文字换行很长很长很长很长很长很长很长很长很长很长很长很长',
        arrow: true,
      },
      {
        title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
        extra: '没有箭头',
      },
      {
        title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
        extra: '子元素垂直对齐',
        align: 'top',
      },
      {
        title: '标题文字换行很长很长很长很长很长很长很长很长很长很长很长很长',
        arrow: true,
      },
      {
        title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
        extra: '没有箭头',
      },
      {
        title: '标题文字很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长很长',
        extra: '子元素垂直对齐',
        align: 'middle',
      },
    ],
  },
  onItemClick(ev) {
    my.alert({
      content: `点击了第${ev.index}行`,
    });
  },
  onScrollToLower() {
    const { items5 } = this.data;
    const newItems = items5.concat(newitems);
    console.log(newItems.length);
    this.setData({
      items5: newItems,
    });
  },
});
```

```
/* API-DEMO page/component/list/list.acss */
.dyt-list {
  margin-top: 0;
}
.dyt-list .am-list-item-thumb {
  border-radius: 5px;
}
.dyt-list .am-list-brief {
  color: #909090;
}

.dyt-list .am-list-extra {
  color: #000;
}

.am-list-sticky {
  position: sticky;
  top: 0;
  background-color: #fff;
  z-index: 2;
}
```

属性

属性名	描述	类型	默认值
className	自定义class。	String	-
loadMore	显示加载更多 item。load：显示加载更多；over：显示加载完成无更多。	String	-
loadContent	需结合 loadMore 属性使用，用于文案展示。	Array	['加载更多...','-- 数据加载完了 --']

loadMore 使用介绍

注意：当需要使用无限循环列表时，可将 list 组件放置入到 scroll-view 可滚动视图区域 中，根据需求对 scroll-view 可滚动视图区域 添加相对应的属性，比如：

```
// API-DEMO page/component/list/list.json
{
  "defaultTitle": "List",
  "usingComponents":{
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index"
  }
}
```

```
<scroll-view style="height: 80vh;" scroll-y onScrollToLower="onScrollToLower" enable-back-to-top="true">
  <list loadMore="{{loadMore}}" loadContent="{{loadContent}}">
    <list-item>...</list-item>
  </list>
</scroll-view>
```

```
Page({
  data: {
    loadMore: '',
    loadContent: [
      '马不停蹄加载更多数据中...',
      '-- 已经到底了，加不了咯 --',
    ],
  },
  onScrollToLower() {
    // 根据实际数据加载情况设定 loadMore 的值即可，分别为 load 和 over
    this.setData({
      loadMore: 'load',
    })
  },
})
```

slots

slotName	说明
header	可选，列表头部。
footer	可选，用于渲染列表尾部。

list-item

属性	说明	类型	默认值
className	自定义的class。	String	-
thumb	缩略图，图片地址。	String	-
arrow	是否带箭头。	Boolean	false
align	子元素垂直对齐，可选 `top`，`middle`，`bottom`。	String	middle
index	列表项的唯一索引。	String	-
onClick	点击 ist-item 时调用此函数。	({index, target}) => void	-
last	是否是列表的最后一项。	Boolean	false
disabled	不可点击，且无 hover 效果。	Boolean	false
multipleLine	多行。	Boolean	false
wrap	是否换行，默认情况下，文字超长会被隐藏。	Boolean	false

slots

slotname	说明
extra	可选，用于渲染列表项右边说明。
prefix	可选，用于渲染列表左侧说明。

3.2.3.2. Tabs 横向选项卡

应用当需要展示二级以下内容时，屏幕顶部会展示一个标签栏，并提供在应用的不同部分之间快速切换的功能。标签栏在所有屏幕方向上保持相同的高度。**选项：** 根据业务场景需要可以设定2个以上选项，当选项超过屏幕宽度后可以横滑选项行查看所有内容。**新内容标记：** 选项卡上能显示未读或者新内容标记。**场景描述：** 当选项卡内容提供给用户自定义配置时提供编辑/新建入口，用户可以由此进入编辑页面进行修改。

说明：

- 通过触发 onChange 事件，设置 setData 对应的数据，从而切换页面上的 tabs 数据。
- 可使用 my.request 传数据给后端。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "tabs": "mini-antui/es/tabs/index",
    "tab-content": "mini-antui/es/tabs/tab-content/index"
  }
}
```

```
<view>
  <tabs
    tabs="{{tabs}}"
    showPlus="{{true}}"
    onTabClick="handleTabClick"
    onChange="handleTabChange"
    onPlusClick="handlePlusClick"
    activeTab="{{activeTab}}"
    swipeable="{{false}}"
  >
    <block a:for="{{tabs}}">
      <tab-content key="{{index}}" tabId="{{index}}" activeTab="{{activeTab}}" a:if="{{index === 1}}">
        <view class="tab-content" style="height: 100px;">高度为 100px {{item.title}}</view>
      </tab-content>
      <tab-content key="{{index}}" tabId="{{index}}" activeTab="{{activeTab}}" a:elif="{{index === 2}}">
        <view class="tab-content" style="height: 200px;">改变 tab-content 高度为 200px {{item.title}}</view>
      </tab-content>
      <tab-content key="{{index}}" tabId="{{index}}" activeTab="{{activeTab}}" a:else>
        <view class="tab-content">content of {{item.title}}</view>
      </tab-content>
    </block>
  </tabs>
</view>
```

```
Page({
  data: {
    tabs: [
      {
        title: '选项',
        badgeType: 'text',
        badgeText: '6',
      },
      {
        title: '选项二',
        badgeType: 'dot',
      },
      { title: '3 Tab' },
      { title: '4 Tab' },
      { title: '5 Tab' },
    ],
    activeTab: 2,
  },
  handleTabClick({ index }) {
    this.setData({
      activeTab: index,
    });
  },
  handleTabChange({ index }) {
    this.setData({
      activeTab: index,
    });
  },
  handlePlusClick() {
    my.alert({
      content: 'plus clicked',
    });
  },
});
```

```
.tab-content {
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 40rpx;
  /* 如果 swipeable="{{true}}"，需要增加 height */
  /* height: 350px; */
}
```

属性

属性名	描述	类型	默认值	必填
className	自定义 class。	String	-	否
activeCls	自定义激活 tabbar 的 class（设置字体样式和宽度）。	String	-	-
tabs	tab数据，其中包括选项标题 `title`，徽标类型 `badgeType` 分为圆点 `dot` 和文本 `text`，不设置 `badgeType` 则不显示徽标。徽标文本 `badgeText` 在 `badgeType` 为 `text` 时生效。	Array<title, badgeType, badgeText>	-	是
activeTab	当前激活 Tab 索引。	Number	-	是
showPlus	是否显示 '+'icon。	Boolean	false	否
onPlusClick	'+'icon 被点击时的回调。	() => {}	-	否
onTabClick	tab 被点击的回调。	(index: Number) => void	-	否
onChange	tab 变化时触发。	(index: Number) => void	-	否
swipeable	是否可以滑动内容切换，同时可控制高度是否自适应。	Boolean	true	否
duration	当 swipeable 为 true 时滑动动画时长，单位 ms。	Number	500(ms)	否
tabBarBackgroundColor	tabBar 背景色。	String	-	否
tabBarActiveTextColor	tabBar 激活 Tab 文字颜色。	String	-	否
tabBarInactiveTextColor	tabBar 非激活 Tab 文字颜色。	String	-	否
tabBarUnderlineColor	tabBar 下划线颜色。	String	-	否
tabBarCls	tabBar 自定义样式 class（样式中需将 background-color/color/width 排除）。	String	-	否

tab-content

视图内容

属性名	描述	类型	默认值
index	列表项的唯一索引。	String	-
tabId	tab 内容序列索引。	Number	{{index}}
activeTab	选项卡当前激活序列索引。	Number	{{activeTab}}

tab-content 高度自适应说明

tabs 组件内容区域高度是否能够自适应，需要注意 swipeable 的值：

- `swipeable='{{true}}'`：内容区域可滑动，且相对应 tab 标签卡；但**内容区域高度为固定值**，需要在 acss 文件中设定 height

值，否则高度会异常；

- `swipeable='{{false}}'`：内容区域不可滑动，此时高度表现形式有两种，且可以不需要在 acss 文件设定 height 值；

3.2.3.3. VTabs 纵向选项卡

用于让用户在不同的视图中进行切换。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "vtabs": "mini-antui/es/vtabs/index",
    "vtab-content": "mini-antui/es/vtabs/vtab-content/index"
  }
}
```

```
<view>
  <vtabs
    tabs="{{tabs}}"
    onTabClick="handleChange"
    onChange="onChange"
    activeTab="{{activeTab}}"
  >
    <block a:for="{{tabs}}">
      <vtab-content anchor="{{item.anchor}}">
        <view style="border: 1px solid #eee; height: 800px; box-sizing: border-box">
          <text>content of {{item.title}}</text>
        </view>
      </vtab-content>
    </block>
  </vtabs>
</view>
```

```
Page({
  data: {
    activeTab: 2,
    tabs: [
      { title: '选项二', anchor: 'a', badgeType: 'dot' },
      { title: '选项', anchor: 'b', badgeType: 'text', badgeText: '新' },
      { title: '不超过五字', anchor: 'c' },
      { title: '选项四', anchor: 'd' },
      { title: '选项五', anchor: 'e' },
      { title: '选项六', anchor: 'f' },
    ],
  },
  handleChange(index) {
    this.setData({
      activeTab: index,
    });
  },
  onChange(index) {
    console.log('onChange', index);
    this.setData({
      activeTab: index,
    });
  },
});
```

属性

属性名	描述	类型	默认值	必选
activeTab	当前激活 Tab 索引。	Number	-	true
tabs	tab 数据，其中包括选项标题 title，列表唯一锚点值，以及徽标类型 badgeType，分为圆点 dot 和文本 text，不设置 badgeType 则不显示徽标。徽标文本 badgeText 在 badgeType 为 text 时生效。	Array<title, anchor>	-	true
animated	是否开启动画。	Boolean	-	false
swipeable	是否可滑动切换。	Boolean	-	true
tabBarActiveBgColor	tabBar 激活状态背景色。	String	-	false
tabBarInactiveBgColor	tabBar 非激活状态背景色。	String	-	false
tabBarActiveTextColor	tabBar 激活 Tab 文字颜色。	String	-	false
tabBarInactiveTextColor	tabBar 非激活 Tab 文字颜色。	String	-	false
tabBarlineColor	tabBar 侧划线颜色。	String	-	false
onTabClick	tab 被点击的回调。	(index: Number) => void	-	false
onChange	vtab-content 变化时触发。	(index: Number) => void	-	false

vtab-content

视图内容

属性名	描述	类型	默认值	必选
anchor	列表唯一锚点值。	String	-	true

3.2.3.4. Card 卡片

卡片。

扫码体验

示例代码

```
// API-DEMO page/component/card.json
{
  "defaultTitle": "Card",
  "usingComponents":{
    "card": "mini-antui/es/card/index"
  }
}
```

```
<!-- API-DEMO page/component/card.axml -->
<view class="container">
  <card
    title="卡片标题1"
    subTitle="副标题非必填1"
    onClick="onCardClick"
    info="点击了第一个card"
  />
  <view style="margin-top: 10px;" />
  <card
    thumb="{{thumb}}"
    title="卡片标题2"
    subTitle="副标题非必填2"
    onClick="onCardClick"
    info="点击了第二个card"
  />
  <view style="margin-top: 10px;" />
  <card
    thumb="{{thumb}}"
    title="卡片标题3"
    subTitle="副标题非必填3"
    onClick="onCardClick"
    footer="描述文字"
    footerImg="{{footerImg}}"
    info="点击了第三个card"
  />
```

```
// API-DEMO page/component/card.js
Page({
  data: {
    thumb: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
    footerImg: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
  },
  onCardClick: function(ev) {
    my.showToast({
      content: ev.info,
    });
  }
});
```

属性

属性名	描述	类型	默认值	必选
thumb	Card 缩略图地址。	String	-	false
title	Card 标题。	String	-	true
subTitle	Card 副标题。	String	-	false
footer	footer 文字。	String	-	false
footerImg	footer 图片地址。	String	-	false
onClick	Card 点击的回调。	(info: Object) => void	-	false
info	用于点击卡片时往外传递数据。	String	-	false

3.2.3.5. Grid 宫格

当业务新上线时，可以通过蒙板引导的形式给与用户强制的信息传达。全蒙层引导是一种很重的操作引导，除非页面功能的重大变动，否则不建议使用。小的变化可以用 Tips 引导。

用法

宫格列数：根据需求可以选择 2、3、4、5 列宫格。

宫格行数：数目不受限制。

宫格样式：圆形底、方形底。

文字限制：标题字数不超过5个中文字符；描述文字不超过限制区域，超过自动截断无法继续输入。

是否有虚线分割：页面比较复杂的建议不要线，页面比较单一的建议要线。

扫码体验

示例代码

```
// API-DEMO page/component/grid/grid.json
{
  "defaultTitle": "Grid",
  "usingComponents":{
    "grid": "mini-antui/es/grid/index"
  }
}
```

```
<!-- API-DEMO page/component/grid/grid.axml -->
<view style="margin-top: 10px;" />
<grid onGridItemClick="onItemClick" columnNum="{{3}}" list="{{list3}}" />
<view style="margin-top: 10px;" />
<grid onGridItemClick="onItemClick" columnNum="{{3}}" list="{{list1}}" />
<view style="margin-top: 10px;" />
<grid onGridItemClick="onItemClick" columnNum="{{3}}" list="{{list3}}" hasLine="{{false}}" />
```

```
// API-DEMO page/component/grid/grid.js
Page({
  data: {
    list1: [
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
      },
    ],
    list3: [
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
```

```
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
      {
        icon: 'https://gw.alipayobjects.com/zos/rmsportal/VBqNBOiGYkCjqocXjdUj.png',
        text: '标题文字',
        desc: '描述信息',
      },
    ],
  },
  onItemClick(ev) {
    my.alert({
      content: ev.detail.index,
    });
  },
});
```

```
/*  API-DEMO page/component/grid/grid.css */
.am-grid {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  background-color: white;
}

.am-grid-4 {
  padding-top: 7px;
  padding-bottom: 7px;
}

.am-grid-5 {
  padding-top: 6px;
  padding-bottom: 7px;
}

.am-grid-item {
  text-align: center;
  position: relative;
}

.am-grid-3 .am-grid-border {
  position: absolute;
  left: 0;
  bottom: 0;
  width: 100%;
  height: 100%;
  border-bottom: 1rpx solid #eee;
  border-right: 1rpx solid #eee;
  box-sizing: border-box;
}

.am-grid-3 .am-grid-right {
  border-right: none;
}

.am-grid-3 .am-grid-bottom {
```

```
  border-bottom: none;
}

.am-grid-3 .am-grid-top {
  height: calc(100% - 15px);
}

.am-grid-item-wrapper {
  position: absolute;
  left: 0;
  bottom: 0;
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

.am-grid-2 .am-grid-item-wrapper {
  flex-direction: row;
  justify-content: flex-start;
}

.am-grid-icon-container {
  display: flex;
  justify-content: center;
  align-items: center;
}

.am-grid-icon {
  flex: 1;
  width: 36px;
  height: 36px;
}

.am-grid-text-wrapper {
  display: flex;
  flex-direction: column;
  align-items: flex-start;
}

.am-grid-3 .am-grid-text-wrapper {
  align-items: center;
}

.am-grid-3 .am-grid-text {
  height: 20px;
  line-height: 20px;
}

.am-grid-3 .am-grid-desc {
  height: 17px;
  line-height: 17px;
}

.am-grid-text {
  color: #333;
  font-size: 14px;
  line-height: 1;
  margin-top: 14px;
}

.am-grid-desc {
  color: #999;
  font-size: 12px;
}
```

```
.am-grid-2 .am-grid-item.has-desc {
  padding-top: 70px;
}

.am-grid-3 .am-grid-item {
  padding-top: 125px;
}

.am-grid-3.am-grid-no-line {
  padding: 8px 16px 0 16px;
}

.am-grid-no-line .am-grid-item {
  padding-top: 110px;
}

.am-grid-2 .am-grid-item {
  padding-top: 64px;
}


.am-grid-4 .am-grid-item {
  padding-top: 68px;
}

.am-grid-4.circular .am-grid-item {
  padding-top: 82px;
}

.am-grid-3 {
  padding: 0 16px;
}

.am-grid-3 .am-grid-text {
  margin-top: 12px;
}

.am-grid-no-line .am-grid-border {
  border-right: none;
  border-bottom: none;
}

.am-grid-4.circular {
  padding-top: 3px;
}

.am-grid-4.circular .am-grid-icon-container {
  margin-top: 13px;
  padding: 8px;
  border-radius: 50%;
  background-color: #D8D8D8;
}

.am-grid-2 .am-grid-icon-container {
  margin-left: 16px;
}

.am-grid-2 .am-grid-icon {
  width: 28px;
  height: 28px;
}

.am-grid-4 .am-grid-icon-container {
  margin-top: 7px;
}

.am-grid-4 .am-grid-icon {
```

```
  height: 28px;
  width: 28px;
}

.am-grid-4.circular .am-grid-icon {
  width: 26px;
  height: 26px;
  flex: 1;
}

.am-grid-4.circular .am-grid-text {
  height: 16px;
  line-height: 16px;
}

.am-grid-5 .am-grid-item {
  padding-top: 75px;
}

.am-grid-5 .am-grid-icon {
  border-radius: 50%;
  width: 43px;
  height: 43px;
}

.am-grid-2 .am-grid-text {
  margin-top: 0;
  margin-left: 12px;
  height: 24px;
  line-height: 24px;
  font-size: 17px;
}

.am-grid-2 .am-grid-desc {
  margin-left: 12px;
  height: 16px;
  line-height: 16px;
}

.am-grid-4 .am-grid-text {
  font-size: 13px;
  height: 13px;
  line-height: 13px;
  margin-top: 7px;
}
.am-grid-5 .am-grid-text {
  font-size: 12px;
  margin-top: 7px;
}
```

属性

属性名	描述	类型	默认值	必选
list	宫格数据。	Array<icon, text>	[]	是
onGridItemClick	点击宫格项回调。	(index: Number) => void	-	否
columnNum	每行显示几列。	2、3、4、5	3	否
circular	是否圆角。	Boolean	false	否
hasLine	是否有边框。	Boolean	true	否

3.2.3.6. Steps 步骤条

步进样式能够很好的可视化呈现给用户当前的服务进度，并且能够对整个流程有更加清晰的了解。

类型

横向步进器

场景描述：适用于步骤描述比较简单的场景（例如步骤不需要描述，或者步骤通过2个字的词语可以描述清楚）。

规则：

1. 标题不允许超过一行，原则上不允许超过自己进度范围以内区域（范围见标注）。
2. 描述可以根据业务需要选择性添加，文字长度标准同标题

纵向步进器

场景描述：适用于步骤描述比较详细的场景（例如步骤描述超过6个中文字，或者有短句样式的内容描述）。

规则：

1. 标题和描述不允许超过2行，超过后自动截断。
2. 描述可以根据业务需要选择性添加。

用法

步骤数量：考虑到空间范围，步骤不能超过4步，至少2步。

失败标记：失败的步骤图标和文字会对应变为红色。

扫码体验

示例代码

```
{
  "usingComponents": {
    "steps": "mini-antui/es/steps/index"
  }
}
```

```
<steps
  activeIndex="{{activeIndex}}"
  items="{{items}}"
></steps>
```

```
Page({
  data: {
    activeIndex: 1,
    items: [{
      title: '步骤1',
      description: '这是步骤1',
    }, {
      title: '步骤2',
      description: '这是步骤2',
    }, {
      title: '步骤3',
      description: '这是步骤3',
    }]
  }
});
```

属性

属性名	描述	类型	默认值	必填
className	最外层覆盖样式。	String	-	否
activeIndex	当前激活步骤。	Number	1	是
failIndex	当前失败步骤（只在 vertical 模式下生效）。	Number	0	否
direction	显示方向，可选值：vertical、horizontal。	String	horizontal	否
size	统一的 icon 大小，单位为 px。	Number	0	否
items	步骤详情（详情请见下方表格）。	Array[{title, description, icon, activeIcon, size}]	[]	是

items

属性名	描述	类型	默认值	必填
items.title	步骤详情标题。	String	-	是
items.description	步骤详情描述。	String	-	是
items.icon	尚未到达步骤的 icon（只在 vertical 模式下生效）。	String	-	是
items.activeIcon	已到达步骤的 icon（只在 vertical 模式下生效）。	String	-	是
items.size	已到达步骤 icon 的图标大小，单位为 px（只在 vertical 模式下生效）。	Number	-	是

3.2.3.7. Footer 页脚

当页面内容加载完成时，需要在背景上通过文字形式给与用户提示。

用法

链接位于页脚描述上方

底部链接：

1. 整体居中对齐，允许1个或者2个链接。
2. 2个链接时左链接右对齐，右链接左对齐。
3. 链接文字不允许超过一行，超过自动截断。

页脚描述：

1. 文字最多一行，超过范围截断。
2. 居中对齐。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents":{
    "footer": "mini-antui/es/footer/index"
  }
}
```

```
<view>
  <footer
    copyright="{{copyright}}"
    links="{{links}}" />
</view>
```

```
Page({
 data: {
    copyright: '© 2004-2017 Alipay.com. All rights reserved.',
    links: [
      { text: '底部链接', url: '../../list/demo/index' },
      { text: '底部链接', url: '../../card/demo/index' },
    ],
 },
});
```

属性

属性名	描述	类型	默认值	必填
copyright	版权信息。	String	-	否
links	页脚链接。	Array<text, url>	-	否

3.2.3.8. Flex Flex布局

CSS flex布局的封装。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "flex": "mini-antui/es/flex/index",
    "flex-item": "mini-antui/es/flex/flex-item/index"
  }
}
```

```
<view class="flex-container">
  <view class="sub-title">Basic</view>
  <flex>
    <flex-item><view class="placeholder">Block</view></flex-item>
    <flex-item><view class="placeholder">Block</view></flex-item>
  </flex>
  <view style="height: 20px;" />
  <flex>
    <flex-item><view class="placeholder">Block</view></flex-item>
    <flex-item><view class="placeholder">Block</view></flex-item>
    <flex-item><view class="placeholder">Block</view></flex-item>
  </flex>
  <view style="height: 20px;" />
  <flex>
    <flex-item><view class="placeholder">Block</view></flex-item>
    <flex-item><view class="placeholder">Block</view></flex-item>
    <flex-item><view class="placeholder">Block</view></flex-item>
    <flex-item><view class="placeholder">Block</view></flex-item>
  </flex>
  <view className="sub-title">Wrap</view>
  <flex wrap="wrap">
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
  </flex>
  <view className="sub-title">Align</view>
  <flex justify="center">
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
  </flex>
  <flex justify="end">
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
  </flex>
  <flex justify="between">
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline">Block</view>
  </flex>
  <flex align="start">
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline small">Block</view>
    <view class="placeholder inline">Block</view>
  </flex>
  <flex align="end">
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline small">Block</view>
    <view class="placeholder inline">Block</view>
  </flex>
  <flex align="baseline">
    <view class="placeholder inline">Block</view>
    <view class="placeholder inline small">Block</view>
    <view class="placeholder inline">Block</view>
  </flex>
</view>
```

```
.flex-container {
  padding: 10px;
}

.sub-title {
  color: #888;
  font-size: 14px;
  padding: 30px 0 18px 0;
}

.placeholder {
  background-color: #ebebef;
  color: #bbb;
  text-align: center;
  height: 30px;
  line-height: 30px;
  width: 100%;
}

.placeholder.inline {
  width: 80px;
  margin: 9px 9px 9px 0;
}

.placeholder.small {
  height: 20px;
  line-height: 20px
}
```

```
Page({});
```

属性

属性名	描述	类型	默认值	必选
direction	项目定位方向，值可以为 row,row-reverse，column，olumn-reverse。	String	row	false
wrap	子元素的换行方式，可选 nowrap，wrap，rap-reverse。	String	nowrap	false
justify	子元素在主轴上的对齐方式，可选 start，end，center，between，around。	String	start	false
align	子元素在交叉轴上的对齐方式，可选 start，center，end，baseline，stretch。	String	center	false
alignContent	有多根轴线时的对齐方式，可选 start，end，center，between，around，stretch。	String	stretch	false

flex-item

flex-item 组件默认加上了样式 flex:1,保证所有 item 平均分宽度，flex 容器的 children 不一定是 flex-item。

3.2.3.9. Pagination 分页

分页。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "pagination": "mini-antui/es/pagination/index"
  }
}
```

```
<view>
  <view class="demo-title">基础用法</view>
  <pagination total="{{20}}" current="{{1}}"/>
  <view class="demo-title">箭头按钮</view>
  <pagination mode="icon" total="{{20}}" current="{{10}}"/>
  <view class="demo-title">简单模式</view>
  <pagination simple total="{{20}}" current="{{1}}"/>
  <view class="demo-title">按钮禁用</view>
  <pagination total="{{20}}" current="{{1}}" disabled/>
  <view class="demo-title">自定义按钮文案</view>
  <pagination arrow prevText="上一篇" nextText="下一篇" total="{{20}}" current="{{1}}"/>
</view>
```

```
Page({})
```

属性

属性名	描述	类型	默认值
mode	按钮的形态可选类型：text、icon。	String	text
total	总页数。	Number	0
current	当前页数。	Number	0
simple	是否隐藏数值。	Boolean	false
disabled	禁用状态。	Boolean	false
prevText	前翻分页按钮文案。	String	上一页
nextText	后翻分页按钮文案。	String	下一页
btnClass	分页按钮样式，限于文字类型按钮。	String	-
onChange	翻页回调函数。	(index: Number) => void	-

说明：prevText 和 nextText 当且仅当 mode 为 text 时生效。

3.2.3.10. Collapse 折叠面板

可以折叠 / 展开的内容区域。

- 对复杂区域进行分组和隐藏，保持页面的整洁。
- **手风琴模式** 是一种特殊的折叠面板，只允许单个内容区域展开。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "collapse": "mini-antui/es/collapse/index",
    "collapse-item": "mini-antui/es/collapse/collapse-item/index"
  }
}
```

```
<view>
  <view class="demo-title">基础用法</view>
  <collapse
    className="demo-collapse"
    collapseKey="collapse1"
    activeKey="{{['item-11', 'item-13']}}"
    onChange="onChange"
  >
    <collapse-item header="标题1" itemKey="item-11" collapseKey="collapse1">
      <view class="item-content content1">
        <view>内容区域</view>
      </view>
    </collapse-item>
    <collapse-item header="标题2" itemKey="item-12" collapseKey="collapse1">
      <view class="item-content content2">
        <view>内容区域</view>
      </view>
    </collapse-item>
    <collapse-item header="标题3" itemKey="item-13" collapseKey="collapse1">
      <view class="item-content content3">
        <view>内容区域</view>
      </view>
    </collapse-item>
  </collapse>
  <view class="demo-title">手风琴模式</view>
  <collapse
    className="demo-collapse"
    collapseKey="collapse2"
    activeKey="{{['item-21', 'item-23']}}"
    onChange="onChange"
    accordion="{{true}}"
  >
    <collapse-item header="标题1" itemKey="item-21" collapseKey="collapse2">
      <view class="item-content content1">
        <view>内容区域</view>
      </view>
    </collapse-item>
    <collapse-item header="标题2" itemKey="item-22" collapseKey="collapse2">
      <view class="item-content content2">
        <view>内容区域</view>
      </view>
    </collapse-item>
    <collapse-item header="标题3" itemKey="item-23" collapseKey="collapse2">
      <view class="item-content content3">
        <view>内容区域</view>
      </view>
    </collapse-item>
  </collapse>
</view>
```

```
.item-content {
  padding: 14px 16px;
  font-size: 17px;
  color: #333;
  line-height: 24px;
}

.content1 {
  height: 200px;
}

.content2 {
  height: 50px;
}

.content3 {
  height: 100px;
}

.demo-title {
  padding: 14px 16px;
  color: #999;
}

.demo-collapse {
  border-bottom: 1px solid #eee;
}
```

```
Page({});
```

属性

属性名	描述	类型	默认值
activeKey	当前激活 tab 面板的 key。	Array / String	默认无，accordion模式下默认第一个元素
onChange	切换面板的回调。	(activeKeys: Array): void	-
accordion	手风琴模式。	Boolean	false
collapseKey	唯一标示 collapse 和对应的 collapse-item。	String	false

collapse-item

属性名	描述	类型	默认值
itemKey	对应 activeKey。	String	组件唯一标识
header	面板头内容。	String	-
collapseKey	唯一标示 collapse 和对应的 collapse-item。	String	false

当 Page 中存在多个 collapse 组件时，collapse 和对应的 collapse-item 的 collapseKey 属性为必选值并且必须相等，当 Page 中只有一个 collapse 组件时，collapseKey 不需要提供。

3.2.4. 操作浮层

3.2.4.1. Popover 气泡

气泡。可通过设置 Popover-item 宽高改变气泡大小，不支持文字自适应宽高。

说明：设置 popover 位于特定元素的正下方，可以把该元素放在 popover 内并且设置 position 为 bottom 。

扫码体验

示例代码

```
// API-DEMO page/component/popover.json
{
  "defaultTitle": "Popover",
  "usingComponents": {
    "popover": "mini-antui/es/popover/index",
    "popover-item": "mini-antui/es/popover/popover-item/index"
  }
}
```

```
<!-- API-DEMO page/component/popover/.axml-->
<view class="demo-popover">
  <popover
    position="{{position}}"
    show="{{show}}"
    showMask="{{showMask}}"
    onMaskClick="onMaskClick"
  >
    <view class="demo-popover-btn" onTap="onShowPopoverTap">点击{{show ? '隐藏' : '显示'}}</view>
    <view slot="items">
      <popover-item onItemClick="itemTap1">
        <text>{{position}}</text>
      </popover-item>
      <popover-item onItemClick="itemTap2">
        <text>line2</text>
      </popover-item>
    </view>
  </popover>
</view>
<view class="demo-popover-test-btns">
  <button class="demo-popover-test-btn" onTap="onNextPositionTap">下个位置</button>
  <button class="demo-popover-test-btn" onTap="onMaskChangeTap">蒙层{{showMask ? '隐藏' : '显示'}}</button>
</view>
```

```
// API-DEMO page/component/popover.js
const position = ['top', 'topRight', 'rightTop', 'right', 'rightBottom', 'bottomRight', 'bottom', 'bottomLeft', 'le
ftBottom', 'left', 'leftTop', 'topLeft'];
Page({
  data: {
    position: position[0],
    show: false,
    showMask: true,
  },
  onShowPopoverTap() {
    this.setData({
      show: !this.data.show,
    });
  },
  onNextPositionTap() {
    let index = position.indexOf(this.data.position);
    index = index >= position.length - 1 ? 0 : index + 1;
    this.setData({
      show: true,
      position: position[index],
    });
  },
  onMaskChangeTap() {
    this.setData({
      showMask: !this.data.showMask,
    });
  },
  onMaskClick() {
    this.setData({
      show: false,
    });
  },
  itemTap1() {
    my.alert({
      content: '点击1',
    });
  },
  itemTap2() {
    my.alert({
      content: '点击2',
    });
  },
});
```

```
/* API-DEMO page/component/popover.css */
.demo-popover {
  display: flex;
  align-items: center;
  justify-content: center;
  width: 100%;
  height: 400px;
}
.demo-popover-btn {
  width: 100px;
  height: 100px;
  line-height: 100px;
  text-align: center;
  background-color: #fff;
  border: 1px solid #dddddd;
  border-radius: 2px;
}
.demo-popover-test-btns {
  display: flex;
  justify-content: space-around;
}
.demo-popover-test-btn {
  width: 45%;
}
```

属性

属性名	描述	类型	默认值	必填
className	最外层覆盖样式。	String	-	否
show	气泡是否展示。	Boolean	false	是
showMask	蒙层是否展示。	Boolean	true	否
position	气泡位置可选值：`top` 、`topRight` 、`topLeft` 、`bottom` 、`bottomLeft` 、`bottomRight` 、`right` 、`rightTop` 、`rightBottom` 、`left` 、`leftBottom` 、`leftTop` 。	String	bottomRight	否

popover-item

属性名	描述	类型
className	单项样式。	String
onItemClick	单项点击事件。	() => void

3.2.4.2. Filter 筛选

用作标签卡筛选。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "filter": "mini-antui/es/filter/index",
    "filter-item": "mini-antui/es/filter/filter-item/index"
  }
}
```

```
<filter show="{{show}}" max="{{5}}" onChange="handleCallBack" onMaskTap="toggleFilter">
  <block a:for="{{items}}">
    <filter-item value="{{item.value}}" id="{{item.id}}" selected="{{item.selected}}"/>
  </block>
</filter>
```

```
Page({
  data: {
    show: true,
    items: [
      { id: 1, value: '衣服', selected: true },
      { id: 1, value: '橱柜' },
      { id: 1, value: '衣架' },
      { id: 3, value: '数码产品' },
      { id: 4, value: '防盗门' },
      { id: 5, value: '椅子' },
      { id: 7, value: '显示器' },
      { id: 6, value: '某最新款电子产品' },
      { id: 8, value: '某某某某某牌电视游戏底座' },
    ]
  },
  handleCallBack(data) {
    my.alert({
      content: data
    });
  },
  toggleFilter() {
    this.setData({
      show: !this.data.show,
    });
  }
});
```

属性

属性名	描述	类型	默认值	必填
show	是否显示 可选值 show、hide。	Boolean	hide	否

max	可选数量最大值，1为单选。	Number	10000	否
onChange	多选时提交选中回调。	(e: Object) => void	-	否
onMaskTap	点击遮罩层时触发，可用于关闭 filter。	() => void	-	否

filter-item

属性名	描述	类型	默认值	必填
className	自定义样式。	String	-	否
value	值。	String	-	是
id	自定义标识符。	String	-	否
selected	默认选中。	Boolean	false	否
onChange	单选时提交选中回调。	(e: Object) => void	-	否

常见问题

Filter 组件是否能过滤，如果 value 里面有一串文字，过滤中间部分？

不支持。目前没有相关操作。

为何 IDE 上可以使用 ES6 的 filter 和 findIndex，真机调试和预览不支持？

filter 暂不支持真机调试和预览，findIndex 支持真机调试和预览。

3.2.4.3. Modal 对话框

当应用中需要比较明显的对用户当前的操作行为进行警示或提醒时，可以使用对话框。用户需要针对对话框进行操作后方可结束。

说明：Modal 组件为官方组件，不支持自定义。可在Modal 组件源代码的 css 中设置弹出的框的背景色为透明。

扫码体验

示例代码

```
// API-DEMO page/component/modal.json
{
  "defaultTitle": "Modal",
  "usingComponents":{
    "modal": "mini-antui/es/modal/index"
  }
}
```

```
<!-- API-DEMO page/component/modal.axml -->
<view>
    <view style="margin-top: 10px;" />
    <button onTap="openModal21" style="margin: 0 10px;">带图弹窗</button>
```

```
    <view style="margin-top: 10px;" />
    <button onTap="openModal22" style="margin: 0 10px;">带图弹窗-小图</button>
    <view style="margin-top: 10px;" />
    <button onTap="openModal2" style="margin: 0 10px;">带图弹窗-大图</button>
    <view style="margin-top: 10px;" />
    <button onTap="openModal" style="margin: 0 10px;">通用modal</button>
    <view style="margin-top: 10px;" />
    <button onTap="openModal3" style="margin: 0 10px;">运营活动弹窗-大</button>
    <view style="margin-top: 10px;" />
    <button onTap="openModal4" style="margin: 0 10px;">运营活动弹窗-小</button>
    <modal
        show="{{modalOpened21}}"
        onModalClick="onModalClick21"
        onModalClose="onModalClose21"
        topImage="https://gw.alipayobjects.com/zos/rmsportal/yFeFExbGpDxvDYnKHcrs.png"
    >
        <view slot="header">标题单行</view>
        说明当前状态、提示用户解决方案，最好不要超过两行。
        <view slot="footer">确定</view>
    </modal>
    <modal
        show="{{modalOpened2}}"
        onModalClick="onModalClick2"
        onModalClose="onModalClose2"
        topImage="https://gw.alipayobjects.com/zos/rmsportal/yFeFExbGpDxvDYnKHcrs.png"
        topImageSize="lg"
        closeType="1"
    >
        <view slot="header">标题单行</view>
        说明当前状态、提示用户解决方案，最好不要超过两行。
        <view slot="footer">确定</view>
    </modal>
    <modal
        show="{{modalOpened22}}"
        onModalClick="onModalClick22"
        onModalClose="onModalClose22"
        topImage="https://gw.alipayobjects.com/zos/rmsportal/yFeFExbGpDxvDYnKHcrs.png"
        topImageSize="sm"
    >
        <view slot="header">标题单行</view>
        说明当前状态、提示用户解决方案，最好不要超过两行。
        <view slot="footer">确定</view>
    </modal>
    <modal
        show="{{modalOpened}}"
        onModalClick="onModalClick"
        onModalClose="onModalClose"
    >
        <view style="margin: 20px 0 10px 0;">让员工用阿里云扫一扫，添加为店员</view>
        <image mode="widthFix" style="width: 100%;" src="https://tfsimg.alipay.com/images/mobilecodec/TB1JjWmXkyEDu
Njme6tXXXIKXXa" />
        二维码每分钟自动更新
        <view slot="footer">确定</view>
    </modal>
    <modal
        show="{{modalOpened3}}"
        onModalClose="onModalClick3"
        advice="{{true}}"
        >
        <view style="display: flex; height: 465px; width: 319px;">
            <image
                style="height: 465px;width: 319px;"
                src="https://gw.alipayobjects.com/zos/rmsportal/vJzAWQLgeDLGOMKiIgZt.png"
            />
        </view>
    </modal>
    <modal
        show="{{modalOpened4}}"
```

```
        onModalClose="onModalClick4"
        advice="{{true}}"
        >
        <view style="display: flex; width: 270px; height: 350px;">
            <image
                style="width: 270px;height: 350px;"
                src="https://gw.alipayobjects.com/zos/rmsportal/vJzAWQLgeDLGOMKiIgZt.png"
            />
        </view>
    </modal>
</view>
```

```
// API-DEMO page/component/modal.js
Page({
  data: {
    modalOpened: false,
    modalOpened2: false,
    modalOpened21: false,
    modalOpened22: false,
    modalOpened3: false,
    modalOpened4: false,
  },
  openModal() {
    this.setData({
      modalOpened: true,
    });
  },
  onModalClick() {
    this.setData({
      modalOpened: false,
    });
  },
  onModalClose() {
    this.setData({
      modalOpened: false,
    });
  },
  openModal2() {
    this.setData({
      modalOpened2: true,
    });
  },
  onModalClick2() {
    this.setData({
      modalOpened2: false,
    });
  },
  onModalClose2() {
    this.setData({
      modalOpened2: false,
    });
  },
  openModal21() {
    this.setData({
      modalOpened21: true,
    });
  },
  onModalClick21() {
    this.setData({
      modalOpened21: false,
    });
  },
  onModalClose21() {
    this.setData({
      modalOpened21: false,
    });
  },
```

```
  openModal22() {
    this.setData({
      modalOpened22: true,
    });
  },
  onModalClick22() {
    this.setData({
      modalOpened22: false,
    });
  },
  onModalClose22() {
    this.setData({
      modalOpened22: false,
    });
  },
  openModal3() {
    this.setData({
      modalOpened3: true,
    });
  },
  onModalClick3() {
    this.setData({
      modalOpened3: false,
    });
  },
  openModal4() {
    this.setData({
      modalOpened4: true,
    });
  },
  onModalClick4() {
    this.setData({
      modalOpened4: false,
    });
  },
});
```

属性

属性名	描述	类型	默认值
className	自定义 class。	String	-
show	是否展示 modal。	Boolean	false
showClose	是否渲染 关闭。	Boolean	true
closeType	关闭图标类型 0：灰色图标 1：白色图标。	String	0
onModalClick	点击 footer 部分的回调。	() => void	-
onModalClose	点击 关闭 的回调, showClose 为 false 时无需设置。	() => void	-
topImage	顶部图片。	String	-
topImageSize	顶部图片规则，可选值：`lg`(带图弹框-大图)、`md`(带图弹框)、`sm`(带图弹框-小图)。	String	md
advice	是否是运营类弹窗。	Boolean	false
disableScroll	modal 展示时是否禁止页面滚动（**以真机效果为准**）	Boolean	true

slots

slotName	说明	必填
header	modal 头部。	否
footer	modal 尾部。	否

3.2.4.4. Popup 弹出菜单

弹出菜单。

扫码体验

示例代码

```
// API-DEMO page/component/popup/popup.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "popup": "mini-antui/es/popup/index"
  }
}
```

```
<!-- API-DEMO page/component/popup/popup.axml -->
<view>
  <view class="btn-container">
    <button onTap="onTopBtnTap">上</button>
    <button onTap="onButtomBtnTap">下</button>
    <button onTap="onRightBtnTap">右 无mask</button>
    <button onTap="onLeftBtnTap">左 无动画</button>
  </view>
  <popup show="{{showLeft}}" animation="{{false}}" position="left" onClose="onPopupClose">
    <view class="box left">hello world</view>
  </popup>
  <popup show="{{showRight}}" position="right" mask="{{false}}" onClose="onPopupClose">
    <view class="box right" style="display: flex; flex-direction: column;">
      <view>hello world</view>
      <view style="margin-top: 20px;">
        <button onTap="onPopupClose" style="width: 100px;">关闭</button>
      </view>
    </view>
  </popup>
  <popup show="{{showTop}}" position="top" onClose="onPopupClose">
    <view class="box top">hello world</view>
  </popup>
  <popup show="{{showBottom}}" position="bottom" onClose="onPopupClose">
    <view class="box bottom">hello world</view>
  </popup>
</view>
```

```
// API-DEMO page/component/popup/popup.js
Page({
  data: {
    showLeft: false,
    showRight: false,
    showTop: false,
    showBottom: false,
  },
  onTopBtnTap() {
    this.setData({
      showTop: true,
    });
  },
  onRightBtnTap() {
    this.setData({
      showRight: true,
    });
  },
  onLeftBtnTap() {
    this.setData({
      showLeft: true,
    });
  },
  onButtomBtnTap() {
    this.setData({
      showBottom: true,
    });
  },
  onPopupClose() {
    this.setData({
      showLeft: false,
      showRight: false,
      showTop: false,
      showBottom: false,
    });
  },
});
```

```
/* API-DEMO page/component/popup/popup.acss */
.btn-container {
  width: 100%;
  height: 100vh;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

.btn-container button {
  width: 250px;
}

.box.top, .box.bottom {
  height: 200px;
}

.box.left, .box.right {
  width: 200px;
  height: 100%;
}

.box {
  background-color: #fff;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

属性

属性名	描述	类型	默认值	必填
className	自定义 class。	String	-	否
show	是否显示菜单。	Boolean	false	否
animation	是否开启动画。	Boolean	true	否
mask	是否显示 mask，不显示时点击外部不会触发 onClose。	Boolean	true	是
position	控制从什么方向弹出菜单，bottom 表示底部，left 表示左侧，top 表示顶部，right 表示右侧。	String	bottom	否
disableScroll	展示 mask 时是否禁止页面滚动。	Boolean	true	否
zIndex	定义 popup 的层级。	Number	0	否

slots

可以在 popup 组件中定义要展示部分，具体可参看示例代码。

3.2.5. 结果类

3.2.5.1. PageResult 异常页

异常页面。

扫码体验

示例代码

```
// API-DEMO page/component/page-result/page-result.json
{
  "defaultTitle": "异常反馈",
  "usingComponents": {
    "page-result": "mini-antui/es/page-result/index"
  }
}
```

```
<!-- API-DEMO page/component/page-result/page-result.axml -->
<page-result
  type="network"
  title="网络不给力"
  brief="世界上最遥远的距离莫过于此"
/>
<page-result
  type="network"
  title="网络不给力"
  brief="世界上最遥远的距离莫过于此"
>
  <view class="am-page-result-btns">
    <view onTap="backHome">回到首页</view>
    <view>示例按钮</view>
  </view>
</page-result>
```

```
// API-DEMO page/component/page-result/page-result.js
Page({
  backHome() {
    my.navigateBack();
  }
});
```

```
.am-page-result {
  display: flex;
  flex-direction: column;
}
.am-page-result-btns {
  flex: 1;
  display: flex;
  flex-direction: column;
  justify-content: flex-end;
  align-content: center;
  padding-bottom: 100rpx;
}
.am-page-result-btns > view {
  color: #108EE9;
  font-size: 40rpx;
  margin-top: 52rpx;
  text-align: center;
}
```

属性

属性名	描述	类型	默认值	必填
type	异常页面类型，网络异常 network，服务繁忙 busy，服务异常 error，空状态 empty，用户注销 logoff。	String	network	否
local	是否是局部异常内容。	Boolean	false	否
title	错误提示标题。	String	-	否
brief	错误提示简要。	String	-	否

3.2.5.2. Message 结果页

结果页。

扫码体验

示例代码

```
// API-DEMO page/component/message/message.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "message": "mini-antui/es/message/index"
  }
}
```

```
<!-- API-DEMO page/component/message/message.axml -->
<view>
  <message
    title="{{title}}"
    subTitle="{{subTitle}}"
    type="success"
    mainButton="{{messageButton.mainButton}}"
    subButton="{{messageButton.subButton}}"
    onTapMain="goBack">
  </message>
</view>
```

```
// API-DEMO page/component/message/message.js
Page({
  data: {
    title: "操作成功",
    subTitle: "内容详情可折行，建议不超过两行",
    messageButton: {
      mainButton: {
        buttonText: "主要操作"
      },
      subButton: {
        buttonText: "辅助操作"
      }
    }
  },
  goBack() {
    my.navigateBack();
  }
});
```

属性

属性名	描述	类型	默认值	必填
className	自定义的 class。	String	-	否
type	有 success、fail、info、warn、waiting 五种状态类型，默认为 success。	String	success	否
title	主标题。	String	-	是
subTitle	副标题。	String	-	否
mainButton	主按钮的文本和可用性相关。	Object<buttonText, disabled>	-	否
subButton	副按钮的文本和可用性相关。	Object<buttonText, disabled>	-	否
onTapMain	主按钮的点击函数。	() => {}	-	否
onTapSub	副按钮的点击函数。	() => {}	-	否

3.2.6. 提示引导

3.2.6.1. Tips 引导

Tips引导是特定应用场景下系统针对用户的一种功能引导方式，例如用户第一次登录后、或者某个新功能上线后的提示等。分 `tips-dialog` （对话型）和 `tips-plain` （简单型）两种类型。

说明：

- 仅用于 UI 展示没有对应的业务逻辑功能。
- 收藏引导 tips 不针对小程序，只针对用户，用户见到第一次引导收藏 tips 之后，所有小程序都将对该用户隐藏引导收藏的 tips。

扫码体验

示例代码

```
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "tips-dialog": "mini-antui/es/tips/tips-dialog/index",
    "tips-plain": "mini-antui/es/tips/tips-plain/index"
  }
}
```

```
<view>
  <tips-dialog
    show="{{showDialog}}"
    className="dialog"
    type="dialog"
  >
    <view class="content" slot="content">
      <view>hello,</view>
      <view>欢迎使用小程序扩展组件库mini-antui</view>
    </view>
    <view slot="operation" class="opt-button" onTap="onDialogTap">知道了</view>
  </tips-dialog>
  <tips-dialog
    iconUrl="https://gw.alipayobjects.com/zos/rmsportal/AzRAgQXlnNbEwQRvEwiu.png"
    type="rectangle"
    className="rectangle"
    onCloseTap="onCloseTap"
    show="{{showRectangle}}">
    <view class="content" slot="content">
      把"城市服务"添加到首页
    </view>
    <view slot="operation" class="add-home" onTap="onRectangleTap">立即添加</view>
  </tips-dialog>
</view>
```

```
Page({
  data: {
    showRectangle: true,
    showDialog: true,
  },
  onCloseTap() {
    this.setData({
      showRectangle: false,
    });
  },
  onRectangleTap() {
    my.alert({
      content: 'do something',
    });
  },
  onDialogTap() {
    this.setData({
      showDialog: false,
    });
  },
});
```

```
.rectangle {
  position: fixed;
  bottom: 100px;
}

.dialog {
  position: fixed;
  bottom: 10px;
}

.content {
  font-size: 14px;
  color: #fff;
}

.opt-button {
  width: 51px;
  height: 27px;
  display: flex;
  justify-content: center;
  align-items: center;
  color: #fff;
  font-size: 12px;
  border: #68BAF7 solid 1rpx;
}

.add-home {
  width: 72px;
  height: 27px;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #56ADEB;
  color: #fff;
  font-size: 14px;
}
```

tips-plain

```
<tips-plain onClose="onClose" time="{{time}}">{{content}}</tips-plain>
```

```
Page({
  data: {
    content: '我知道了',
    time: 2000,
  },
  onClose() {
    my.alert({
      title: '12321'
    });
  }
});
```

属性

tips-dialog

属性	说明	类型	默认值	必填
className	自定义 class	String	-	否
show	控制组件是否展示	Boolean	true	否
type	dialog 表示对话框的样式类型，rectangle 表示矩形的样式类型。	String	dialog	否
onCloseTap	当 type 值为 rectangle 时，组件点击关闭 icon 的回调	() => void	-	否
iconUrl	展示 icon 的 url 地址	String	-	否

slots

slotName	说明
content	用于渲染提示的正文内容
operation	用于渲染右侧操作区域

tips-plain

属性	说明	类型	默认值	必填
className	自定义class。	String	-	否
time	自动关闭时间(单位毫秒)。	Number	5000(ms)	否
onClose	回调并关闭提示框。	() => void	-	false

3.2.6.2. Notice 通告栏

当应用有重要公告或者由于用户的刷新操作产生提示反馈时可以使用通告栏系统。通告栏不会对用户浏览当前页面内容产生影响，但又能明显的引起用户的注意。公告内容不超过一行。

说明：

- 仅用于 UI 展示没有对应的业务逻辑功能。
- notice 为瀑布流布局不会定位到页面头部，用户可以根据需求将它放在相应位置。

扫码体验

示例代码

```
// API-DEMO page/component/notice/notice.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "notice": "mini-antui/es/notice/index"
  }
}
```

```
<!-- API-DEMO page/component/notice/notice.axml -->
<view class="demo-title">NoticeBar 通告栏</view>
<view class="demo-item">
  <notice>因全国公民身份系统升级，添加银行卡银行卡银行卡银行卡</notice>
</view>
<view class="demo-item">
  <notice enableMarquee="{{true}}" marqueeProps="{{loop: true, leading: 500, trailing: 800, fps: 40 }}">因全国公民身份系统升级，添加银行卡银行卡银行卡银行卡</notice>
</view>
<view class="demo-item">
  <notice mode="link" onClick="linkClick">因全国公民身份系统升级，添加银行卡银行卡银行卡银行卡</notice>
</view>
<view class="demo-item">
  <notice mode="closable" onClick="closableClick" show="{{closeShow}}">因全国公民身份系统升级，添加银行卡银行卡银行卡银行卡</notice>
</view>
<view class="demo-item">
  <notice mode="link" action="去看看" onClick="linkActionClick">因全国公民身份系统升级，添加银行卡银行卡银行卡银行卡</notice>
</view>
<view class="demo-item">
  <notice mode="closable" action="不再提示" onClick="closableActionClick" show="{{closeActionShow}}">因全国公民身份系统升级，添加银行卡银行卡银行卡银行卡</notice>
</view>
```

```
// API-DEMO page/component/notice/notice.js
Page({
  data:{
    closeShow:true,
    closeActionShow:true
  },
  linkClick() {
    my.showToast({
      content: '你点击了图标Link NoticeBar',
      duration: 3000
    });
  },
  closableClick() {
    this.setData({
      closeShow:false
    })
    my.showToast({
      content: '你点击了图标close NoticeBar',
      duration: 3000
    });
  },
  linkActionClick() {
    my.showToast({
      content: '你点击了文本Link NoticeBar',
      duration: 3000
    });
  },
  closableActionClick() {
    this.setData({
      closeActionShow:false
    })
    my.showToast({
      content: '你点击了文本close NoticeBar',
      duration: 3000
    });
  }
})
```

```
/* API-DEMO page/component/notice/notice.acss */
.demo-title{
  font-size: 15px;
  font-weight: 500;
  padding: 10px 5px;
  border-bottom: 1px solid #ccc;
}

.demo-item{
  margin-bottom: 10px;
}
```

属性

属性名	描述	类型	默认值
mode	提示可选类型：link（以链接形式展示通告，默认以 > 表示）、closable（告知该通告可关闭，默认以 X 表示）。	String	''
action	提示显示文本。	String	''
actionCls	提示显示文本自定义 class。	String	''
show	是否显示通告栏。	Boolean	true
onClick	点击按钮回调。	() => void	-

enableMarquee	是否开启动画。	Boolean	false
marqueeProps	marquee 参数，其中 loop 表示是否循环，leading 表示动画开启前停顿，trailing 表示 loop 为 true 时，动画间停顿，fps 表示动画帧率。	Object<loop, leading, trailing, fps>	{loop: false, leading: 500, trailing: 800, fps: 40 }

3.2.6.3. Badge 徽标

红点、数字或文字。用于告诉用户待处理的事物或者更新数。

扫码体验

示例代码

```
// API-DEMO page/component/badge/badge.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index",
    "badge": "mini-antui/es/badge/index"
  }
}
```

```
<!-- API-DEMO page/component/badge/badge.axml -->
<view>
  <list>
    <block a:for="{{items}}">
      <list-item
        arrow="{{true}}"
        index="{{index}}"
        key="items-{{index}}"
        last="{{index === (items.length - 1)}}"
      >
        <view>
          <badge a:if="{{item.isWrap}}" text="{{item.text}}" dot="{{item.dot}}">
            <view slot="inner" style="height: 26px; width: 26px; background-color: #ddd;"></view>
          </badge>
          <text style="margin-left: {{ item.isWrap ? '12px' : '0' }}">{{item.intro}}</text>
        </view>
        <view slot="extra">
          <badge a:if="{{!item.isWrap}}" text="{{item.text}}" dot="{{item.dot}}" overflowCount="{{item.overflowCount}}" />
        </view>
      </list-item>
    </block>
  </list>
</view>
```

```
// API-DEMO page/component/badge/badge.js
Page({
  data: {
    items: [
      {
        dot: true,
        text: '',
        isWrap: true,
        intro: 'Dot Badge',
      },
      {
        dot: false,
        text: 1,
        isWrap: true,
        intro: 'Text Badge',
      },
      {
        dot: false,
        text: 99,
        isWrap: false,
        intro: '数字',
      },
      {
        dot: false,
        text: 100,
        overflowCount: 99,
        isWrap: false,
        intro: '数字超过overflowCount',
      },
      {
        dot: false,
        text: 'new',
        isWrap: false,
        intro: '文字',
      },
    ],
  },
});
```

属性

属性名	描述	类型	默认值	必填
text	展示的数字或文案。	String / Number	-	否
dot	不展示数字，只有一个小红点。	Boolean	-	否
overflowCount	展示封顶的数字值，超出部分用“+”号表示。	Number	99	否

slots

slotName	说明
inner	可选，badge 作为 wrapper 时，用于渲染内部的区域。

3.2.7. 表单类

3.2.7.1. InputItem 文本输入

文本输入。

扫码体验

示例代码

```
// API-DEMO page/component/input-item/input-item.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index",
    "input-item": "mini-antui/es/input-item/index",
    "picker-item": "mini-antui/es/picker-item/index"
  }
}
```

```
<!-- API-DEMO page/component/input-item/input-item.axml -->
<view>
  <view style="margin-top: 10px;" />
  <list>
    <input-item
      data-field="cardNo"
      clear="{{true}}"
      value="{{cardNo}}"
      className="dadada"
      placeholder="银行卡号"
      focus="{{inputFocus}}"
      onInput="onItemInput"
      onFocus="onItemFocus"
      onBlur="onItemBlur"
      onConfirm="onItemConfirm"
      onClear="onClear"
    >
      卡号
      <view slot="extra" class="extra" onTap="onExtraTap"></view>
    </input-item>
    <picker-item
      data-field="bank"
      placeholder="选择发卡银行"
      value="{{bank}}"
      onPickerTap="onPickerTap"
    >
      发卡银行
    </picker-item>
    <input-item
      data-field="name"
      placeholder="姓名"
      type="text"
      value="{{name}}"
      clear="{{true}}"
      onInput="onItemInput"
      onClear="onClear"
    >
      姓名
    </input-item>
    <input-item
      data-field="password"
      placeholder="密码"
      password
    >
      密码
    </input-item>
    <input-item
      data-field="remark"
      placeholder="备注"
      last="{{true}}"
    />
  </list>
  <view style="margin: 10px;">
    <button type="primary" onTap="onAutoFocus">聚焦</button>
  </view>
</view>
```

```
// API-DEMO page/component/input-item/input-item.js
const banks = ['网商银行', '建设银行', '工商银行', '浦发银行'];

Page({
  data: {
    cardNo: '1234****',
    inputFocus: true,
    bank: '',
    name: '',
  },
  onAutoFocus() {
    this.setData({
      inputFocus: true,
    });
  },
  onExtraTap() {
    my.alert({
      content: 'extra tapped',
    });
  },
  onItemInput(e) {
    this.setData({
      [e.target.dataset.field]: e.detail.value,
    });
  },
  onItemFocus() {
    this.setData({
      inputFocus: false,
    });
  },
  onItemBlur() {},
  onItemConfirm() {},
  onClear(e) {
    this.setData({
      [e.target.dataset.field]: '',
    });
  },
  onPickerTap() {
    my.showActionSheet({
      title: '选择发卡银行',
      items: banks,
      success: (res) => {
        this.setData({
          bank: banks[res.index],
        });
      },
    });
  },
});
```

```
/* API-DEMO page/component/input-item/input-item.acss */
.extra {
  background-image: url('https://gw.alipayobjects.com/zos/rmsportal/dOfSJfWQvYdvsZiJStvg.svg');
  background-size: contain;
  background-repeat: no-repeat;
  background-position: right center;
  opacity: 0.2;
  height: 20px;
  width: 20px;
  padding-left: 10px;
}
```

属性

属性名	描述	类型	默认值
className	自定义的 class。	String	''
labelCls	自定义 abel 的 class。	String	''
inputCls	自定义 input 的 class。	String	''
last	是否最后一行。	Boolean	false
value	初始内容。	String	''
name	组件名字，用于表单提交获取数据。	String	''
type	input 的类型，有效值：text，number，idcard，digit（详情请见下方表格）。	String	text
password	是否是密码类型。	Boolean	false
placeholder	占位符。	String	''
placeholderStyle	指定 placeholder 的样式。	String	''
placeholderClass	指定 placeholder 的样式类。	String	''
disabled	是否禁用。	Boolean	false
maxlength	最大长度。	Number	140
focus	获取焦点。	Boolean	false
clear	是否带清除功能，仅 disabled 为 false 才生效。	Boolean	false
onInput	键盘输入时触发 input 事件。	(e: Object) => void	-
onConfirm	点击键盘完成时触发。	(e: Object) => void	-
onFocus	聚焦时触发。	(e: Object) => void	-
onBlur	失去焦点时触发。	(e: Object) => void	-
onClear	点击清除 icon 时触发。	() => void	-

type 属性值介绍

- text： 字符输入框
- number： 纯数字输入框（ 0-9 之间的数字）
- idcard：身份证输入框（ 0-9 之间的数字，以及字符 x）
- digit：数字输入框，（ 0-9 之间的数字，以及小数点 . 字符，可用于含有小数的数字）

注意：type 的属性值影响的是真机中的键盘类型，在模拟器中并不一定会有效果。

slots

slotname	说明	必填
extra	用于渲染input-item项右边说明。	否

常见问题

为何 setData 数据为空时，断点 money 值已经置空，但是在输入框还是显示0?

this.setData 设置 data 为空时，不会渲染页面，建议使用组件的 clear。

相关链接

- input 输入框

3.2.7.2. PickerItem 选择输入

选择输入。

扫码体验

示例代码

```
// API-DEMO page/component/input-item/input-item.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index",
    "input-item": "mini-antui/es/input-item/index",
    "picker-item": "mini-antui/es/picker-item/index"
  }
}
```

```
<!-- API-DEMO page/component/input-item/input-item.axml -->
<view>
  <view style="margin-top: 10px;" />
  <list>
    <input-item
      data-field="cardNo"
      clear="{{true}}"
      value="{{cardNo}}"
      className="dadada"
      placeholder="银行卡号"
      focus="{{inputFocus}}"
      onInput="onItemInput"
      onFocus="onItemFocus"
      onBlur="onItemBlur"
      onConfirm="onItemConfirm"
      onClear="onClear"
    >
      卡号
      <view slot="extra" class="extra" onTap="onExtraTap"></view>
    </input-item>
    <picker-item
      data-field="bank"
      placeholder="选择发卡银行"
      value="{{bank}}"
      onPickerTap="onPickerTap"
    >
      发卡银行
    </picker-item>
    <input-item
      data-field="name"
      placeholder="姓名"
      type="text"
      value="{{name}}"
      clear="{{true}}"
      onInput="onItemInput"
      onClear="onClear"
    >
      姓名
    </input-item>
    <input-item
      data-field="password"
      placeholder="密码"
      password
    >
      密码
    </input-item>
    <input-item
      data-field="remark"
      placeholder="备注"
      last="{{true}}"
    />
  </list>
  <view style="margin: 10px;">
    <button type="primary" onTap="onAutoFocus">聚焦</button>
  </view>
</view>
```

```
// API-DEMO page/component/input-item/input-item.js
const banks = ['网商银行', '建设银行', '工商银行', '浦发银行'];

Page({
  data: {
    cardNo: '1234****',
    inputFocus: true,
    bank: '',
    name: '',
  },
  onAutoFocus() {
    this.setData({
      inputFocus: true,
    });
  },
  onExtraTap() {
    my.alert({
      content: 'extra tapped',
    });
  },
  onItemInput(e) {
    this.setData({
      [e.target.dataset.field]: e.detail.value,
    });
  },
  onItemFocus() {
    this.setData({
      inputFocus: false,
    });
  },
  onItemBlur() {},
  onItemConfirm() {},
  onClear(e) {
    this.setData({
      [e.target.dataset.field]: '',
    });
  },
  onPickerTap() {
    my.showActionSheet({
      title: '选择发卡银行',
      items: banks,
      success: (res) => {
        this.setData({
          bank: banks[res.index],
        });
      },
    });
  },
});
```

```
/* API-DEMO page/component/input-item/input-item.acss */
.extra {
  background-image: url('https://gw.alipayobjects.com/zos/rmsportal/dOfSJfWQvYdvsZiJStvg.svg');
  background-size: contain;
  background-repeat: no-repeat;
  background-position: right center;
  opacity: 0.2;
  height: 20px;
  width: 20px;
  padding-left: 10px;
}
```

属性

属性名	描述	类型	默认值

className	自定义的 class。	String	-
labelCls	自定义 label 的class。	String	-
pickerCls	自定义选择区域的 class。	String	-
last	是否最后一行。	Boolean	false
value	初始内容。	String	-
name	组件名字，用于表单提交获取数据。	String	-
placeholder	占位符。	String	-
onPickerTap	点击 pickeritem 时触发。	(e: Object) => void	-

slots

slotname	说明	必填
extra	用于渲染 picker-item 项右边说明。	否

3.2.7.3. AmountInput 金额输入

金额输入框。

扫码体验

示例代码

```
// API-DEMO page/component/amount-input/amount-input.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "amount-input": "mini-antui/es/amount-input/index"
  }
}
```

```
<!-- API-DEMO page/component/amount-input/amount-input.axml -->
<view>
  <amount-input
    type="digit"
    title="转入金额"
    extra="建议转入¥100以上金额"
    placeholder="输入转入金额"
    value="{{value}}"
    maxLength="5"
    focus="{{true}}"
    btnText="全部提现"
    onClear="onInputClear"
    onInput="onInput"
    onConfirm="onInputConfirm" />
</view>
```

```
// API-DEMO page/component/amount-input/amount-input.js
Page({
  data: {
    value: 200,
  },
  onInputClear() {
    this.setData({
      value: '',
    });
  },
  onInputConfirm(e) {
    console.log(e);
    my.alert({
      content: 'confirmed',
    });
  },
  onInput(e) {
    console.log(e);
    const { value } = e.detail;
    this.setData({
      value,
    });
  },
  onButtonClick() {
    my.alert({
      content: 'button clicked',
    });
  },
  onInputFocus(e) {
    console.log(e);
  },
  onInputBlur(e) {
    console.log(e);
  },
});
```

属性

属性名	描述	类型	默认值	必填
type	input 的类型，有效值：digit、number。	String	number	否
title	左上角标题。	String	-	否
extra	左下角说明。	String	-	否
value	输入框当前值。	String	-	否

btnText	右下角按钮文案。	String	-	否
placeholder	placeholder。	String	-	否
focus	自动获取光标。	Boolean	false	否
onInput	键盘输入时触发。	(e: Object) => void	-	否
onFocus	获取焦点时触发。	(e: Object) => void	-	否
onBlur	失去焦点时触发。	(e: Object) => void	-	否
onConfirm	点击键盘完成时触发。	(e: Object) => void	-	否
onClear	点击 clear 图标触发。	() => void	-	否
onButtonClick	点击右下角按钮时触发。	() => void	-	否
maxLength	最多允许输入的字符个数。	Number	-	否
controlled	是否为受控组件。为 true 时，value 内容会完全受 setData 控制。	Boolean	false	否

3.2.7.4. SearchBar 搜索框

搜索提供了用户进行文本查询的功能，用户可以针对当前页面的内容通过精确搜索和模糊搜索进行内容筛选和定位，提高查询效率。搜索栏激活后出现取消按钮。

说明：仅用于 UI 展示没有对应的业务逻辑功能。

扫码体验

示例代码

```
// API-DEMO page/component/search-bar/search-bar.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "search-bar": "mini-antui/es/search-bar/index"
  }
}
```

```
<!-- API-DEMO page/component/search-bar/search-bar.axml -->
<view>
  <search-bar
    value="{{value}}"
    placeholder="搜索"
    onInput="handleInput"
    onClear="handleClear"
    onFocus="handleFocus"
    onBlur="handleBlur"
    onCancel="handleCancel"
    onSubmit="handleSubmit"
    showCancelButton="{{false}}" />
</view>
```

```
// API-DEMO page/component/search-bar/search-bar.js
Page({
  data: {
    value: '美食',
  },
  handleInput(value) {
    this.setData({
      value,
    });
  },
  handleClear(value) {
    this.setData({
      value: '',
    });
  },
  handleFocus() {},
  handleBlur() {},
  handleCancel() {
    this.setData({
      value: '',
    });
  },
  handleSubmit(value) {
    my.alert({
      content: value,
    });
  },
});
```

属性

属性名	描述	类型	默认值	必填
value	搜索框的当前值。	String	-	否
placeholder	placeholder。	String	-	否
focus	自动获取光标。	Boolean	false	否
onInput	键盘输入时触发。	(value: String) => void	-	否
onClear	点击 clear 图标触发。	(val: String) => void	-	否
onFocus	获取焦点时触发。	() => void	-	否
onBlur	失去焦点时触发。	() => void	-	否
onCancel	点击取消时触发。	() => void	-	否
onSubmit	点击键盘的 enter 时触发。	(val: String) => void	-	否

disabled	设置禁用。	Boolean	-	否
maxLength	最多允许输入的字符个数。	Number	-	否
showCancelButton	是否一直显示取消按钮。	Boolean	-	否

3.2.7.5. AMCheckBox 复选框

复选框。

扫码体验

示例代码

```
// API-DEMO page/component/am-checkbox/am-checkbox.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index",
    "am-checkbox": "mini-antui/es/am-checkbox/index"
  }
}
```

```
<!-- API-DEMO page/component/am-checkbox/am-checkbox.axml -->
<list>
  <view slot="header">
    列表+复选框
  </view>
  <block a:for="{{items}}">
    <list-item
      thumb=""
      arrow="{{false}}"
      index="{{index}}"
      key="items-{{index}}"
      last="{{index === (items.length - 1)}}"
    >
      <view slot="prefix" style="display: flex; align-items: center;">
        <am-checkbox id="{{item.id}}" data-name="{{item.value}}" disabled="{{item.disabled}}" checked="{{item.check
ed}}" onChange="onChange" />
      </view>
      <label for="{{item.id}}">{{item.title}}</label>
    </list-item>
  </block>
</list>
<view style="padding: 16px;">
  <view style="color: #888; font-size: 14px;">
    协议
  </view>
  <view style="margin-top: 10px;">
    <label style="display: flex; line-height: 24px;">
      <am-checkbox />
      <text style="text-indent: 8px; color: #888">同意 《信用支付服务合同》</text>
    </label>
  </view>
</view>
<view style="padding: 16px; background-color: #fff;">
  <form onSubmit="onSubmit" onReset="onReset">
    <view>
      <view style="color: #666; font-size: 14px; margin-bottom: 5px;">选择你用过的框架：</view>
      <view>
        <checkbox-group name="libs">
          <label a:for="{{items2}}" style="display: flex; align-items: center; height: 30px;">
            <am-checkbox value="{{item.name}}" checked="{{item.checked}}" disabled="{{item.disabled}}" />
            <text style="color: #888; font-size: 14px; margin-left: 8px;">{{item.value}}</text>
          </label>
        </checkbox-group>
      </view>
      <view style="margin-top: 10px;">
        <button type="primary" size="mini" formType="submit">submit</button>
      </view>
    </view>
  </form>
</view>
```

```
// API-DEMO page/component/am-checkbox/am-checkbox.js
Page({
  data: {
    items: [
      { checked: true, disabled: false, value: 'a', title: '复选框-默认选中', id: 'checkbox1' },
      { checked: false, disabled: false, value: 'b', title: '复选框-默认未选中', id: 'checkbox2' },
      { checked: true, disabled: true, value: 'c', title: '复选框-默认选中disabled', id: 'checkbox3' },
      { checked: false, disabled: true, value: 'd', title: '复选框-默认未选中disabled', id: 'checkbox4' },
    ],
    items2: [
      { name: 'react', value: 'React', checked: true },
      { name: 'vue', value: 'Vue.js' },
      { name: 'ember', value: 'Ember.js' },
      { name: 'backbone', value: 'Backbone.js', disabled: true },
    ],
  },
  onSubmit(e) {
    my.alert({
      content: `你选择的框架是 ${e.detail.value.libs.join(', ')}`,
    });
  },
  onReset() {},
  onChange(e) { console.log(e); },
});
```

属性

属性名	描述	类型	默认值	必填
value	组件值，选中时 change 事件会携带的 value。	String	-	否
checked	当前是否选中，可用来设置默认选中。	Boolean	false	否
disabled	是否禁用。	Boolean	false	否
onChange	change 事件触发的回调函数。	(e: Object) => void	-	否
id	与 label 组件的 for 属性组合使用。	String	-	否

3.2.8. 手势类

3.2.8.1. SwipeAction 可滑动单元格

可滑动单元格。

扫码体验

示例代码

```
// API-DEMO page/component/swiper-action/swiper-action.json
{
  "defaultTitle": "SwipeAction",
  "usingComponents": {
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index",
    "swipe-action": "mini-antui/es/swipe-action/index"
  }
}
```

```
<!-- API-DEMO page/component/swiper-action/swiper-action.axml -->
<view>
  <list>
    <view a:for="{{list}}" key="{{item.content}}">
      <swipe-action
        index="{{index}}"
        restore="{{swipeIndex === null || swipeIndex !== index}}"
        right="{{item.right}}"
        onRightItemClick="onRightItemClick"
        onSwipeStart="onSwipeStart"
        extra="item{{index}}"
      >
        <list-item
          arrow="horizontal"
          index="{{index}}"
          key="items-{{index}}"
          onClick="onItemClick"
          last="{{index === list.length - 1}}"
        >
          {{item.content}}
        </list-item>
      </swipe-action>
    </view>
  </list>
</view>
```

```
// API-DEMO page/component/swiper-action/swiper-action.js
Page({
  data: {
    swipeIndex: null,
    list: [
      { right: [{ type: 'edit', text: '取消收藏', bgColor: '#ccc', fColor: '#f00' }, { type: 'delete', text: '删除',
bgColor: '#0ff', fColor: '#333' }], content: '文字和背景色同时改变' },
      { right: [{ type: 'delete', text: '删除' }], content: 'AAA' },
      { right: [{ type: 'edit', text: '取消收藏' }, { type: 'delete', text: '删除' }], content: 'BBB' },
      { right: [{ type: 'delete', text: '删除' }], content: 'CCC' },
    ],
  },
  onRightItemClick(e) {
    const { type } = e.detail;
    my.confirm({
      title: '温馨提示',
      content: `${e.index}-${e.extra}-${JSON.stringify(e.detail)}`,
      confirmButtonText: '确定',
      cancelButtonText: '取消',
      success: (result) => {
        const { list } = this.data;
        if (result.confirm) {
          if (type === 'delete') {
            list.splice(this.data.swipeIndex, 1);
            this.setData({
              list: [...list],
            });
          }

          my.showToast({
            content: '确定 => 执行滑动删除还原',
          });
          e.done();
        } else {
          my.showToast({
            content: '取消 => 滑动删除状态保持不变',
          });
        }
      },
    });
  },
  onItemClick(e) {
    my.alert({
      content: `dada${e.index}`,
    });
  },
  onSwipeStart(e) {
    this.setData({
      swipeIndex: e.index,
    });
  },
});
```

属性

属性名	描述	类型	默认值
className	自定义class。	String	-

right	滑动选项，最多两项。	Array[Object{type: `edit` / `delete`, text: string, fColor: '颜色值', bgColor: '颜色值'}]	[]
onRightItemClick	点击滑动选项。	({index, detail, extra, done}) => void	调用 done 从而使 swipeAction 合上
restore	还原组件到初始状态，当有多个 swipeAction 组件时，当滑动其中一个时，需要将其他的组件的 restore 属性设置为 true，避免一个页面同时存在多个 swipeAction 处于活动状态。	Boolean	false
onSwipeStart	开始滑动回调。	(e: Object) => void	-
extra	附属信息，会在 onRightItemClick 回调中获取。	any	-

3.2.9. 其他

3.2.9.1. Calendar 日历

日历。

注意：

- 日历组件不支持跨月份选择日期范围。
- 使用 onMonthChange 属性取值拿到选中的日期。

扫码体验

示例代码

```
// API-DEMO page/component/calendar/calendar.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents":{
    "calendar": "mini-antui/es/calendar/index"
  }
}
```

```
<!-- API-DEMO page/component/calendar/calendar.axml -->
<view>
  <calendar
    type="range"
    haveYear="{{true}}"
    tagData="{{tagData}}"
    onSelect="handleSelect"
    onMonthChange="onMonthChange"
    onYearChange="onYearChange"
    onSelectHasDisableDate="onSelectHasDisableDate" />
</view>
```

```
// API-DEMO page/component/calendar/calendar.js
Page({
  data: {
    tagData: [
      { date: '2019-09-14', tag: '还房贷', tagColor: 5 },
      { date: '2019-09-28', tag: '公积金', tagColor: 2 },
      { date: '2019-09-18', tag: 'xx', disable: true },
    ],
  },
  handleSelect() {},
  onMonthChange() {},
  onYearChange() {},
  onSelectHasDisableDate() {
    my.alert({
      content: 'SelectHasDisableDate',
    });
  },
});
```

属性

属性名	描述	类型	默认值	必填
type	选择类型 single: 单日 range: 日期区间。	String	single	否
haveYear	是否展示年份控制箭头。	Boolean	false	false
tagData	标记数据，其中包括 日期：date，标记：tag，是否禁用：disable，标记颜色：tagColor，tagColor有4个可选值，1.#f5a911，2.#e8541e 3.#07a89b 4.#108ee9，5.#b5b5b5。	Array<date, tag, tagColor>	-	否
onSelect	选择区间回调。	([startDate, endDate]) => void	-	否
onMonthChange	点击切换月份时回调，带两个参数：currentMonth 切换后月份和 prevMonth 切换前月份。	(currentMonth, prevMonth) => void	-	否

onSelectHasDisableDate	选择区间包含不可用日期。	(currentMonth, prevMonth) => void	-	否
onYearChange	点击切换年份时回调，带两个参数：currentYear 切换后的年份和 prevYear 切换前的年份。	(currentYear, prevYear) => void	-	否

说明：onYearChange 仅支持 mini-antui 0.4.32 以上版本。

3.2.9.2. Stepper 步进器

用作增加或者减少当前数值。

注意：

- 输入最大值无提示，超过最大值时系统会自动回显数值为最大值。
- 不支持输入小数，可通过 + 和 - 改变数值大小。

扫码体验

示例代码

```
// API-DEMO page/component/stepper/stepper.json
{
  "defaultTitle": "Stepper",
  "usingComponents":{
    "stepper": "mini-antui/es/stepper/index",
    "list": "mini-antui/es/list/index",
    "list-item": "mini-antui/es/list/list-item/index"
  }
}
```

```
<!-- API-DEMO page/component/stepper/stepper.axml -->
<list>
  <list-item disabled="{{true}}">
    Show number value
    <view slot="extra">
      <stepper onChange="callBackFn" step="{{1}}" showNumber readOnly="{{false}}" value="{{value}}" min="{{2}}" max
="{{12}}" />
    </view>
  </list-item>
  <list-item disabled="{{true}}">
    Do not show number value
    <view slot="extra">
      <stepper onChange="callBackFn" step="{{1}}" readOnly="{{false}}" value="{{value}}" min="{{2}}" max="{{12}}" /
>
    </view>
  </list-item>
  <list-item disabled="{{true}}">
    Disabled
    <view slot="extra">
      <stepper onChange="callBackFn" showNumber value="{{11}}" min="{{2}}" max="{{12}}" disabled />
    </view>
  </list-item>
  <list-item disabled="{{true}}">
    readOnly
    <view slot="extra">
      <stepper onChange="callBackFn" showNumber value="{{11}}" min="{{2}}" max="{{12}}" readOnly />
    </view>
  </list-item>
  <list-item>
    <button onTap="modifyValue">修改stepper初始值</button>
  </list-item>
</list>
```

```
// API-DEMO page/component/stepper/stepper.js
Page({
  data: {
    value: 8,
  },
  callBackFn(value){
   console.log(value);
  },
  modifyValue() {
    this.setData({
      value: this.data.value + 1,
    });
  }
});
```

属性

属性名	描述	类型	默认值	必填
min	最小值。	Number	0	是
max	最大值。	Number	10000	是
value	初始值。	Number	10	是
step	每次改变步数，可以为小数。	Number	1	否
onChange	变化时回调函数。	(value: Number) => void	-	否
disabled	禁用。	Boolean	false	否
readOnly	input 只读。	Boolean	false	否

showNumber	是否显示数值，默认不显示。	Boolean	false	否

3.2.9.3. AMIcon 图标

图标。

注意：

- 建议使用体积较小的字体。
- Amicon不支持 onTap 事件，可以在外层加一个 view 标签。
- AMIcon 是自定义组件封装的扩展组件，上面没有提供点击事件；自行绑定一个点击事件，不会生效，可以到 modules 里面找到源码，修改源码来新增点击事件。

扫码体验

示例代码

```
// API-DEMO page/component/am-icon/am-icon.json
{
  "defaultTitle": "小程序AntUI组件库",
  "usingComponents": {
    "am-icon": "mini-antui/es/am-icon/index"
  }
}
```

```
<!-- API-DEMO page/component/am-icon/am-icon.axml -->
<view>
  <view class="icon-title">基础</view>
  <view class="icon-list">
    <block a:for="{{basicTypes}}">
      <view class="icon-item">
        <am-icon type="{{item}}" />
        <text class="icon-desc">{{item}}</text>
      </view>
    </block>
  </view>
  <view class="icon-title">描边风格</view>
  <view class="icon-list">
    <block a:for="{{strokeTypes}}">
      <view class="icon-item">
        <am-icon type="{{item}}" />
        <text class="icon-desc">{{item}}</text>
      </view>
    </block>
  </view>
  <view class="icon-title">实心风格</view>
  <view class="icon-list">
    <block a:for="{{solidTypes}}">
      <view class="icon-item">
        <am-icon type="{{item}}" />
        <text class="icon-desc">{{item}}</text>
      </view>
    </block>
  </view>
</view>
```

```
// API-DEMO page/component/am-icom/am-icon.js
Page({
  data: {
    basicTypes: [
      'arrow-left',
      'arrow-up',
      'arrow-right',
      'arrow-down',
      'cross',
      'plus',
    ],
    strokeTypes: [
      'close-o',
      'dislike-o',
      'heart-o',
      'help-o',
      'like-o',
      'location-o',
      'info-o',
      'success-o',
      'wait-o',
      'warning-o',
      'star-o',
      'download',
      'friends',
      'circle',
      'delete',
      'charge',
      'card',
      'notice',
      'qrcode',
      'reload',
      'scan',
      'money',
      'search',
      'setting',
      'share',
      'zoom-in',
      'zoom-out',
    ],
    solidTypes: [
      'close',
      'dislike',
      'heart',
      'help',
      'like',
      'location',
      'info',
      'success',
      'wait',
      'warning',
      'star',
    ],
  },
});
```

```
/* API-DEMO page/component/am-icon/am-icon.acss */
.icon-title {
  margin-top: 20px;
  margin-bottom: 10px;
  margin-left: 10px;
  color: #333;
  font-size: 16px;
}

.icon-list {
  background: #fff;
}

.icon-item {
  display: inline-flex;
  width: 33.33333%;
  height: 80px;
  align-items: center;
  flex-direction: column;
  justify-content: center;
}

.icon-desc {
  margin-top: 10px;
}
```

属性

属性名	描述	类型	必填
type	icon 类型，具体效果扫上面二维码预览（有效值请见下方表格）。	String	是
size	icon 大小，单位 px。	String	否
color	icon 颜色，同 css 的 color。	String	否

type 有效值

图标风格	type有效值
基础类型	`arrow-left` 、 `arrow-up` 、 `arrow-right` 、 `arrow-down` 、 `cross` 、 `plus`
描边风格	`close-o` 、 `dislike-o` 、 `heart-o` 、 `help-o` 、 `like-o` 、 `location-o` 、 `info-o` 、 `success-o` 、 `wait-o` 、 `warning-o` 、 `star-o` 、 `download` 、 `friends` 、 `circle` 、 `delete` 、 `charge` 、 `card` 、 `notice` 、 `qrcode` 、 `reload` 、 `scan` 、 `money` 、 `search` 、 `setting` 、 `share` 、 `zoom-in` 、 `zoom-out`
实心风格	`close` 、 `dislike` 、 `heart` 、 `help` 、 `like` 、 `location` 、 `info` 、 `success` 、 `wait` 、 `warning` 、 `star`

4.API

4.1. 概览

小程序

名称	功能说明
my.getAppIdSync	同步获取小程序 appId。
my.onAppHide	监听小程序切后台事件。
my.offAppHide	取消监听小程序切后台事件。
my.onAppShow	监听小程序切前台事件。
my.offAppShow	取消监听小程序切前台事件。

界面

导航栏

名称	功能说明
my.getTitleColor	获取导航栏背景色。
my.hideBackHome	隐藏 TitleBar 上的返回首页图标，和通用菜单中的返回首页功能。
my.showNavigationBarLoading	在当前页面显示导航条加载动画。
my.hideNavigationBarLoading	在当前页面隐藏导航条加载动画。
my.setNavigationBar	设置导航栏文字及样式。

TabBar

名称	功能说明
my.hideTabBar	隐藏 tabBar。
my.hideTabBarRedDot	隐藏 tabBar 某一项的右上角的红点。
my.removeTabBarBadge	移除 tabBar 某一项右上角的文本。
my.setTabBarBadge	为 tabBar 某一项的右上角添加文本。
my.setTabBarItem	动态设置 tabBar 某一项的内容。

my.setTabBarStyle	动态设置 tabBar 的整体样式。
my.showTabBar	显示 tabBar。
my.showTabBarRedDot	显示 tabBar 某一项的右上角的红点。
onTabItemTap	点击 tab 时触发。
TabBar 常见问题	对 TabBar 标签的常见问题解答。

路由

名称	功能说明
my.switchTab	跳转到指定 tabBar 页面，并关闭其他所有非 tabBar 页面。
my.reLaunch	关闭当前所有页面，跳转到应用内的某个指定页面。
my.redirectTo	关闭当前页面，跳转到应用内的某个指定页面。
my.navigateTo	从当前页面，跳转到应用内的某个指定页面，以使用 `my.navigateBack` 返回到原来页面。
my.navigateBack	关闭当前页面，返回上一级或多级页面。
路由 FAQ	对路由的常见问题解答。

交互反馈

名称	功能说明
my.alert	警告框。
my.confirm	确认框。
my.prompt	弹出一个对话框，让用户在对话框内输入文本。
my.showToast	显示一个弱提示，可选择多少秒之后消失。
my.hideToast	隐藏弱提示。
my.showLoading	显示加载提示。
my.hideLoading	隐藏加载提示。

my.showActionSheet	显示操作菜单。

下拉刷新

名称	功能说明
onPullDownRefresh	监听该页面用户的下拉刷新事件。
my.startPullDownRefresh	启用当前页面的下拉刷新。
my.stopPullDownRefresh	停用当前页面的下拉刷新。

选择日期

名称	功能说明
my.datePicker	打开日期选择列表。

动画

名称	功能说明
my.createAnimation	创建动画实例。

画布

名称	功能说明
my.createCanvasContext	创建 canvas 绘图上下文。

滚动

名称	功能说明
my.pageScrollTo	滚动到页面的目标位置。

设置窗口背景

名称	功能说明
my.setBackgroundColor	动态设置窗口的背景色。
my.setBackgroundTextStyle	动态设置下拉背景字体、loading 图的样式。

字体

名称	功能说明
my.loadFontFace	动态加载网络字体。

缓存

名称	功能说明
my.setStorage	将数据存储在本地缓存中指定的 key 中的异步接口。
my.setStorageSync	同步将数据存储在本地缓存中指定的 key 中的同步接口。
my.getStorage	获取缓存数据的异步接口。
my.getStorageSync	获取缓存数据的同步接口。
my.removeStorage	删除缓存数据的异步接口。
my.removeStorageSync	删除缓存数据的同步接口。
my.clearStorage	清除本地数据缓存的异步接口。
my.clearStorageSync	清除本地数据缓存的同步接口。
my.getStorageInfo	获取当前 storage 的相关信息的异步接口。
my.getStorageInfoSync	获取当前 storage 相关信息的同步接口。

网络

名称	功能说明
my.request	小程序网络请求。
my.uploadFile	上传本地资源到开发者服务器。
my.downloadFile	下载文件资源到本地。
my.connectSocket	创建一个 WebSocket 的连接。
my.onSocketOpen	监听 WebSocket 连接打开事件。
my.offSocketOpen	取消监听 WebSocket 连接打开事件。
my.onSocketError	监听 WebSocket 错误。

my.offSocketError	取消监听 WebSocket 错误。
my.sendSocketMessage	通过 WebSocket 连接发送数据。
my.onSocketMessage	监听 WebSocket 接受到服务器的消息事件。
my.closeSocket	关闭 WebSocket 连接。
my.onSocketClose	监听 WebSocket 关闭。
my.offSocketClose	取消监听WebSocket关闭。

4.2. 使用说明

阿里云企业应用开发框架提供了丰富的 JSAPI（原生 API）和 OpenAPI（开放能力 API），开发者可方便快捷地调用这些 API，详情请参见 概览。

- OpenAPI 是阿里云开放平台在企业应用上开放的开放能力 API。通过 OpenAPI，小程序可以轻松实现用户授权、获取会员基础信息、跳转ECS列表详情、SSH等多种多样的功能。
- JSAPI 按实现的功能分类，可分为界面、多媒体、缓存、文件、位置、网络、自定义通用菜单等。

其中，JSAPI 分为两大类：事件监听 API、功能 API。

事件监听 API

事件监听型 API 以 `my.on` 开头，用于监听某个系统事件是否触发。

事件监听型 API 接受一个 callback 回调函数作为参数。当具体事件触发时，会触发 callback 函数调用。该 callback 函数可以传给对应以 `my.off` 开头的同名 API 来解除监听关系，如果直接调用以 `my.off` 开头的同名 API 则解除所有监听关系。

以监听低功耗蓝牙设备的特征值变化的事件 API my.onBLECharacteristicValueChange 为例：

```
Page({
  onLoad() {
    this.callback = this.callback.bind(this);
    my.onBLECharacteristicValueChange(this.callback);
  },
  onUnload() {
    // 页面卸载时解除某个监听
    my.offBLECharacteristicValueChange(this.callback);
    // 或者解除所有监听
    // my.offBLECharacteristicValueChange();
  },
  callback(res) {
    console.log(res);
  },
});
```

功能 API

功能型 API 是不以 `my.on` 或 `my.off` 开头的 API，用于实现某个特定功能。功能型 API 可分为异步型 API 和同步型 API。

异步型功能 API

大部分 API 都是异步型功能 API，例如 my.navigateTo、my.request。异步型功能 API 的入参都为一个 Object 对象，并包含三个子属性：

属性	类型	必填	描述
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

回调结果若无特殊说明，一般为一个 Object 对象，包含以下子属性：

属性	类型	说明
errCode	String	错误码，接口调用成功，errorCode=0 。
errorMsg	String	错误信息，接口调用成功返回 `ok` 。
其他	-	API 返回的其他数据。

以发起网络请求的 my.request API 为例：

```
// dataType 为 json 示例
my.request({
  url: 'https://httpbin.org/post',
  method: 'POST',
  data: {
    from: '阿里云',
    production: 'AliyunJSAPI',
  },
  dataType: 'json',
  success: function(res) {
    my.alert({content: 'success'});
  },
  fail: function(res) {
    my.alert({content: 'fail'});
  },
  complete: function(res) {
    my.alert({content: 'complete'});
  }
});
```

同步型功能 API

以 `Sync` 结尾的 API 都是同步型功能 API，例如 my.setStorageSync、my.getStorageInfoSync 等。

同步型功能 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：

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

以上为通用说明，特定 API 的入参及返回值以详细 API 文档为准。

4.3. 小程序

4.3.1. my.canIUse

判断当前小程序的 API、入参或返回值、组件、属性等在当前版本是否支持。

示例代码

```
// 新增 API 是否可用
my.canIUse('getFileInfo')
// API 新增属性是否可用
my.canIUse('closeSocket.object.code')
// API 新增属性是否可用
my.canIUse('getLocation.object.type')
// API 返回值新增属性是否可用
my.canIUse('getSystemInfo.return.brand')
// 新增组件「关注生活号」是否可用
my.canIUse('lifestyle')
// 组件新增属性值是否可用
my.canIUse('button.open-type.share')
```

入参

参数使用 `${API}.${type}.${param}.${option}` 或者 `${component}.${attribute}.${option}` 方式来调用。

- API 表示 API 的名称，不包括 my. 的名称。例如：您想判断 my.getFileInfo，您只需传入 getFileInfo 即可。
- type 取值 object/return/callback 表示 API 的判断类型。
- param 表示参数的某一个属性名。
- option 表示参数属性的具体属性值。
- component 表示组件名称。
- attribute 表示组件属性名。
- option 表示组件属性值。

返回值

为 Boolean 类型，表示是否支持。

4.3.2. my.getAppIdSync

同步获取小程序 APPID。

示例代码

```
const appIdRes = my.getAppIdSync();
console.log(appIdRes.appId);
```

返回值

属性	类型	说明
appId	String	当前小程序的 appId

4.3.3. my.offAppHide

取消监听小程序切后台事件。

使用说明：

- 由于开发者工具版本限制，目前本 API 暂不支持在开发者工具调试和真机调试，仅支持真机预览。开发者请调至 **预览** 模式，在阿里云客户端扫码查看效果。
- 请勿使用 API 监听匿名函数，否则将无法关闭监听。

示例代码

```
<!-- .axml-->
<button size="default" onTap="offAppHideHanlder" type="primary">关闭监听到后台</button>
```

```
//.js
onLoad() {
    my.onAppHide(this.onAppHideHandler)
},
// 监听切换到后台方法
onAppHideHandler() {
    console.log('监听切换到后台方法')
},
// 取消监听切换到后台方法
offAppHideHanlder() {
    my.offAppHide(this.onAppHideHandler)
},
```

入参

入参为回调函数：

属性	类型	说明
回调函数	Function	小程序切后台事件的回调函数。

4.3.4. my.offAppShow

取消监听小程序切前台事件。

使用说明：

- 由于开发者工具版本限制，目前本 API 暂不支持在开发者工具调试和真机调试，仅支持真机预览。开发者请调至 **预览** 模式，在阿里云客户端扫码查看效果。
- 请勿使用 API 监听匿名函数，否则将无法关闭监听。

示例代码

```
<!-- .axml-->
<button size="default" onTap="offAppShowHanlder" type="primary">关闭监听到前台</button>
```

```
//.js
onLoad() {
    my.onAppShow(this.onAppShowHandler)
},
//监听切换到前台方法
onAppShowHandler() {
    console.log('前台了，到了')
},
//取消监听切换到前台方法
offAppShowHanlder() {
    my.offAppShow(this.onAppShowHandler)
},
apphide() {
    console.log('监听后台加载成功')
}
```

入参

入参为回调函数：

属性	类型	说明
回调函数	Function	小程序切前台事件的回调函数。

4.3.5. my.onAppHide

监听小程序切后台事件，该事件与 App.onHide 的回调时机一致。对应的取消监听 API 请参见 my.offAppHide。

使用说明：

- 由于开发者工具版本限制，目前本 API 暂不支持在开发者工具调试和真机调试，仅支持真机预览。开发者请调至 **预览** 模式，在阿里云客户端扫码查看效果。
- 请勿使用 API 监听匿名函数，否则将无法关闭监听。

示例代码

```
<!-- .axml-->
<button size="default" onTap="offAppHideHanlder" type="primary">关闭监听到后台</button>
```

```
//.js
onLoad() {
    my.onAppHide(this.onAppHideHandler)
},
// 监听切换到后台方法
onAppHideHandler() {
    console.log('监听切换到后台方法')
},
// 取消监听切换到后台方法
offAppHideHanlder() {
    my.offAppHide(this.onAppHideHandler)
},
```

入参

入参为回调函数：

属性	类型	说明
回调函数	Function	小程序切后台事件的回调函数。

4.3.6. my.onAppShow

监听小程序切前台事件，该事件与 App.onShow 的回调时机一致。对应的取消监听 API 请参见 my.offAppShow。

使用说明：

- 由于开发者工具版本限制，目前本 API 暂不支持在开发者工具调试和真机调试，仅支持真机预览。开发者请调至 **预览** 模式，在阿里云客户端扫码查看效果。
- 请勿使用 API 监听匿名函数，否则将无法关闭监听。

示例代码

```
<!-- .axml-->
<button size="default" onTap="offAppShowHanlder" type="primary">关闭监听到前台</button>
```

```
//.js
onLoad() {
    my.onAppShow(this.onAppShowHandler)
},
//监听切换到前台方法
onAppShowHandler() {
    console.log('前台了，到了')
},
//取消监听切换到前台方法
offAppShowHanlder() {
    my.offAppShow(this.onAppShowHandler)
},
apphide() {
    console.log('监听后台加载成功')
}
```

入参

入参为回调函数：

属性	类型	说明
回调函数	Function	小程序切前台事件的回调函数。

4.3.7. my.offError

取消监听小程序错误事件。

示例代码

```
App({
  onReady() {
    this.handleError = error => {
      // 小程序执行出错时
      console.log(error);
    }

    // 注意，error的类型为String
    my.onError(this.handleError);
  },
  onHide() {
    // 取消监听error事件
    my.offError(this.handleError);
  }
})
```

入参

入参为回调函数：

属性	类型	说明
回调函数	Function	小程序 JS 错误事件的回调函数。

4.3.8. my.onError

监听小程序错误事件。目前仅指 JS 执行错误。触发时机和参数与 App.onError 的一致。

使用说明：

- 使用 my.onError 监听到的报错，app.js 中的 onError 方法也会监听到。

- 使用 my.onError 监听页面报错，如果在多个页面开启监听没有关闭，则页面报错时会触发多个监听事件，建议在页面关闭时调用 my.offError 关闭监听。

示例代码

```
App({
  onReady() {
    // 注意，error的类型为String
    my.onError(function(error) {
      // 小程序执行出错时
      console.log(error);
    });
  }
})
```

JS 执行错误的示例代码如下：

```
onShow: function() {
    console.log(b);

    // 小程序执行出错时
    my.onError(function(error) {
      // 页面显示
      console.warn(error, '1212');
    });
  }
```

入参

入参为回调函数：

属性	类型	说明
回调函数	Function	小程序 JS 错误事件的回调函数。

回调参数

属性	类型	说明
error	String	error 错误描述。

4.4. 界面

4.4.1. 导航栏

4.4.1.1. my.getTitleColor

示例代码

```
<!-- get-title-color.axml -->
<view>
  <view class="page-section-demo">
    <text>目前导航栏的背景色:</text>
    <input type="text" disabled="{{true}}" value="{{titleColor.color}}"></input>
  </view>
  <view class="page-section-btns">
    <view onTap="getTitleColor">获取导航栏背景颜色
    </view>
  </view>
</view>
```

```
// get-title-color.js
Page({
  data: {
    titleColor: {},
  },
  getTitleColor() {
    my.getTitleColor({
      success: (res) => {
        this.setData({
          titleColor: res
        })
      }
    })
  }
});
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

success 回调函数

Object 类型，属性如下：

属性	类型	描述
color	HexColor	返回当前导航栏背景色，ARGB 格式的十六进制颜色值，如 #323239FF。

常见问题 FAQ

小程序右上角的 分享与收藏 可以设置颜色吗？

这是默认的，无法设置颜色。

4.4.1.2. my.hideBackHome

隐藏标题栏上的返回首页图标和右上角通用菜单中的返回首页功能。

说明：

- 用户启动小程序时，若进入的页面不是小程序首页，则会在左上角出现返回首页图标。
- 若用户继续进入下一级页面，则在右上角通用菜单中，会出现 **返回首页** 功能。

- 如果 app.json 中配置了 tabbar 跳转 pages/index/index 时，不会出现 **返回首页** 功能。

示例代码

```
//.js
Page({
  onReady() {
    if (my.canIUse('hideBackHome')) {
      my.hideBackHome();
    }
  },
});
```

```
//.js
onLoad(){
    my.reLaunch()({
    url:'../swiper/swiper'// 在页面中添加的非首页
  })

  setTimeout(() => {
    //5秒后隐藏返回首页按钮
    my.hideBackHome()
  }, 5000)
}
```

4.4.1.3. my.hideNavigationBarLoading

在当前页面隐藏导航条的加载动画。

示例代码

```
// API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.json
{
    "defaultTitle": "标题栏加载动画"
}
```

```
<!-- API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.axml-->
<view class="page">
  <view class="page-section">
    <button type="primary" onTap="showNavigationBarLoading">显示加载动画</button>
    <button onTap="hideNavigationBarLoading">隐藏加载动画</button>
  </view>
</view>
```

```
// API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.js
Page({
  showNavigationBarLoading() {
    my.showNavigationBarLoading()
  },
  hideNavigationBarLoading() {
    my.hideNavigationBarLoading()
  }
})
```

```
/* API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.acss */
button + button {
  margin-top: 20rpx;
}
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
success	Function	否	接口调用成功的回调函数。
fail	Function	否	接口调用失败的回调函数。
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.1.4. my.setNavigationBar

设置导航栏样式：导航栏标题、导航栏背景色、导航栏底部边框颜色、导航栏左上角 logo 图片。

使用说明

- 导航栏左上角 logo 图片支持 gif 格式，必须使用 https 图片链接。
- 若设置了导航栏背景色 backgroundColor，则导航栏底部边颜色 borderBottomColor 不会生效，默认会和 backgroundColor 颜色一样。
- 导航栏背景色不支持渐变色。

示例代码

```
// API-DEMO page/API/set-navigation-bar/set-navigation-bar.json
{
"defaultTitle": "设置页面导航栏"
}
```

```
<!-- API-DEMO page/API/set-navigation-bar/set-navigation-bar.axml-->
<view class="page">
<view class="page-description">设置导航栏 API</view>
<form onSubmit="setNavigationBar" style="align-self:stretch">
<view class="page-section">
<view class="page-section-demo">
<input class="page-body-form-value" type="text" placeholder="标题" name="title"></input>
<input class="page-body-form-value" type="text" placeholder="导航栏背景色" name="backgroundColor"></input>
<input class="page-body-form-value" type="text" placeholder="导航栏底部边框颜色" name="borderBottomColor"></input>
<input class="page-body-form-value" type="text" placeholder="导航栏图片地址" name="image"></input>
</view>
<view class="page-section-btns">
<button type="primary" size="mini" formType="submit">设置</button>
<button type="primary" size="mini" onTap="resetNavigationBar">重置</button>
</view>
</view>
</form>
<view class="tips">
tips:
<view class="item">1. image:图片链接地址，必须 https，请使用一张3x高清图。若设置了 image，则 title 参数失效</view>
<view class="item">2. backgroundColor: 导航栏背景色，支持 16 进制颜色值</view>
<view class="item">3. borderBottomColor: 导航栏底部边框颜色，支持16进制颜色值。若设置了 backgroundColor，borderBottomColo
r 会不生效，默认会和 backgroundColor 颜色一样。</view>
</view>
</view>
```

```
// API-DEMO page/API/set-navigation-bar/set-navigation-bar.js
Page({
setNavigationBar(e) {
var title = e.detail.value.title;
var backgroundColor = e.detail.value.backgroundColor;
var borderBottomColor = e.detail.value.borderBottomColor;
var image = e.detail.value.image;
console.log(title)
my.setNavigationBar({
title,
backgroundColor,
borderBottomColor,
image,
})
},
resetNavigationBar() {
my.setNavigationBar({
reset: true,
title: '重置导航栏样式',
});
}
})
```

```
/* API-DEMO page/API/set-navigation-bar/set-navigation-bar.acss */
.page-section-btns {
padding: 26rpx;
}
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
title	String	否	导航栏标题。
image	String	否	图片链接地址（支持 gif 格式图片），必须是https，请使用 iOS @3x 分辨率标准的高清图片。若设置了 image 则 title 参数失效。
backgroundColor	String	否	导航栏背景色，支持十六进制颜色值。
borderBottomColor	String	否	导航栏底部边框颜色，支持十六进制颜色值。若设置了 backgroundColor，则 borderBottomColor 不会生效，默认会和 backgroundColor 颜色一样。
reset	Boolean	否	是否重置导航栏为阿里云默认配色，默认为 false。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

常见问题 FAQ

小程序右上角的 分享与收藏 可以设置颜色吗？

这是默认的，无法设置颜色。

相关信息

iOS @3x 分辨率标准的更多信息，请参见 Image Size and Resolution。

4.4.1.5. my.showNavigationBarLoading

在当前页面显示导航条的加载动画。

示例代码

```
// API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.json
{
    "defaultTitle": "标题栏加载动画"
}
```

```
<!-- API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.axml-->
<view class="page">
  <view class="page-section">
    <button type="primary" onTap="showNavigationBarLoading">显示加载动画</button>
    <button onTap="hideNavigationBarLoading">隐藏加载动画</button>
  </view>
</view>
```

```
// API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.js
Page({
  showNavigationBarLoading() {
    my.showNavigationBarLoading()
  },
  hideNavigationBarLoading() {
    my.hideNavigationBarLoading()
  }
})
```

```
/* API-DEMO page/API/navigation-bar-loading/navigation-bar-loading.acss */
button + button {
  margin-top: 20rpx;
}
```

4.4.1.6. 导航栏 FAQ

小程序右上角的 分享与收藏 可以设置颜色吗？

这是默认的，无法设置颜色。

小程序胶囊按钮内的菜单页是否可以支持自定义修改？

目前小程序胶囊按钮内的菜单页暂不支持自定义修改。

导航栏的字体颜色可以自定义修改吗？

导航栏字体颜色无法自定义修改，但是可以通过修改背景颜色，自动调整变成黑色或白色的导航栏字体颜色。

4.4.2. TabBar

4.4.2.1. my.hideTabBar

注意： IDE 暂不支持模拟，请以真机调试效果为准。

隐藏标签页（tabbar），相关问题请参见 TabBar 常见问题。

示例代码

```
my.hideTabBar({
    animation: true
})
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	说明
animation	Boolean	否	是否需要动画效果，默认无。
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数。
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.2.2. my.hideTabBarRedDot

注意： IDE 暂不支持模拟，请以真机调试效果为准。

隐藏标签页（tabbar）某一项右上角的红点。

相关问题请参见 TabBar 常见问题。

示例代码

```
my.hideTabBarRedDot({
  index: 0
})
```

入参

Object 类型，属性如下：

属性	类型	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）。

4.4.2.3. my.removeTabBarBadge

注意： IDE 暂不支持模拟，请以真机调试效果为准。

移除标签页（tabbar） 某一项右上角的文本。

相关问题请参见 TabBar 常见问题。

示例代码

```
my.removeTabBarBadge({
  index: 0
})
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
index	Number	是	标签页的项数序号，从左边开始计数。
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）。

4.4.2.4. my.setTabBarBadge

注意： IDE 暂不支持模拟，请以真机调试效果为准。

为标签页（tabbar）某一项的右上角添加文本（数值型）。可用于设置消息条数的红点提醒。

相关问题请参见 TabBar 常见问题。

示例代码

```
my.setTabBarBadge({
  index: 0,
  text: '42'
})
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
index	Number	是	标签页的项数序号，从左边开始计数。
text	String	是	显示的文本，超限字数表现会有所区别，iOS超限字符长度为3，Android 为2。 超过字符长度则显示成前两个字符+“...”，例如：“999显示为“999”；” “9999”显示“99+”
success	Function	否	接口调用成功的回调函数。

fail	Function	否	接口调用失败的回调函数。
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.2.5. my.setTabBarItem

注意： IDE 暂不支持模拟，请以真机调试效果为准。

动态设置标签页（tabbar）某一项的内容。

相关问题请参见 TabBar 常见问题。

示例代码

```
my.setTabBarItem({
    index: 0,
    text: 'text',
    iconPath: '/image/iconPath',
    selectedIconPath: '/image/selectedIconPath'
})
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
index	Number	是	标签页的项数序号，从左边开始计数。
text	String	是	标签页按钮上的文字。
iconPath	String	是	图片路径，建议尺寸为 81px * 81px，支持 png/jpeg/jpg/gif 图片格式，支持网络图片。
selectedIconPath	String	是	选中时的图片路径，建议尺寸为 81px * 81px，支持 png/jpeg/jpg/gif 图片格式，支持网络图片。
success	Function	否	接口调用成功的回调函数。
fail	Function	否	接口调用失败的回调函数。
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.2.6. my.setTabBarStyle

注意： IDE 暂不支持模拟，请以真机调试效果为准。

注意2： iOS无效

动态设置标签页（tabbar）的整体样式，如文字颜色、标签背景色、标签边框颜色等。

相关问题请参见 TabBar 常见问题。

示例代码

```
my.setTabBarStyle({
    color: '#FF0000',
    selectedColor: '#00FF00',
    backgroundColor: '#0000FF',
    borderStyle: 'white'
})
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
color	HexColor	是	标签（tab）上的文字默认颜色
selectedColor	HexColor	是	标签（tab）上的文字选中时的颜色
backgroundColor	HexColor	是	标签（tab）的背景色
borderStyle	String	是	标签页（tabbar）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）。

4.4.2.7. my.showTabBar

注意： IDE 暂不支持模拟，请以真机调试效果为准。

显示标签页（tabbar）。

相关问题请参见 TabBar 常见问题。

示例代码

```
my.showTabBar({
    animation: true
})
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
animation	Boolean	否	是否需要动画效果，默认无。

success	Function	否	接口调用成功的回调函数。
fail	Function	否	接口调用失败的回调函数。
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.2.8. my.showTabBarRedDot

注意： IDE 暂不支持模拟，请以真机调试效果为准。

显示标签页（tabbar）某一项的右上角的红点。

相关问题请参见 TabBar 常见问题。

示例代码

```
my.showTabBarRedDot({
  index: 0
})
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
index	Number	是	标签页的项数序号，从左边开始计数。
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）

4.4.2.9. onTabItemTap

注意： IDE 暂不支持模拟，请以真机调试效果为准。

点击标签（tab）时触发，可用于监听 tabBar 点击事件。

相关问题请参见 TabBar 常见问题。

示例代码

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

4.4.2.10. TabBar 常见问题

如何监听 TabBar 点击事件？

在小程序页面中用 onTabItemTap 即可监听 TabBar 点击事件。

TabBar 页面不支持带参数跳转吗？

TabBar 页面目前不支持带参数跳转，建议跳转传参使用缓存或者全局变量。

切换 TabBar 时报错“Cannot read property 'getCurrentPages' of undefined”，如何处理？

tabBar 路径错误导致，请检查 tabBar 路径。

TabBar 为何在真机中不显示，但在 IDE 模拟器中可以正常显示？

若 TabBar 配置的 icon 路径是相对路径，将导致真机不显示。icon 图标需使用绝对路径，示例代码如下所示：

```
"tabBar": {
  "textColor": "#888888",
  "selectedColor": "#ff6a00",
  "backgroundColor": "#ffffff",
  "items": [
    {
      "pagePath": "pages/index/index",
      "icon": "https://gw.alicdn.com/tfs/TB1D5aEq8v0gK0jSZKbXXbK2FXa-96-96.png",
      "activeIcon": "https://gw.alicdn.com/tfs/TB1BvOEq9f2gK0jSZFPXXXsopXa-96-96.png",
      "name": "组件"
    },
    {
      "pagePath": "pages/api/api",
      "icon": "https://gw.alicdn.com/tfs/TB1VS5Hq4v1gK0jSZFFXXb0sXXa-96-96.png",
      "activeIcon": "https://gw.alicdn.com/tfs/TB1XnmHqYj1gK0jSZFOXXc7GpXa-96-96.png",
      "name": "API"
    }
  ]
}
```

TabBar 的 icon 图标是否支持 svg 格式？

不支持 svg 格式，只支持 png/jpeg/jpg/gif 图片格式。

在 IDE 中调试，为何 TabBar 切换时 onshow、onload 生命周期函数不执行？

TabBar 切换需要在真机上测试，IDE 中调试时不触发。

如何设置 Tab 的样式？

可以在 JSON 文件中直接设置样式（示例代码如下所示），或者调用 my.setTabBarStyle API 进行设置。

```
"tabBar": {
  "textColor": "#404040",
  "selectedColor": "#108ee9",
  "backgroundColor": "#F5F5F9"
}
```

跳转页面后，为何不显示 tabBar 呢？

通过页面跳转（my.navigateTo）或者页面重定向（my.redirectTo）所到达的页面，不会显示底部的 tab 栏。另外，tabBar 的第一个页面必须是首页。

tabBar

tabBar 配置项如下：

属性	类型	是否必填	描述
textColor	HexColor	否	文字颜色

selectedColor	HexColor	否	选中文字颜色
backgroundColor	HexColor	否	背景色
items	Array	是	每个 tab 配置

每个 item 配置：

属性	类型	是否必填	描述
pagePath	String	是	设置页面路径
name	String	是	名称
icon	String	否	平常图标路径
activeIcon	String	否	高亮图标路径

icon 图标推荐大小为 60×60 px 大小，系统会对传入的非推荐尺寸的图片进行非等比拉伸或缩放。

4.4.3. 路由

4.4.3.1. my.switchTab

跳转到指定标签页（tabbar）页面，并关闭其他所有非标签页页面。

如果小程序是一个多标签（tab）应用，即客户端窗口的底部栏可以切换页面，那么可以通过标签页配置项指定标签栏的表现形式，以及标签切换时显示的对应页面。

通过页面跳转（my.navigateTo）或者页面重定向（my.redirectTo）所到达的页面，即使是定义在标签页配置中的页面，也不会显示底部的标签栏。标签页的第一个页面必须是首页。

相关问题请参见 路由 FAQ 。

示例代码

```
// app.json
{
  "tabBar": {
    "items": [{
      "pagePath": "pages/home/index",
      "name": "首页"
    },{
      "pagePath": "pages/user/index",
      "name": "用户"
    }]
  }
}
```

```
//.js
my.switchTab({
  url: 'pages/home/index'
})
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
url	String	是	跳转的标签页的路径（需在 app.json 的 tabbar 字段定义的页面）。 **注意：**路径后不能带参数。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.4.3.2. my.reLaunch

关闭当前所有页面，跳转到应用内的某个指定页面。

相关问题请参见 路由 FAQ 。

示例代码

```
my.reLaunch({
  url: '/pages/index'
})
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
url	String	是	页面路径。如果页面不为 tabbar 页面则路径后可以带参数。 **参数规则：**路径与参数之间使用 `?` 分隔，参数键与参数值用 `=` 相连，不同参数必须用 `&` 分隔。 **示例：** `path?key1=value1&key2=value2`
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.4.3.3. my.redirectTo

关闭当前页面，跳转到应用内的某个指定页面。

示例代码

```
my.redirectTo({
  url: 'new_page?count=100' //路径可以使用相对路径或绝对路径的方式进行传递
})
```

以跳转至首页 index 页面为例：

- 使用绝对路径：url: /pages/index/index
- 使用相对路径：url: ../index/index

入参

Object 类型，属性如下：

属性	类型	必填	描述
url	String	是	需要跳转的应用内非 tabbar 的目标页面路径 ,路径后可以带参数。 **参数规则：** 路径与参数之间使用 `?` 分隔，参数键与参数值用 `=` 相连，不同参数必须用 `&` 分隔。 **示例：** `path?key1=value1&key2=value2`
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.4.3.4. my.navigateTo

从当前页面，跳转到应用内的某个指定页面。可使用 my.navigateBack 返回到原来页面。小程序中页面栈最多十层。

my.navigateTo 和 my.redirectTo 不允许跳转到选项卡（tabbar）页面；若需跳转到 tabbar 页面，请使用 my.switchTab。

相关问题请参见 路由 FAQ 。

示例代码

```
// API-DEMO page/API/navigator/navigator.json
{
    "defaultTitle": "页面跳转"
}
```

```
<!-- API-DEMO page/API/navigator/navigator.axml-->
<view class="page">
  <view class="page-section">
    <button type="primary" onTap="navigateTo">跳转新页面</button>
    <button type="primary" onTap="navigateBack">返回上一页</button>
    <button type="primary" onTap="redirectTo">在当前页面打开 - 获取用户信息</button>
    <button type="primary" onTap="switchTab">跳转 Tab - 组件</button>
    <view class="page-description">本Demo不具备小程序跳转功能，仅展示 API 的使用，具体接入请参考小程序官方文档 API 的小程序相互跳转部分。</view>
    <button type="primary" onTap="navigateToMiniProgram">跳转到小程序</button>
    <button type="primary" onTap="navigateBackMiniProgram">跳回小程序</button>
  </view>
</view>
```

```
// API-DEMO page/API/navigator/navigator.js
Page({
  navigateTo() {
    my.navigateTo({ url: '../get-user-info/get-user-info' })
  },
  navigateBack() {
    my.navigateBack()
  },
  redirectTo() {
    my.redirectTo({ url: '../get-user-info/get-user-info' })
  },
  navigateToMiniProgram() {
    if (my.canIUse('navigateToMiniProgram')) {
      my.navigateToMiniProgram({
        appId: '2017072607907880',
        extraData: {
          "data1": "test"
        },
        success: (res) => {
          console.log(JSON.stringify(res))
        },
        fail: (res) => {
          console.log(JSON.stringify(res))
        }
      });
    }
  },
  navigateBackMiniProgram() {
    if (my.canIUse('navigateBackMiniProgram')) {
      my.navigateBackMiniProgram({
        extraData: {
          "data1": "test"
        },
        success: (res) => {
          console.log(JSON.stringify(res))
        },
        fail: (res) => {
          console.log(JSON.stringify(res))
        }
      });
    }
  },
  switchTab() {
    my.switchTab({
        url: '/page/tabBar/component/index',
        success: () => {
          my.showToast({
            content: '成功',
            type: 'success',
            duration: 4000
          });
        }
      }
    );
  },
})
```

```
/* API-DEMO page/API/navigator/navigator.acss */
button + button {
  margin-top: 20rpx;
}
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
url	String	是	需要跳转的应用内非 tabbar 的目标页面路径 ,路径后可以带参数。 **参数规则：** 路径与参数之间使用 `?` 分隔，参数键与参数值用 `=` 相连，不同参数必须用 `&` 分隔。 **示例：** `path?key1=value1&key2=value2`
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.4.3.5. my.navigateBack

关闭当前页面，返回上一级或多级页面。可通过 `Page.getCurrentPages` 获取当前的页面栈信息，决定需要返回几层。

相关问题请参见 路由 FAQ 。

示例代码

```
// API-DEMO page/API/navigator/navigator.json
{
    "defaultTitle": "页面跳转"
}
```

```
<!-- API-DEMO page/API/navigator/navigator.axml-->
<view class="page">
  <view class="page-section">
    <button type="primary" onTap="navigateTo">跳转新页面</button>
    <button type="primary" onTap="navigateBack">返回上一页</button>
    <button type="primary" onTap="redirectTo">在当前页面打开 - 获取用户信息</button>
    <button type="primary" onTap="switchTab">跳转 Tab - 组件</button>
    <view class="page-description">本Demo不具备小程序跳转功能，仅展示 API 的使用，具体接入请参考小程序官方文档 API 的小程序相互跳转部分。</view>
    <button type="primary" onTap="navigateToMiniProgram">跳转到小程序</button>
    <button type="primary" onTap="navigateBackMiniProgram">跳回小程序</button>
  </view>
</view>
```

```
// API-DEMO page/API/navigator/navigator.js
Page({
  navigateTo() {
    my.navigateTo({ url: '../get-user-info/get-user-info' })
  },
  navigateBack() {
    my.navigateBack()
  },
  redirectTo() {
    my.redirectTo({ url: '../get-user-info/get-user-info' })
  },
  navigateToMiniProgram() {
    if (my.canIUse('navigateToMiniProgram')) {
      my.navigateToMiniProgram({
        appId: '2017072607907880',
        extraData: {
          "data1": "test"
        },
        success: (res) => {
          console.log(JSON.stringify(res))
        },
        fail: (res) => {
          console.log(JSON.stringify(res))
        }
      });
    }
  },
  navigateBackMiniProgram() {
    if (my.canIUse('navigateBackMiniProgram')) {
      my.navigateBackMiniProgram({
        extraData: {
          "data1": "test"
        },
        success: (res) => {
          console.log(JSON.stringify(res))
        },
        fail: (res) => {
          console.log(JSON.stringify(res))
        }
      });
    }
  },
  switchTab() {
    my.switchTab({
        url: '/page/tabBar/component/index',
        success: () => {
          my.showToast({
            content: '成功',
            type: 'success',
            duration: 4000
          });
        }
      }
    );
  },
})
```

```
/* API-DEMO page/API/navigator/navigator.acss */
button + button {
  margin-top: 20rpx;
}
```

入参

Object 类型，属性如下：

属性	类型	必填	默认值	描述
delta	Number	否	1	返回的页面数，如果 delta 大于现有打开的页面数，则返回到首页。

4.4.3.6. my.navigateToMiniProgram

跳转到其他小程序。

相关问题请参见 路由 FAQ 。

示例代码

```
my.navigateToMiniProgram({
  appId: '3000000002163743',
  extraData: {
  },
  success: (res) => {
    console.log(JSON.stringify(res))
  },
  fail: (res) => {
    console.log(JSON.stringify(res))
  }
});
```

4.4.3.7. my.navigateBackMiniProgram

跳转回上一个小程序

相关问题请参见 路由 FAQ 。

示例代码

```
my.navigateBackMiniProgram({
  extraData: {
    "data1": "test"
  },
  success: (res) => {
    console.log(JSON.stringify(res))
  },
  fail: (res) => {
    console.log(JSON.stringify(res))
  }
});
```

4.4.3.8. 路由 FAQ

使用 my.navigateTo 或者 my.redirectTo 跳转的页面为什么不显示底部的 tab 栏?

通过页面跳转（my.navigateTo）或者页面重定向（my.redirectTo）所到达的页面，即使它是定义在 tabBar 配置中的页面，也不会显示底部的 tab 栏。若要跳转到 tab 页面，请使用 my.switchTab 方法。

my.navigateTo 是否支持传参?

支持。

参数规则：路径与参数之间使用 `?` 分隔，参数键与参数值用 `=` 相连，不同参数必须用 `&` 分隔。

示例： `path?key1=value1&key2=value2`

使用 my.redirectTo 跳转页面，是否可以去掉左上角的返回按钮?

当页面栈深度为 1 时，使用 my.redirectTo 跳转页面的左上角不会有返回按钮，建议通过 getCurrentPages 方法判断页面栈峰值。或者用户可以直接使用 my.reLaunch 进行跳转，使用 my.reLaunch 进行跳转时，不允许跳转到 tabbar 页面。

小程序多次通过 my.navigateTo 跳转，尝试几次后为何再点击不会跳转了？

小程序规定最多不能超过 10 层页面栈，建议通过 getCurrentPages 方法判断页面栈峰值，超过后用重定向跳转页面。

小程序中的导航栏返回按钮是否能隐藏？

因为有层级的原因，所以会有返回按钮。可以调用 my.reLaunch 方法关闭当前所有页面去跳转到此页面，就没有这个返回按钮了。

4.4.4. 交互反馈

4.4.4.1. my.alert

警告框，可以设置警告框的标题、内容、按钮文字等，暂不支持设置图片等样式。

示例代码

```
// API-DEMO page/API/alert/alert.json
{
  "defaultTitle": "Alert"
}
```

```
<!-- API-DEMO page/API/alert/alert.axml-->
<view class="page">
  <view class="page-description">警告框 API</view>
  <view class="page-section">
    <view class="page-section-title">my.alert</view>
    <view class="page-section-demo">
      <button type="primary" onTap="alert">显示警告框</button>
    </view>
  </view>
</view>
```

```
// API-DEMO page/API/alert/alert.js
Page({
  alert() {
    my.alert({
      title: '亲',
      content: '您本月的账单已出',
      buttonText: '我知道了',
      success: () => {
        my.alert({
          title: '用户点击了「我知道了」',
        });
      }
    });
  },
})
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
title	String	否	警告框的标题。
content	String	否	警告框的内容。

buttonText	String	否	按钮文字，默认为 “确定”。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.4.4.2. my.confirm

用于提示的确认框，可以配置确认框的标题、内容、确认或取消按钮的文字等。

示例代码

```
// API-DEMO page/API/confirm/confirm.json
{
    "defaultTitle": "Confirm"
}
```

```
<!-- API-DEMO page/API/confirm/confirm.axml-->
<view class="page">
  <view class="page-description">确认框 API</view>
  <view class="page-section">
    <view class="page-section-title">my.confirm</view>
    <view class="page-section-demo">
      <button type="primary" onTap="comfirm">显示确认框</button>
    </view>
  </view>
</view>
```

```
// API-DEMO page/API/confirm/confirm.js
Page({
  comfirm() {
    my.confirm({
      title: '温馨提示',
      content: '您是否想查询快递单号：\n1234567890',
      confirmButtonText: '马上查询',
      cancelButtonText: '暂不需要',
      success: (result) => {
        my.alert({
          title: `${result.confirm}`,
        });
      },
    });
  },
});
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
title	String	否	confirm 框的标题。
content	String	否	confirm 框的内容。

confirmButtonText	String	否	确认按钮文字，默认为“确定”。
cancelButtonText	String	否	取消按钮文字，默认为“取消”。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

success 回调函数

入参为 Object 类型，属性如下：

名称	类型	描述
confirm	Boolean	点击确定按钮，返回 true；点击取消按钮，返回 false。

4.4.4.3. my.showActionSheet

显示操作菜单。

示例代码

```
// API-DEMO page/API/action-sheet/action-sheet.json
{
    "defaultTitle": "Action Sheet"
}
```

```
<!-- API-DEMO page/API/action-sheet/action-sheet.axml-->
<view class="page">
  <view class="page-description">操作菜单 API</view>
  <view class="page-section">
    <view class="page-section-title">my.showActionSheet</view>
    <view class="page-section-demo">
      <button type="primary" onTap="showActionSheet">显示操作菜单</button>
    </view>
  </view>
</view>
```

```
// API-DEMO page/API/action-sheet/action-sheet.js
Page({
  showActionSheet() {
    my.showActionSheet({
      title: '阿里云-ActionSheet',
      items: ['菜单一', '菜单二', '菜单三'],
      cancelButtonText: '取消好了',
      success: (res) => {
        const btn = res.index === -1 ? '取消' : '第' + res.index + '个';
        my.alert({
          title: `你点了${btn}按钮`
        });
      },
    });
  },
});
```

入参

Object 类型，属性如下：

属性	类型	必填	描述	最低基础库版本
title	String	否	菜单标题。	-
items	String Array	是	菜单按钮文字数组。	-
cancelButtonText	String	否	取消按钮文案。默认为'取消'。注：Android平台此字段无效，不会显示取消按钮。	-
destructiveBtnIndex	Number	否	（iOS特殊处理）指定按钮的索引号，从0开始，使用场景：需要删除或清除数据等类似场景，默认红色。	-
badges	Object Array	否	需飘红选项的数组，数组内部对象字段见下表。	
success	Function	否	调用成功的回调函数。	-
fail	Function	否	调用失败的回调函数。	-
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。	-

badges 属性

属性	类型	描述
index	Number	需要飘红的选项的索引，从 0 开始。
type	String	飘红类型，支持 none（无红点）/ point（纯红点） / num（数字红点）/ text（文案红点）/ more（...）。
text	String	自定义飘红文案：type 为 none / point / more 时，text 可不填； type为 num 时，text 为小数或 ≤ 0均不显示, ≥ 100 显示"..."。

4.4.4.4. my.prompt

弹出一个对话框，让用户在对话框内输入文本。

示例代码

```
my.prompt({
  title: '标题单行',
  message: '说明当前状态、提示用户解决方案，最好不要超过两行。',
  placeholder: '给朋友留言',
  okButtonText: '确定',
  cancelButtonText: '取消',
  success: (result) => {
    my.alert({
      title: JSON.stringify(result),
    });
  },
});
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
title	String	否	prompt 框标题。
message	String	否	prompt框文本，默认“请输入内容”。
placeholder	String	否	输入框内的提示文案。
align	String	否	message 对齐方式，可用值为： left 、center 、right。
okButtonText	String	否	确认按钮文字，默认值为“确定”。
cancelButtonText	String	否	确认按钮文字，默认值为 “取消”。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

success 回调函数

入参为 Object 类型，属性如下：

名称	类型	描述
ok	Boolean	点击 ok 返回 true，点击 cancel 返回 false。
inputValue	String	当 ok 返回 true 时，返回用户输入的内容。

4.4.4.5. my.showToast

显示一个弱提示，在到达设定的显示时间后，自动消失。

注意：在 my.showToast 之前调用 my.hideLoading，toast 被 my.hideLoading 覆盖，将不展示。

示例代码

```
// API-DEMO page/API/toast/toast.json
{
    "defaultTitle": "Toast"
}
```

```
<!-- API-DEMO page/API/toast/toast.axml-->
<view class="page">
  <view class="page-description">Toast API</view>
  <view class="page-section">
    <view class="page-section-title">my.showToast</view>
    <view class="page-section-btns">
      <view type="primary" onTap="showToastSuccess">显示 success 提示</view>
      <view type="primary" onTap="showToastFail">显示 fail 提示</view>
    </view>
    <view class="page-section-btns">
      <view type="primary" onTap="showToastException">显示 exception 提示</view>
      <view type="primary" onTap="showToastNone">显示 none 弱提示</view>
    </view>
  </view>

  <view class="page-section">
    <view class="page-section-title">my.hideToast</view>
    <view class="page-section-btns">
      <view onTap="hideToast">隐藏弱提示</view>
    </view>
  </view>
</view>undefined
```

```
// API-DEMO page/API/toast/toast.js
Page({
  showToastSuccess() {
    my.showToast({
      type: 'success',
      content: '操作成功',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  showToastFail() {
    my.showToast({
      type: 'fail',
      content: '操作失败',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  showToastException() {
    my.showToast({
      type: 'exception',
      content: '网络异常',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  showToastNone() {
    my.showToast({
      type: 'none',
      content: '提醒',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  hideToast() {
    my.hideToast()
  },
})undefined
```

入参

Object 类型，属性如下：

名称	类型	必填	描述
content	String	否	文字内容。

type	String	否	toast 类型，展示相应图标，默认 none，支持 success / fail / exception / none。其中 exception 类型必须传文字信息。
duration	Number	否	显示时长，单位为 ms，默认 2000。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

> *type 类型暂不支持*

4.4.4.6. my.hideToast

隐藏弱提示。

示例代码

```
// API-DEMO page/API/toast/toast.json
{
    "defaultTitle": "Toast"
}
```

```
<!-- API-DEMO page/API/toast/toast.axml-->
<view class="page">
  <view class="page-description">Toast API</view>
  <view class="page-section">
    <view class="page-section-title">my.showToast</view>
    <view class="page-section-btns">
      <view type="primary" onTap="showToastSuccess">显示 success 提示</view>
      <view type="primary" onTap="showToastFail">显示 fail 提示</view>
    </view>
    <view class="page-section-btns">
      <view type="primary" onTap="showToastException">显示 exception 提示</view>
      <view type="primary" onTap="showToastNone">显示 none 弱提示</view>
    </view>
  </view>

  <view class="page-section">
    <view class="page-section-title">my.hideToast</view>
    <view class="page-section-btns">
      <view onTap="hideToast">隐藏弱提示</view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/API/toast/toast.js
Page({
  showToastSuccess() {
    my.showToast({
      type: 'success',
      content: '操作成功',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  showToastFail() {
    my.showToast({
      type: 'fail',
      content: '操作失败',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  showToastException() {
    my.showToast({
      type: 'exception',
      content: '网络异常',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  showToastNone() {
    my.showToast({
      type: 'none',
      content: '提醒',
      duration: 3000,
      success: () => {
        my.alert({
          title: 'toast 消失了',
        });
      },
    });
  },
  hideToast() {
    my.hideToast()
  },
})
```

入参

名称	类型	必填	说明
success	function	否	接口调用成功的回调函数。
fail	function	否	接口调用失败的回调函数。

complete	function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.4.7. my.showLoading

显示加载提示的过渡效果，可与 my.hideLoading 配合使用。

示例代码

```
// API-DEMO page/API/loading/loading.json
{
    "defaultTitle": "加载提示"
}
```

```
<!-- API-DEMO page/API/loading/loading.axml-->
<view class="page">
  <view class="page-section">
    <view class="page-section-title">
      显示 loading 后，会覆盖整个h5页面，页面元素不能交互。
    </view>
    <view class="page-section-btns">
      <view onTap="showLoading">显示加载提示</view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/API/loading/loading.js
Page({
  showLoading() {
    my.showLoading({
      content: '加载中...',
      delay: '1000',
    });
    setTimeout(() => {
      my.hideLoading();
    }, 5000);
  },
});
```

```
/* API-DEMO page/API/loading/loading.acss */
.tips{
  margin-left: 10px;
  margin-top: 20px;
  color: red;
  font-size: 12px;
}
.tips .item {
  margin: 5px 0;
  color: #888888;
  line-height: 14px;
}
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
content	String	否	提示中的文字内容。

delay	Number	否	延迟显示，单位 ms，默认 0。如果在此时间之前调用了 my.hideLoading 则不会显示。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.4.4.8. my.hideLoading

隐藏加载提示的过渡效果，可与 my.showLoading 配合使用。

注意：在 my.showToast 之前调用 my.hideLoading，toast 被 my.hideLoading 覆盖，将不展示。

示例代码

```
my.hideLoading();

Page({
  onLoad() {
    my.showLoading();
    const that = this;
    setTimeout(() => {
      my.hideLoading({
        page: that,  // 防止执行时已经切换到其它页面，page 指向不准确
      });
    }, 4000);
  }
})
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
page	Object	否	具体指当前 page 实例，某些场景下，需要指明在哪个 page 执行 hideLoading。

4.4.5. 下拉刷新

4.4.5.1. my.startPullDownRefresh

主动开启下拉刷新。代码执行后触发下拉刷新动画，效果与用户手动下拉刷新一致（会触发 onPullDownRefresh 监听方法）。

注意：IDE 暂不支持模拟，请以真机调试效果为准。

注意2：需要配置 "pullRefresh": true 才可生效

示例代码

```
//.js
my.startPullDownRefresh()
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
success	Function	否	接口调用成功的回调函数。
fail	Function	否	接口调用失败的回调函数。
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.5.2. my.stopPullDownRefresh

停止当前页面的下拉刷新。

注意：IDE 暂不支持模拟，请以真机调试效果为准。

注意2：需要配置 "pullRefresh": true 才可生效

示例代码

```
// API-DEMO page/API/pull-down-refresh/pull-down-refresh.json
{
    "defaultTitle": "下拉刷新",
    "pullRefresh": true
}
```

```
<!-- API-DEMO page/API/pull-down-refresh/pull-down-refresh.axml-->
<view class="page">
  <view class="page-section">
    <view class="page-section-title">下滑页面即可刷新</view>
    <view class="page-section-btns">
      <view type="primary" onTap="stopPullDownRefresh">停止刷新</view>
    </view>
  </view>
</view>undefined
```

```
// API-DEMO page/API/pull-down-refresh/pull-down-refresh.js
Page({
  onPullDownRefresh() {
    console.log('onPullDownRefresh', new Date());
  },
  stopPullDownRefresh() {
    my.stopPullDownRefresh({
      complete(res) {
        console.log(res, new Date())
      }
    })
  }
});
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
success	Function	否	接口调用成功的回调函数.
fail	Function	否	接口调用失败的回调函数.
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）.

4.4.5.3. onPullDownRefresh

在 Page 中自定义 onPullDownRefresh 函数，可以监听该页面用户的下拉刷新事件。

- 在页面对应的 .json 配置文件中配置 `"pullRefresh": true` 选项，方可开启下拉刷新事件。
- 数据刷新处理完毕，调用 `my.stopPullDownRefresh` 可停止当前页面的下拉刷新。

注意： window 中 `pullRefresh` 、 `allowsBounceVertical` 属性的设置会影响下拉刷新的使用。

示例代码

```
// API-DEMO page/API/pull-down-refresh/pull-down-refresh.json
{
    "defaultTitle": "下拉刷新",
    "pullRefresh": true
}
```

```
<!-- API-DEMO page/API/pull-down-refresh/pull-down-refresh.axml-->
<view class="page">
  <view class="page-section">
    <view class="page-section-title">下滑页面即可刷新</view>
    <view class="page-section-btns">
      <view type="primary" onTap="stopPullDownRefresh">停止刷新</view>
    </view>
  </view>
</view>undefined
```

```
// API-DEMO page/API/pull-down-refresh/pull-down-refresh.js
Page({
  onPullDownRefresh() {
    console.log('onPullDownRefresh', new Date());
  },
  stopPullDownRefresh() {
    my.stopPullDownRefresh({
      complete(res) {
        console.log(res, new Date())
      }
    })
  }
});undefined
```

入参

属性	类型	必填	描述
pullRefresh	Boolean	否	是否允许下拉刷新。默认 true。 **说明**：下拉刷新生效的前提是allowsBounceVertical 值为 `YES` 。
allowsBounceVertical	String	否	页面是否支持纵向拽拉超出实际内容。默认 YES，支持 `YES` / `NO` 。

4.4.6. 选择日期

4.4.6.1. my.datePicker

打开日期选择列表。

示例代码

```
// API-DEMO page/API/date-picker/date-picker.json
{
    "defaultTitle": "Date Picker"
}
```

```
<!-- API-DEMO page/API/date-picker/date-picker.axml -->
<view class="page">
  <view class="page-description">选择日期 API</view>
  <view class="page-section">
    <view class="page-section-title">my.datePicker</view>
    <view class="page-section-demo">
      <button class="page-body-button" type="primary" onTap="datePicker">选择日期-1</button>
      <button class="page-body-button" type="primary" onTap="datePickerHMS">选择日期-2</button>
      <button class="page-body-button" type="primary" onTap="datePickerYMDHMS">选择日期-3</button>
    </view>
  </view>
</view>undefined
```

```
// API-DEMO page/API/date-picker/date-picker.js
Page({
  datePicker() {
    my.datePicker({
      currentDate: '2016-10-10',
      startDate: '2016-10-9',
      endDate: '2017-10-9',
      success: (res) => {
        my.alert({
          title: 'datePicker response: ' + JSON.stringify(res)
        });
      },
    });
  },
  datePickerHMS() {
    my.datePicker({
      format: 'HH:mm',
      currentDate: '12:12',
      startDate: '11:11',
      endDate: '13:13',
      success: (res) => {
        my.alert({
          title: 'datePicker response: ' + JSON.stringify(res)
        });
      },
    });
  },
  datePickerYMDHMS() {
    my.datePicker({
      format: 'yyyy-MM-dd HH:mm',
      currentDate: '2012-01-09 11:11',
      startDate: '2012-01-01 11:11',
      endDate: '2012-01-10 11:11',
      success: (res) => {
        my.alert({
          title: 'datePicker response: ' + JSON.stringify(res)
        });
      },
    });
  },
});undefined
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
format	String	否	返回的日期格式。
currentDate	String	否	初始选择的日期时间，默认当前时间。
startDate	String	否	最小日期时间。
endDate	String	否	最大日期时间。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

返回的日期格式有：

- yyyy-MM-dd（默认）
- HH:mm
- yyyy-MM-dd HH:mm
- yyyy-MM，可用 **canIUse('datePicker.object.format.yyyy-MM')** 判断
- yyyy，可用 **canIUse('datePicker.object.format.yyyy')** 判断

success 回调函数

入参为 Object 类型，属性如下：

属性	类型	描述
date	String	选择的日期

结果码

结果码	描述
11	用户取消操作

4.4.7. 动画

4.4.7.1. my.createAnimation

创建动画实例 animation。调用实例的方法来描述动画，最后通过动画实例的 `export` 方法将动画数据导出并传递给组件的 `animation` 属性。

说明：调用 `export` 方法后，之前的动画操作将会被清除。

示例代码

```
// API-DEMO page/API/animation/animation.json
{
    "defaultTitle": "Animation"
}
```

```
<!-- API-DEMO page/API/animation/animation.axml-->
<view class="page">
  <view class="page-description">动画 API</view>
  <view class="page-section">
    <view class="page-section-title">my.createAnimation</view>
    <view class="page-section-demo">
      <view class="animation-element" animation="{{animation}}"></view>
    </view>
    <view class="page-section-btns">
      <view type="primary" onTap="rotate">旋转</view>
      <view type="primary" onTap="scale"> 缩放</view>
      <view type="primary" onTap="translate">移动</view>
    </view>
    <view class="page-section-btns">
      <view type="primary" onTap="skew">倾斜</view>
      <view type="primary" onTap="rotateAndScale">旋转并缩放</view>
      <view type="primary" onTap="rotateThenScale">旋转后缩放</view>
    </view>
    <view class="page-section-btns">
      <view type="primary" onTap="all">同时展示全部</view>
      <view type="primary" onTap="allInQueue">顺序展示全部</view>
      <view type="primary" onTap="reset">还原</view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/API/animation/animation.js
Page({
  onReady() {
    this.animation = my.createAnimation()
  },
  rotate() {
    this.animation.rotate(Math.random() * 720 - 360).step()
    this.setData({ animation: this.animation.export() })
  },
  scale() {
    this.animation.scale(Math.random() * 2).step()
    this.setData({ animation: this.animation.export() })
  },
  translate() {
    this.animation.translate(Math.random() * 100 - 50, Math.random() * 100 - 50).step()
    this.setData({ animation: this.animation.export() })
  },
  skew() {
    this.animation.skew(Math.random() * 90, Math.random() * 90).step()
    this.setData({ animation: this.animation.export() })
  },
  rotateAndScale() {
    this.animation.rotate(Math.random() * 720 - 360)
      .scale(Math.random() * 2)
      .step()
    this.setData({ animation: this.animation.export() })
  },
  rotateThenScale() {
    this.animation.rotate(Math.random() * 720 - 360).step()
      .scale(Math.random() * 2).step()
    this.setData({ animation: this.animation.export() })
  },
  all() {
    this.animation.rotate(Math.random() * 720 - 360)
      .scale(Math.random() * 2)
      .translate(Math.random() * 100 - 50, Math.random() * 100 - 50)
      .skew(Math.random() * 90, Math.random() * 90)
      .step()
    this.setData({ animation: this.animation.export() })
  },
  allInQueue() {
    this.animation.rotate(Math.random() * 720 - 360).step()
      .scale(Math.random() * 2).step()
      .translate(Math.random() * 100 - 50, Math.random() * 100 - 50).step()
      .skew(Math.random() * 90, Math.random() * 90).step()
    this.setData({ animation: this.animation.export() })
  },
  reset() {
    this.animation.rotate3d(0, 0, 0, 0)
      .rotateX(0)
      .rotateY(0)
      .rotateZ(0)
      .scale(1)
      .translate(0, 0)
      .skew(0, 0)
      .step({ duration: 0 })
    this.setData({ animation: this.animation.export() })
  }
})
```

入参

Object 类型，属性如下：

参数	类型	必填	说明

duration	Integer	否	动画的持续时间，单位 ms，默认值 400。
timeFunction	String	否	定义动画的效果，默认值“linear”，有效值：linear、ease，ease-in、ease-in-out、ease-out、step-start、step-end。
delay	Integer	否	动画延迟时间，单位 ms，默认值 0。
transformOrigin	String	否	设置 transform-origin，默认值“50% 50% 0”。

示例代码

//.jsconst=.createAnimation({:"top right",:3000,:"ease-in-out",:100,})

```
animation  my
 transformOrigin
 duration
 timeFunction
 delay
```

animation

动画实例可以调用以下方法来描述动画，调用结束后会返回实例本身，支持链式调用的写法。view 的 animation 属性初始化为 `{}` 时，在基础库 1.11.0（不包含 1.11.0）及以下版本会报错，建议初始化为 `null` 。

样式

方法	参数	说明
opacity	value	透明度，参数范围 0~1。
backgroundColor	color	颜色值。
width	length	设置宽度：长度值，单位为 px，例如：300 px。
height	length	设置高度：长度值，单位为 px，例如：300 px。
top	length	设置 top 值：长度值，单位为 px，例如：300 px。
left	length	设置 left 值：长度值，单位为 px，例如：300 px。
bottom	length	设置 bottom 值：长度值，单位为 px，例如：300 px。
right	length	设置 right 值：长度值，单位为 px，例如：300 px。

旋转

方法	参数	说明
rotate	deg	deg 范围 -180 ~ 180，从原点顺时针旋转一个 deg 角度。

rotateX	deg	deg 范围 -180 ~ 180，在 X 轴旋转一个 deg 角度。
rotateY	deg	deg 范围 -180 ~ 180，在 Y 轴旋转一个 deg 角度。
rotateZ	deg	deg 范围 -180 ~ 180，在 Z 轴旋转一个 deg 角度。
rotate3d	(x, y , z, deg)	同 transform-function rotate3d。

缩放

方法	参数	说明
scale	sx,[sy]	只有一个参数时，表示在 X 轴、Y 轴同时缩放 sx 倍； 两个参数时表示在 X 轴缩放 sx 倍，在 Y 轴缩放 sy 倍。
scaleX	sx	在 X 轴缩放 sx 倍。
scaleY	sy	在 Y 轴缩放 sy 倍。
scaleZ	sz	在 Z 轴缩放 sy 倍。
scale3d	(sx,sy,sz)	在 X 轴缩放 sx 倍，在 Y 轴缩放sy 倍，在 Z 轴缩放 sz 倍。

偏移

方法	参数	说明
translate	tx,[ty]	只有一个参数时，表示在 X 轴偏移 tx；两个参数时，表示在 X 轴偏移 tx，在 Y 轴偏移 ty，单位均为 px。
translateX	tx	在 X 轴偏移 tx，单位 px。
translateY	ty	在 Y 轴偏移 tx，单位 px。
translateZ	tz	在 Z 轴偏移 tx，单位 px。
translate3d	(tx,ty,tz)	在 X 轴偏移 tx，在 Y 轴偏移 ty，在Z轴偏移 tz，单位 px。

倾斜

方法	参数	说明
skew	ax,[ay]	参数范围 -180 ~ 180。只有一个参数时，Y 轴坐标不变，X 轴坐标延顺时针倾斜 ax 度；两个参数时，分别在 X 轴倾斜 ax 度，在 Y 轴倾斜 ay 度。
skewX	ax	参数范围 -180 ~ 180。Y 轴坐标不变，X 轴坐标延顺时针倾斜 ax。度。

skewY	ay	参数范围 -180~180。X 轴坐标不变，Y 轴坐标延顺时针倾斜 ay 度。

矩阵变形

方法	参数	说明
matrix	(a,b,c,d,tx,ty)	同 transform-function。
matrix3d		同 transform-function matrix3d。

动画队列

- 调用动画操作方法后，需调用 `step()` 用于表示一组动画完成，在一组动画中可以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，当一组动画完成后才会进行下一组动画。
- `step()` 可以传入一个跟 `my.createAnimation()` 一样的配置参数用于指定当前组动画的配置。

4.4.8. 画布

4.4.8.1. my.createCanvasContext

创建 canvas 绘图上下文。该绘图上下文只作用于对应 `canvasId` 的 `<canvas/>` 。

参数	类型	说明
canvasId	String	定义在 `<canvas/>` 上的 id

返回值

CanvasContext

4.4.8.2. CanvasContext

4.4.8.2.1. CanvasContext 概览

名称	描述
CanvasContext.addColorStop	创建一个圆形的渐变色。
CanvasContext.arc	画一条弧线。
CanvasContext.beginPath	开始创建一个路径，需调用 `fill` 或 `stroke` 才会使用路径进行填充或描边。
CanvasContext.bezierCurveTo	创建三次方贝塞尔曲线路径。
CanvasContext.clearRect	清除画布上指定矩形区域的内容。
CanvasContext.clip	将当前创建的路径设置为剪切路径。

CanvasContext.closePath	闭合一个路径。
CanvasContext.createCircularGradient	创建一个圆形渐变。起点在圆心，终点在圆环。
CanvasContext.createLinearGradient	创建一个线性渐变。
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。
CanvasContext.drawImage	绘制图像，图像保持原始尺寸。
CanvasContext.fill	对当前路径中的内容进行填充。
CanvasContext.fillRect	填充矩形。
CanvasContext.fillText	在画布上绘制被填充的文本。
CanvasContext.getImageData	获取 canvas 区域隐含的像素数据。
CanvasContext.lineTo	新增一个点，然后创建一条从上一个指定点到目标点的线。
CanvasContext.measureText	测量文本尺寸信息，目前仅返回文本宽度。同步接口。
CanvasContext.moveTo	把路径移动到画布的指定点，不创建线条。
CanvasContext.quadraticCurveTo	创建二次贝塞尔曲线路径。
CanvasContext-rect	创建矩形
CanvasContext-restore	恢复之前保存的绘图上下文。
CanvasContext-rotate	以原点为中心，原点可以用 CanvasContext.translate 方法修改。顺时针旋转当前坐标轴。多次调用 `rotate` 旋转的角度会叠加。
CanvasContext-save	保存当前的绘图上下文。
CanvasContext-scale	调用 `scale` 方法后创建的路径的横纵坐标会被缩放。 多次调用 `scale` ，倍数会相乘。
CanvasContext-setFillStyle	设置填充色。
CanvasContext-setFontSize	设置字体大小。
CanvasContext-setGlobalAlpha	设置全局透明度。

CanvasContext-setLineCap	设置线条的端点样式。
CanvasContext-setLineDash	设置虚线的样式。
CanvasContext-setLineJoin	设置线条的交点样式。
CanvasContext-setLineWidth	设置线条的宽度。
CanvasContext-setMiterLimit	设置最大斜接长度。
CanvasContext.setShadow	设置阴影样式。
CanvasContext.setStrokeStyle	设置描边样式。
CanvasContext.setTextAlign	设置文本对齐方式。
CanvasContext.setTextBaseline	设置文本基线。
CanvasContext.setTransform	使用单位矩阵重新设置（覆盖）当前的变换并调用变换的方法，此变换由方法的变量进行描述。
CanvasContext.stroke	描边当前路径。默认 `black` 。
CanvasContext.strokeRect	描边矩形。
CanvasContext.toDataURL	获取画布指定区域的 data URL 数据。
CanvasContext.toTempFilePath	把当前画布的内容导出生成图片，并返回文件路径。
CanvasContext.transform	使用矩阵多次叠加当前变换的方法，矩阵由方法的参数进行描述。可以缩放、旋转、移动和倾斜上下文。
CanvasContext.translate	平移当前坐标系的原点，默认的坐标系原点为页面左上角。

4.4.8.2.2. CanvasContext.addColorStop

添加颜色的渐变点。

入参

Object 类型，属性如下：

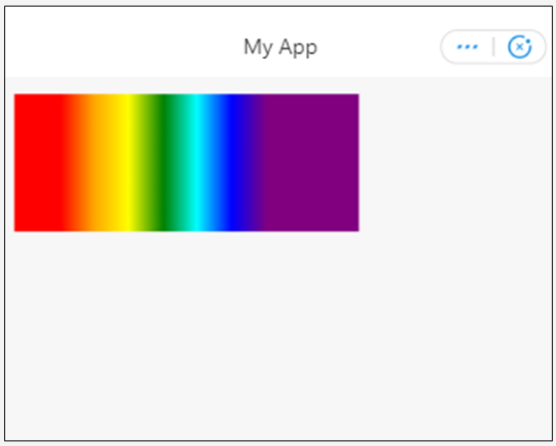
属性	类型	定义
stop	Number	表示渐变中开始与结束之间的位置，范围 0-1。

color	Color	渐变点的颜色。

示例代码

```
//.js
const ctx = my.createCanvasContext('myCanvas')

// 创建圆形梯度色
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')

// 填充梯度色
ctx.setFillStyle(grd)
ctx.fillRect(10, 10, 150, 80)
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.3. CanvasContext.arc

画一条弧线。

创建一个圆，可以用 `arc()` 方法指定起始弧度为 0，终止弧度为 `2 * Math.PI` 。用 `stroke()` 或者 `fill()` 方法在 canvas 中画弧线。

入参

Object 类型，属性如下：

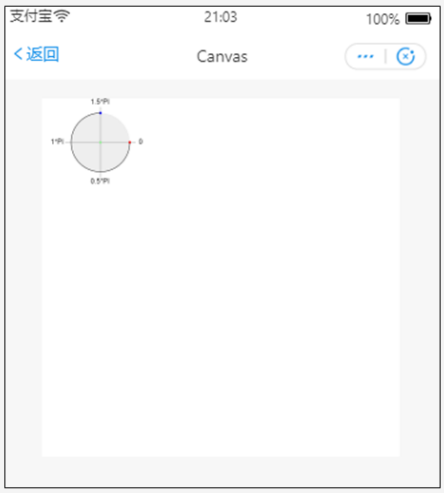
属性	类型	说明
x	Number	圆 x 坐标。

y	Number	圆 y 坐标。
r	Number	圆半径。
sAngle	Number	起始弧度，单位弧度（在3点钟方向）。
eAngle	Number	终止弧度。
counterclockwise	Boolean	可选，指定弧度的方向是逆时针还是顺时针，默认为 false。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

const ctx = my.createCanvasContext('myCanvas')

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)

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()

ctx.beginPath()
ctx.arc(100, 75, 50, 0, 1.5 * Math.PI)
ctx.setStrokeStyle('#333333')
ctx.stroke()

ctx.draw()
```

显示效果如下图所示：



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)

4.4.8.2.4. CanvasContext.beginPath

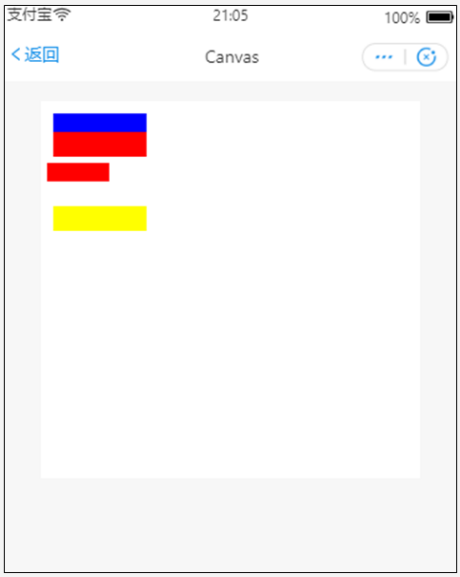
开始创建一个路径，需要调用 `fill` 或者 `stroke` 才会使用路径进行填充或描边。

在创建路径最开始时，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

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.rect(20, 20, 150, 50)
ctx.setFillStyle('blue')
ctx.fill();

ctx.beginPath();
ctx.rect(20, 50, 150, 40)

ctx.setFillStyle('yellow')
ctx.fillRect(20, 170, 150, 40)

ctx.rect(10, 100, 100, 30)

ctx.setFillStyle('red')
ctx.fill()
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.5. CanvasContext.bezierCurveTo

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。

入参

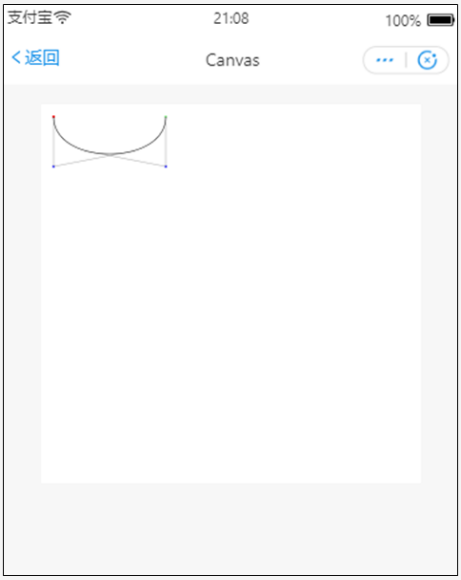
Object 类型，属性如下：

属性	类型	说明
cp1x	Number	第一个贝塞尔控制点 x 坐标。
cp1y	Number	第一个贝塞尔控制点 y 坐标。
cp2x	Number	第二个贝塞尔控制点 x 坐标。
cp2y	Number	第二个贝塞尔控制点 y 坐标。
x	Number	结束点 x 坐标。
y	Number	结束点 y 坐标。

示例代码

```
//.js
const ctx = my.createCanvasContext('myCanvas')

// 画点
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)

// 画参考线
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()

// 画二次曲线
ctx.beginPath()
ctx.moveTo(20, 20)
ctx.bezierCurveTo(20, 100, 200, 100, 200, 20)
ctx.setStrokeStyle('black')
ctx.stroke()

ctx.draw()
```

显示效果如下图所示：



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)

4.4.8.2.6. CanvasContext.clearRect

清除画布上在该矩形区域内的内容。

入参

Object 类型，属性如下：

属性	类型	描述
x	Number	矩形左上角的 x 坐标。
y	Number	矩形左上角的 y 坐标。
width	Number	矩形宽度。
height	Number	矩形高度。

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，可以对 canvas 加一层背景色。

```
//.axml
<canvas id="awesomeCanvas" style="border: 1px solid; background: red;"/>
```

```
// .js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.setFillStyle('blue')
ctx.fillRect(250, 10, 250, 200)
ctx.setFillStyle('yellow')
ctx.fillRect(0, 0, 150, 200)
ctx.clearRect(10, 10, 150, 75)
ctx.draw()
```

4.4.8.2.7. CanvasContext.clip

将当前创建的路径设置为当前剪切路径。

扫码体验

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
my.downloadFile({
  url: 'https://gw.alipayobjects.com/zos/skylark-tools/public/files/dda114e320567e1d304790287d75a029.png',
  success: function(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()
  }
})
```

4.4.8.2.8. CanvasContext.closePath

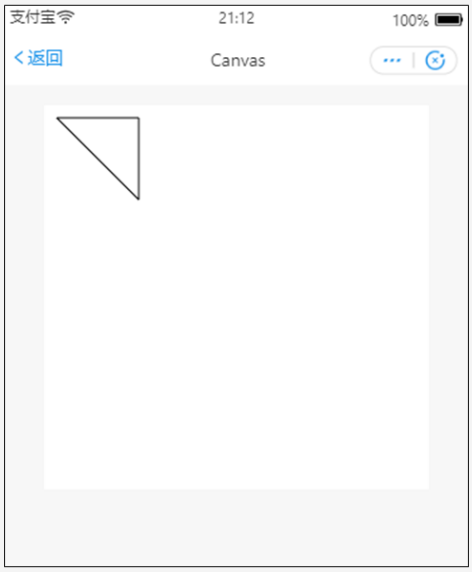
关闭一个路径。关闭路径会连接起点和终点。如果关闭路径后没有调用 `fill()` 或者 `stroke()` 并开启了新的路径，那之前的路径将不会被渲染。

示例代码

示例代码 1

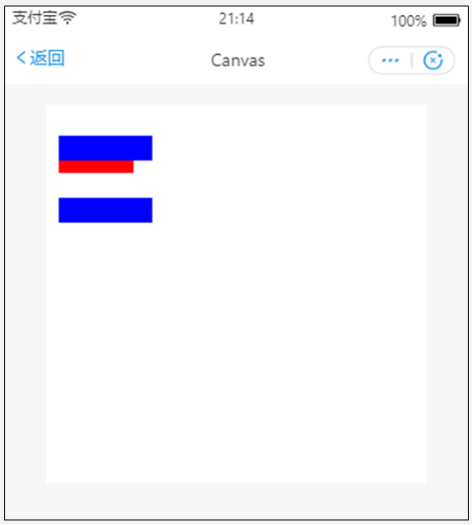
```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.moveTo(20, 20)
ctx.lineTo(150, 20)
ctx.lineTo(150, 150)
ctx.closePath()
ctx.stroke()
ctx.draw()
```

显示效果如下图所示：




```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.rect(20, 20, 150, 50)
ctx.closePath()

ctx.beginPath()
ctx.rect(20, 50, 150, 40)

ctx.setFillStyle('red')
ctx.fillRect(20, 80, 120, 30)

ctx.rect(20, 150, 150, 40)

ctx.setFillStyle('blue')
ctx.fill()
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.9. CanvasContext.createCircularGradient

创建一个圆形的渐变色。起点在圆心，终点在圆环。需要使用 `addColorStop()` 来指定渐变点，至少需要两个。

入参

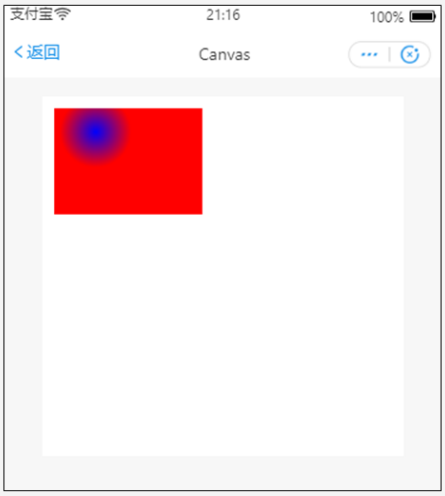
Object 类型，属性如下：

属性	类型	描述
x	Number	圆心 x 坐标。
y	Number	圆心 y 坐标。
r	Number	圆半径。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

const grd = ctx.createCircularGradient(90, 60, 60)
grd.addColorStop(0, 'blue')
grd.addColorStop(1, 'red')

ctx.setFillStyle(grd)
ctx.fillRect(20, 20, 250, 180)
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.10. CanvasContext.createLinearGradient

创建一个线性的渐变色。需要使用 `addColorStop()` 来指定渐变点，至少需要两个。

入参

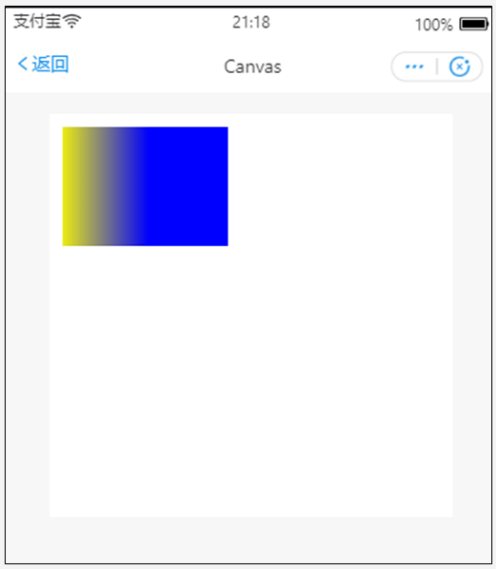
Object 类型，属性如下：

属性	类型	描述
x0	Number	起点 x 坐标。
y0	Number	起点 y 坐标。
x1	Number	终点 x 坐标。
y1	Number	终点 y 坐标。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

const grd = ctx.createLinearGradient(10, 10, 150, 10)
grd.addColorStop(0, 'yellow')
grd.addColorStop(1, 'blue')

ctx.setFillStyle(grd)
ctx.fillRect(20, 20, 250, 180)
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.11. 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。绘图上下文需要由 `my.createCanvasContext(canvasId)` 来创建。

入参

Object 类型，属性如下：

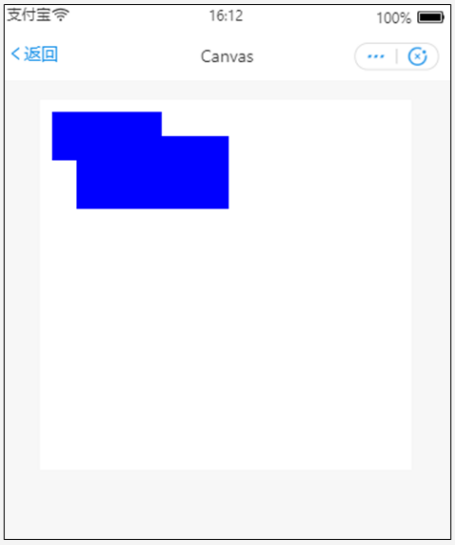
属性	类型	必填	默认值	说明
reserve	Boolean	否	false	本次绘制是否接着上一次绘制，即 reserve 参数为 false ，则在本次调用 drawCanvas 绘制之前 native 层应先清空画布再继续绘制；若 reserver 参数为 true 时，则保留当前画布上的内容，本次调用 drawCanvas 绘制的内容覆盖在上面。
callback	Function	是	-	绘制完成后执行的回调函数。

示例代码

示例代码 1

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.setFillStyle('blue')
ctx.fillRect(20, 20, 180, 80)
ctx.draw()
ctx.fillRect(60, 60, 250, 120)
// 保留上一次的绘制结果
ctx.draw(true)
```

显示效果如下图所示：

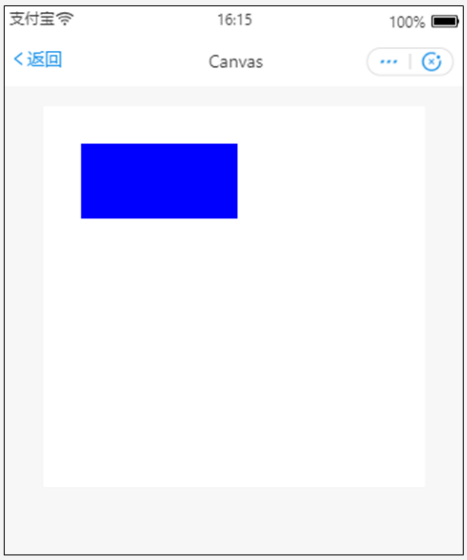


示例代码 2

```
//.jsconst=.createCanvasContext('awesomeCanvas')
ctx.setFillStyle('blue')
ctx.fillRect(20, 20, 180, 80)
ctx.draw()
ctx.fillRect(60, 60, 250, 120)
// 不保留上一次的绘制结果
ctx.draw(false)
```

```
ctx  my
```

显示效果如下图所示：



4.4.8.2.12. CanvasContext.drawImage

绘制图像，图像保持原始尺寸。

扫码体验

入参

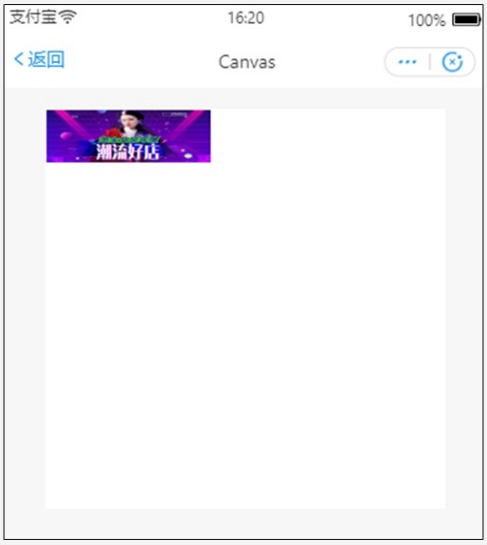
Object 类型，属性如下：

属性	类型	说明
imageResource	String	图片资源，只支持线上 cdn 地址或离线包地址，线上 cdn 需返回头 `Access-Control-Allow-Origin: *` 。
x	Number	图像左上角 x 坐标。
y	Number	图像左上角 y 坐标 。
width	Number	图像宽度。
height	Number	图像高度。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.drawImage('https://img.alicdn.com/tfs/TB1GvVMj2BNTKJjy0FdXXcPpVXa-520-280.jpg', 2, 2, 250, 80)
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.13. CanvasContext.fill

对当前路径中的内容进行填充。默认的填充色为黑色。

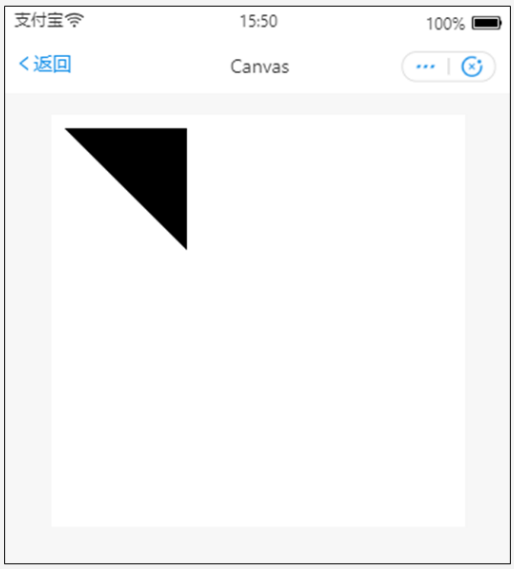
- 如果当前路径没有闭合，`fill()` 方法会将起点和终点进行连接，然后填充，详情见示例代码 1。
- `fill()` 填充的的路径从 `beginPath()` 开始计算，但不包含 `fillRect()` 包含，详情见示例代码 2。

示例代码

示例代码 1

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.moveTo(20, 20)
ctx.lineTo(200, 20)
ctx.lineTo(200, 200)
ctx.fill()
ctx.draw()
```

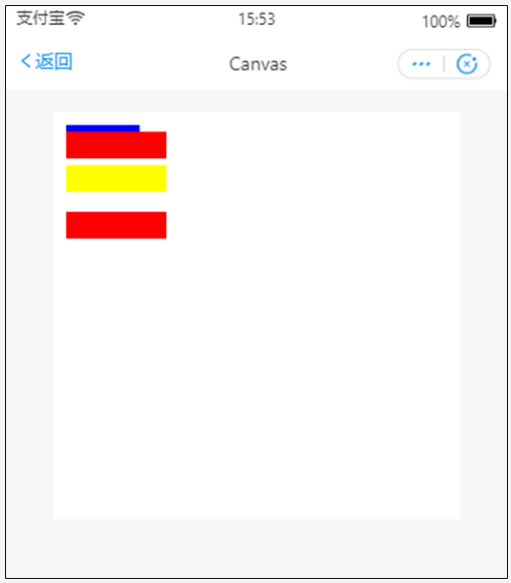
显示效果如下图所示：



示例代码 2

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.rect(20, 20, 110, 40)
ctx.setFillStyle('blue')
ctx.fill()

ctx.beginPath()
ctx.rect(20, 30, 150, 40)

ctx.setFillStyle('yellow')
ctx.fillRect(20, 80, 150, 40)

ctx.rect(20, 150, 150, 40)

ctx.setFillStyle('red')
ctx.fill()
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.14. CanvasContext.fillRect

填充矩形。

`setFillStyle()` 用于设置矩形的填充色，如果没设置则默认是 `black` 。

入参

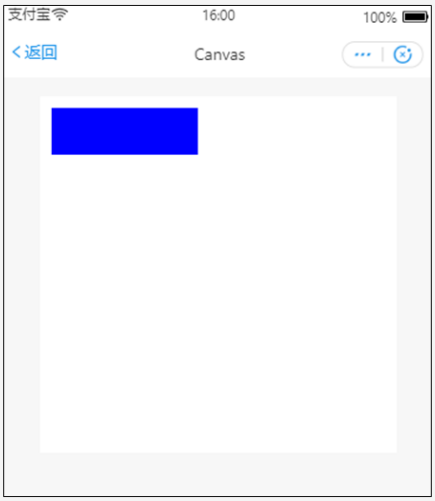
Object 类型，属性如下：

属性	类型	说明
x	Number	矩形左上角的 x 坐标。
y	Number	矩形左上角的 y 坐标。
width	Number	矩形路径宽度。
height	Number	矩形路径高度。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.fillRect(20, 20, 250, 80)
ctx.setFillStyle('blue')
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.15. CanvasContext.fillText

在画布上绘制被填充的文本。

入参

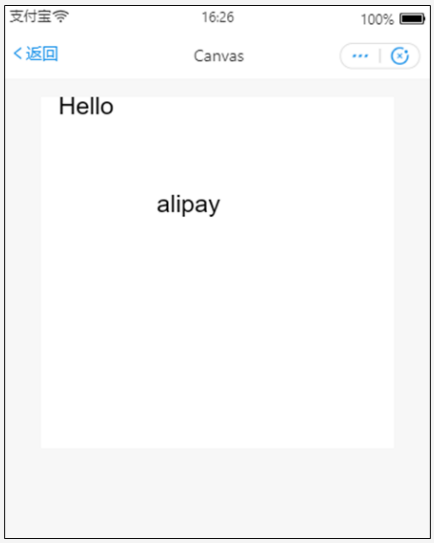
Object 类型，属性如下：

属性	类型	说明
text	String	文本。
x	Number	绘制文本的左下角 x 坐标。
y	Number	绘制文本的左下角 y 坐标。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.setFontSize(42)
ctx.fillText('Hello', 30, 30)
ctx.fillText('alipay', 200, 200)

ctx.draw()
```

显示效果如下图所示：



4.4.8.2.16. CanvasContext.getImageData

版本需求： 支持基础库 1.10 及以上版本，低版本需做 兼容处理。

获取 canvas 区域隐含的像素数据。

入参

Object 类型，属性如下：

属性	类型	必填	说明
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	成功回调。
fail	Function	否	失败回调。
complete	Function	否	完成回调。

success 回调入参

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。

示例代码

```
// .js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.getImageData({
  x: 0,
  y: 0,
  width: 100,
  height: 100,
  success(res) {
    console.log(res.width) // 100
    console.log(res.height) // 100
    console.log(res.data instanceof Uint8ClampedArray) // true
    console.log(res.data.length) // 100 * 100 * 4
  }
})
```

4.4.8.2.17. CanvasContext.lineTo

`lineTo` 方法增加一个新点，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。

入参

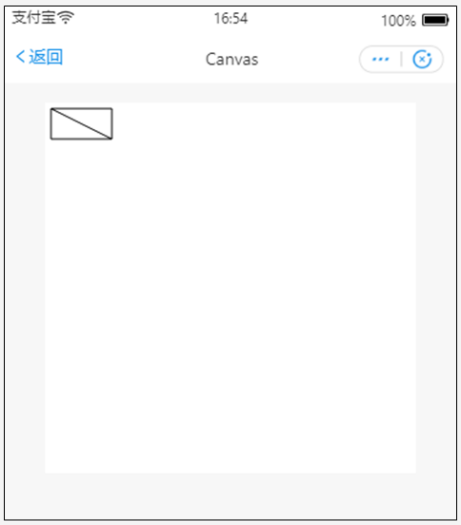
Object 类型，属性如下：

属性	类型	说明
x	Number	目标位置 x 坐标。
y	Number	目标位置 y 坐标。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.moveTo(10, 10)
ctx.rect(10, 10, 100, 50)
ctx.lineTo(110, 60)
ctx.stroke()
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.18. CanvasContext.measureText

测量文本尺寸信息，目前仅返回文本宽度。同步接口。

入参

Object 类型，属性如下：

属性	类型	必填	说明
text	String	是	要测量的文本。

返回值

返回 TextMetrics 对象，结构如下：

参数	类型	说明
width	Number	文本的宽度。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.font = 'italic bold 50px cursive'
const { width } = ctx.measureText('hello world')
console.log(width)
```

4.4.8.2.19. CanvasContext.moveTo

把路径移动到画布中的指定点，不创建线条。 `stroke()` 方法用于画线条。

入参

Object 类型，属性如下：

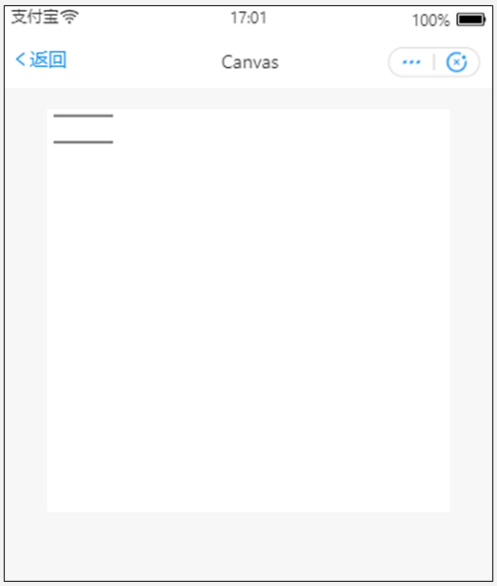
属性	类型	说明
x	Number	目标位置 x 坐标。

y	Number	目标位置 y 坐标。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.moveTo(10, 10)
ctx.lineTo(100, 10)

ctx.moveTo(10, 50)
ctx.lineTo(100, 50)
ctx.stroke()
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.20. CanvasContext.moveTo

版本要求： 基础库 1.11.0 或更高版本，若版本较低，建议做 兼容处理。

将像素数据绘制到画布。

入参

Object 类型，属性如下：

属性	类型	必填	说明
data	Uint8ClampedArray	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 。
height	Number	是	源图像数据矩形区域的高度。

success	Function	否	成功回调。
fail	Function	否	失败回调。
complete	Function	否	完成回调。

示例代码

```
// .js
const data = new Uint8ClampedArray([255, 0, 0, 1])
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.putImageData({
    x: 0,
    y: 0,
    width: 1,
    height: 1,
    data: data,
    success(res) {}
})
```

4.4.8.2.21. CanvasContext.quadraticCurveTo

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。

入参

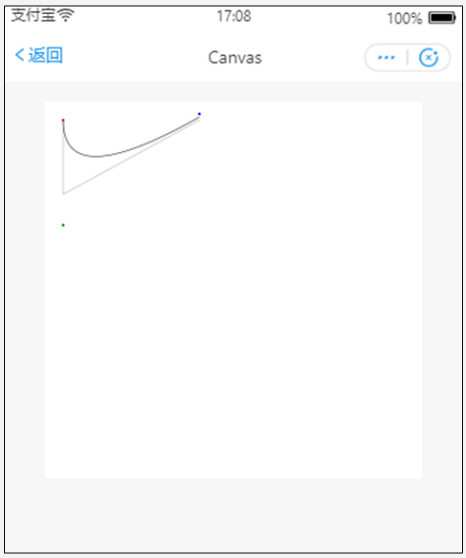
Object 类型，属性如下：

属性	类型	说明
cpx	Number	贝塞尔控制点 x 坐标。
cpy	Number	贝塞尔控制点 y 坐标。
x	Number	结束点 x 坐标。
y	Number	结束点 y 坐标。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.beginPath()
ctx.arc(30, 30, 2, 0, 2 * Math.PI)
ctx.setFillStyle('red')
ctx.fill()

ctx.beginPath()
ctx.arc(250, 20, 2, 0, 2 * Math.PI)
ctx.setFillStyle('blue')
ctx.fill()

ctx.beginPath()
ctx.arc(30, 200, 2, 0, 2 * Math.PI)
ctx.setFillStyle('green')
ctx.fill()

ctx.setFillStyle('black')
ctx.setFontSize(12)

ctx.beginPath()
ctx.moveTo(30, 30)
ctx.lineTo(30, 150)
ctx.lineTo(250, 30)
ctx.setStrokeStyle('#AAAAAA')
ctx.stroke()

ctx.beginPath()
ctx.moveTo(30, 30)
ctx.quadraticCurveTo(30, 150, 250, 25)
ctx.setStrokeStyle('black')
ctx.stroke()

ctx.draw()
```

显示效果如下图所示：



针对 `moveTo(30, 30)` `quadraticCurveTo(30, 150, 250, 25)` 的三个关键坐标如下：

- 红色：起始点(30, 30)

- 蓝色：控制点(30, 150)
- 绿色：终止点(250, 25)

4.4.8.2.22. CanvasContext.rect

创建一个矩形。用 `fill()` 或者 `stroke()` 方法将矩形画到 canvas 中。

入参

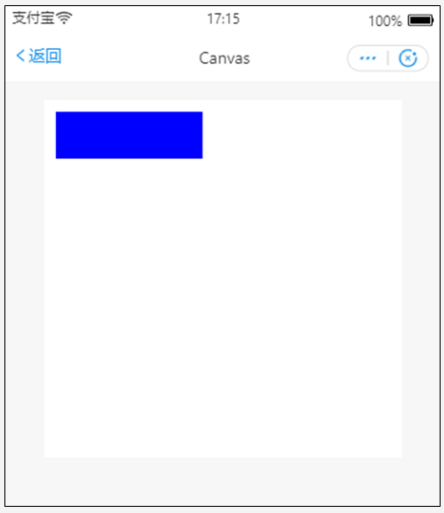
Object 类型，属性如下：

属性	类型	说明
x	Number	矩形左上角的 x 坐标。
y	Number	矩形左上角的 y 坐标。
width	Number	矩形路径宽度。
height	Number	矩形路径高度 。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.rect(20, 20, 250, 80)
ctx.setFillStyle('blue')
ctx.fill()
ctx.draw()
```

显示效果如下图所示：



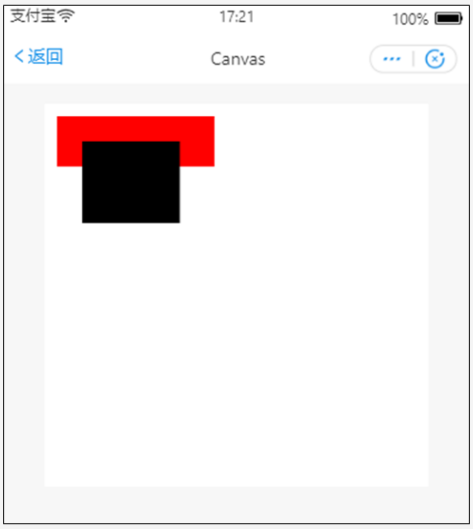
4.4.8.2.23. CanvasContext.restore

恢复之前保存的绘图上下文。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas');

ctx.save();
ctx.setFillStyle('red');
ctx.fillRect(20, 20, 250, 80);

ctx.restore();
ctx.fillRect(60, 60, 155, 130);

ctx.draw();
```

显示效果如下图所示：



4.4.8.2.24. CanvasContext.rotate

以原点为中心，原点可以用 translate 方法修改。顺时针旋转当前坐标轴。多次调用 `rotate` ，旋转的角度会叠加。

入参

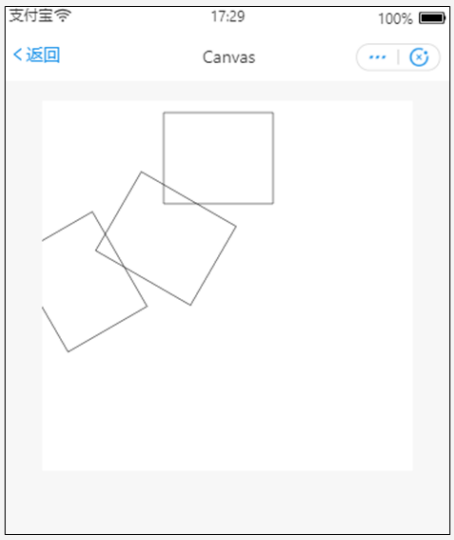
Object 类型，属性如下：

属性	类型	说明
rotate	Number	旋转角度，以弧度计(degrees * Math.PI/180；degrees 范围为 0~360) 。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.strokeRect(200, 20, 180, 150)
ctx.rotate(30 * Math.PI / 180)
ctx.strokeRect(200, 20, 180, 150)
ctx.rotate(30 * Math.PI / 180)
ctx.strokeRect(200, 20, 180, 150)

ctx.draw()
```

显示效果如下图所示：



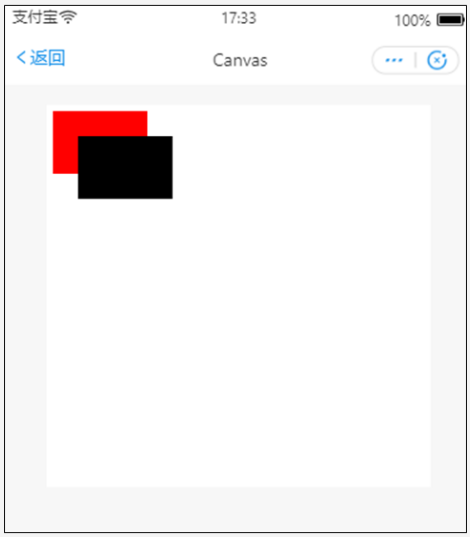
4.4.8.2.25. CanvasContext.save

保存当前的绘图上下文。

示例代码

```
//.js
const ctx = my.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()
```

显示效果如下图所示：



4.4.8.2.26. CanvasContext.scale

在调用 `scale` 方法后，之后创建的路径其横纵坐标会被缩放。多次调用 `scale` ，倍数会相乘。

入参

Object 类型，属性如下：

属性	类型	说明
scaleWidth	Number	横坐标缩放倍数 (1 = 100%，0.5 = 50%，2 = 200%)。
scaleHeight	Number	纵坐标轴缩放倍数 (1 = 100%，0.5 = 50%，2 = 200%)。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas');

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()

ctx.draw()
```

4.4.8.2.27. CanvasContext-setFillStyle

设置填充色。如果没有设置 `fillStyle` ，则默认颜色为 `black` 。

入参

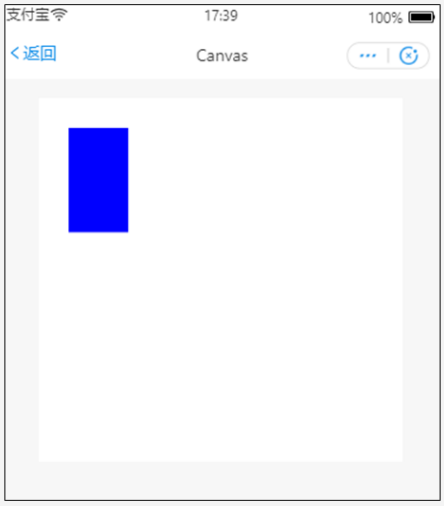
Object 类型，属性如下：

属性	类型	定义
color	Color	颜色。

示例代码

```
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.setFillStyle('blue')
ctx.fillRect(50, 50, 100, 175)
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.28. CanvasContext.setFontSize

设置字体大小。

入参

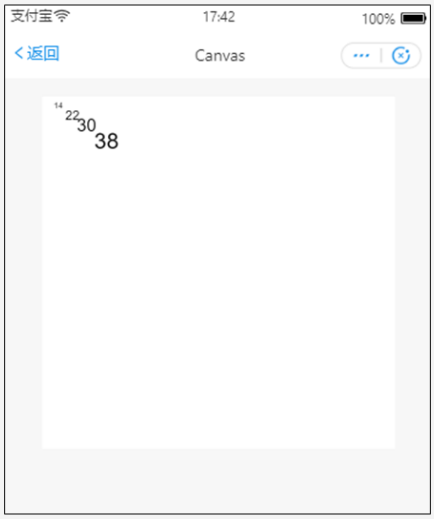
Object 类型，属性如下：

属性	类型	说明
fontSize	Number	字号。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas');

ctx.setFontSize(14)
ctx.fillText('14', 20, 20)
ctx.setFontSize(22)
ctx.fillText('22', 40, 40)
ctx.setFontSize(30)
ctx.fillText('30', 60, 60)
ctx.setFontSize(38)
ctx.fillText('38', 90, 90)

ctx.draw()
```

显示效果如下图所示：



4.4.8.2.29. CanvasContext.setGlobalAlpha

设置全局画笔透明度。

入参

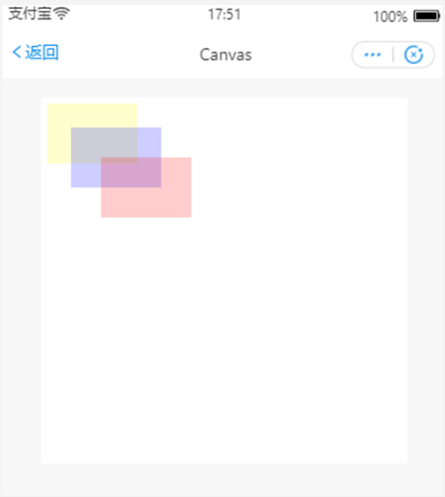
Object 类型，属性如下：

属性	类型	范围	说明
alpha	Number	0~1	透明度，0 为完全透明，1 为完全不透明。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.setFillStyle('yellow')
ctx.fillRect(10, 10, 150, 100)
ctx.setGlobalAlpha(0.2)
ctx.setFillStyle('blue')
ctx.fillRect(50, 50, 150, 100)
ctx.setFillStyle('red')
ctx.fillRect(100, 100, 150, 100)

ctx.draw()
```

显示效果如下图所示：



4.4.8.2.30. CanvasContext-setLineCap

设置线条的端点样式。

入参

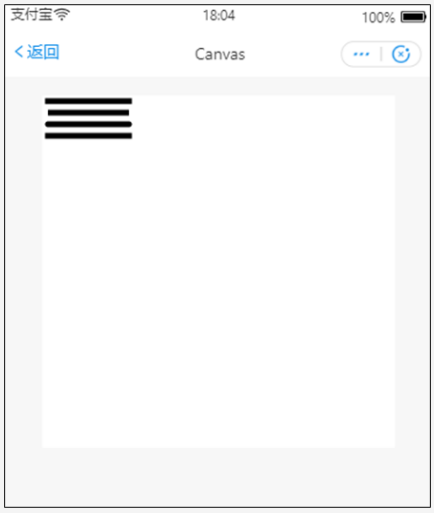
Object 类型，属性如下：

属性	类型	可选值	说明
lineCap	String	'round'、'butt'、'square'	线条的结束端点样式。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
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()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()
```

显示效果如下图所示：



4.4.8.2.31. CanvasContext.setLineDash

设置虚线的样式。

入参

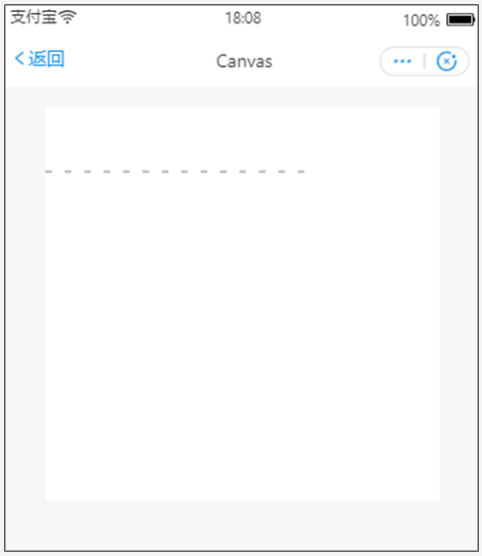
Object 类型，属性如下：

属性	类型	说明
segments	Array	一组用于描述交替绘制线段和间距（坐标空间单位）长度的数字。 如果数组元素的数量是奇数， 数组的元素会被复制并重复。例如， [5, 15, 25] 会变成 [5, 15, 25, 5, 15, 25]。

示例代码

```
 // .js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.setLineDash([10, 20], 5)

ctx.beginPath()
ctx.moveTo(0,100)
ctx.lineTo(400, 100)
ctx.stroke()

ctx.draw()
```

显示效果如下：



4.4.8.2.32. CanvasContext-setLineJoin

设置线条的交点样式。

入参

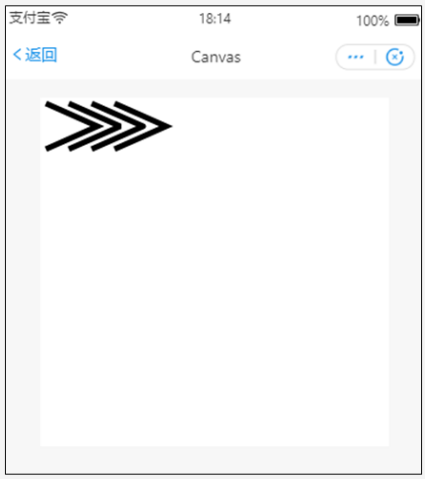
Object 类型，属性如下：

属性	类型	可选值	说明
lineJoin	String	'round'、'bevel'、'miter'	线条的结束交点样式。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
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()
```

显示效果如下图所示：



4.4.8.2.33. CanvasContext-setLineWidth

设置线条的宽度。

入参

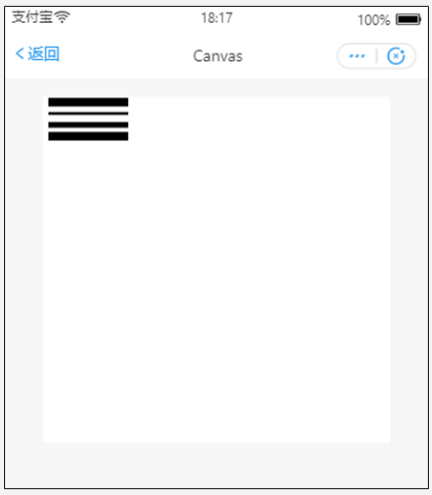
Object 类型，属性如下：

属性	类型	说明
lineWidth	Number	线条宽度，单位为 px。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
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()
```

显示效果如下所示：



4.4.8.2.34. CanvasContext.setMiterLimit

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。 仅当 `setLineJoin()` 为 `miter` 时有效。超过最大倾斜长度的，连接处将以 `lineJoin` 为 bevel 显示。

入参

Object 类型，属性如下：

属性	类型	说明
miterLimit	Number	最大斜接长度。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.beginPath()
ctx.setLineWidth(15)
ctx.setLineJoin('miter')
ctx.setMiterLimit(1)
ctx.moveTo(10, 10)
ctx.lineTo(100, 50)
ctx.lineTo(10, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(15)
ctx.setLineJoin('miter')
ctx.setMiterLimit(2)
ctx.moveTo(50, 10)
ctx.lineTo(140, 50)
ctx.lineTo(50, 90)
ctx.stroke()

ctx.beginPath()
ctx.setLineWidth(15)
ctx.setLineJoin('miter')
ctx.setMiterLimit(3)
ctx.moveTo(90, 10)
ctx.lineTo(180, 50)
ctx.lineTo(90, 90)
ctx.stroke()

ctx.draw()
```

显示效果如下图所示：

4.4.8.2.35. CanvasContext.setShadow

设置阴影的位置和颜色。

入参

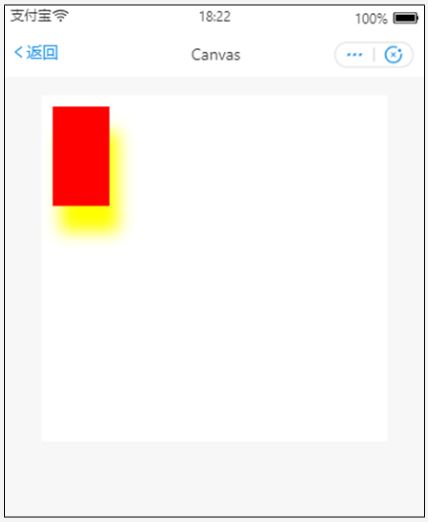
Object 类型，属性如下：

属性	类型	范围	默认值	定义
offsetX	Number		0	阴影相对于形状水平方向的偏移
offsetY	Number		0	阴影相对于形状竖直方向的偏移
blur	Number	0~100	0	阴影的模糊级别，值越大越模糊
color	Color		black	阴影颜色

示例代码

```
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.setFillStyle('red')
ctx.setShadow(15, 45, 45, 'yellow')
ctx.fillRect(20, 20, 100, 175)
ctx.draw()
```

显示效果如下所示：



4.4.8.2.36. CanvasContext.setStrokeStyle

设置边框颜色。若没有设置，则默认颜色为 `black` 。

入参

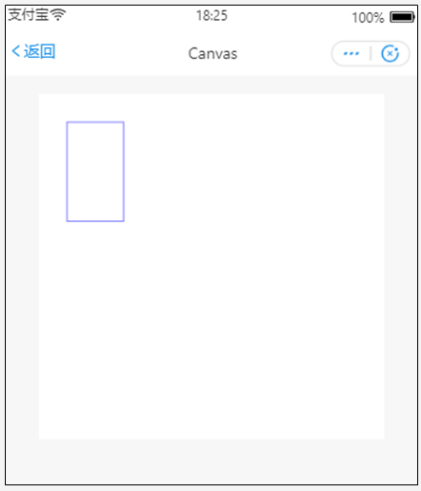
Object 类型，属性如下：

属性	类型	定义
color	Color	颜色

示例代码

```
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.setStrokeStyle('blue')
ctx.strokeRect(50, 50, 100, 175)
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.37. CanvasContext.setTextAlign

设置文本的对齐方式。该对齐是基于 CanvasRenderingContext2D.fillText 方法的 x 的值。如果 textAlign="center"，那么该文本将画在 x-50%*width 的位置。

入参

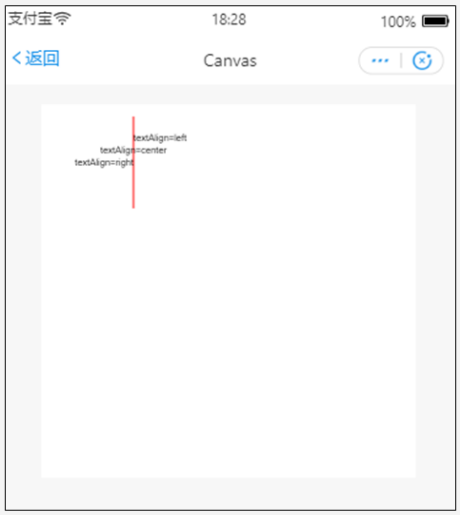
Object 类型，属性如下：

属性	类型	定义
textAlign	String	枚举 "left" "right" "center" "start" "end"

示例代码

```
const ctx = my.createCanvasContext('awesomeCanvas');
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()
```

显示效果如下图所示：



4.4.8.2.38. CanvasContext.setTextBaseline

设置当前文本基线的属性。

入参

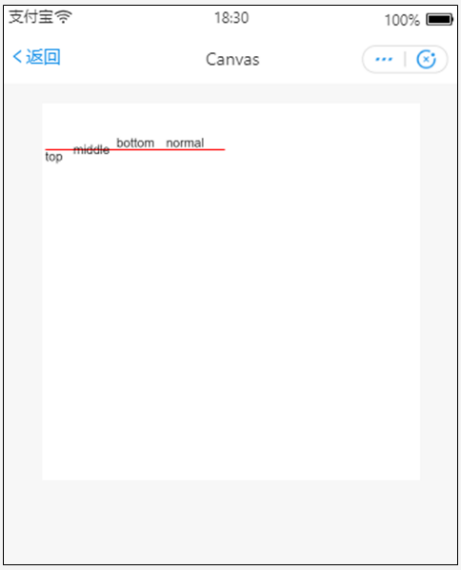
Object 类型，属性如下：

属性	类型	可选值	描述
textBaseline	String	"top" 、"hanging"、 "middle"、 "alphabetic"、 "ideographic"、 "bottom"	Canvas 2D API 描述绘制文本时，当前文本基线的属性。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
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()
```

显示效果如下图所示：



4.4.8.2.39. CanvasContext.setTransform

使用单位矩阵重新设置（覆盖）当前的变换并调用变换的方法，此变换由方法的变量进行描述。

入参

Object 类型，属性如下：

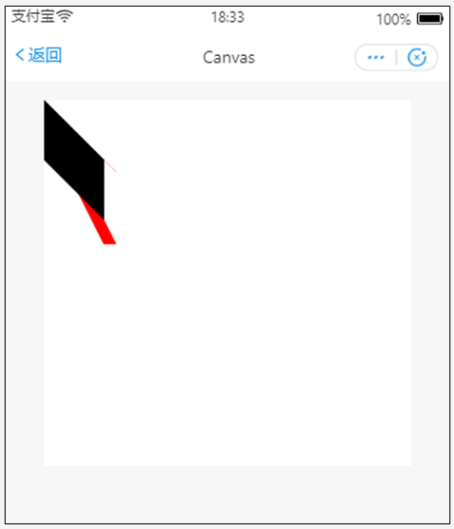
属性	类型	说明
scaleX	Number	水平缩放。
skewX	Number	水平倾斜。
skewY	Number	垂直倾斜。
scaleY	Number	垂直缩放。
translateX	Number	水平移动。
translateY	Number	垂直移动。

示例代码

//.jsconst=.createCanvasContext('awesomeCanvas')

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.rotate(45 * Math.PI / 180)
ctx.setFillStyle('red')
ctx.fillRect(70,0,100,30)

ctx.setTransform(1, 1, 0, 1, 0, 0)
ctx.setFillStyle('#000')
ctx.fillRect(0, 0, 100, 100)

ctx.draw()
```

显示效果如下图所示：



4.4.8.2.40. CanvasContext.stroke

绘制描边路径。

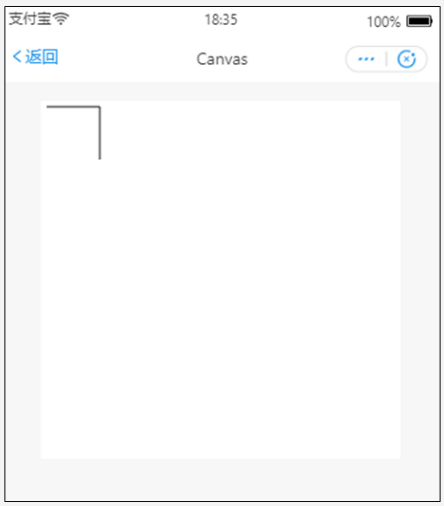
说明： `stroke()` 绘制的的路径从 `beginPath()` 开始计算，但是不会将 `strokeRect()` 包含进去，请参见示例代码 2。

示例代码

示例代码 1

```
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.moveTo(10, 10)
ctx.lineTo(100, 10)
ctx.lineTo(100, 100)
ctx.stroke()
ctx.draw()
```

显示效果如下图所示：

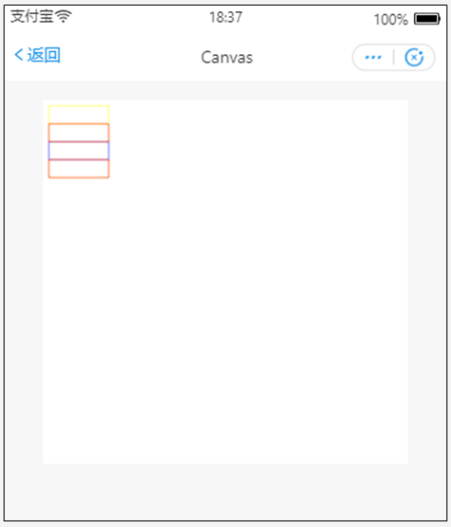


示例代码 2

```
const=.createCanvasContext('awesomeCanvas');
ctx.rect(10, 10, 100, 30)
ctx.setStrokeStyle('yellow')
ctx.stroke()
ctx.beginPath()
ctx.rect(10, 40, 100, 30)
ctx.setStrokeStyle('blue')
ctx.strokeRect(10, 70, 100, 30)
ctx.rect(10, 100, 100, 30)
ctx.setStrokeStyle('red')
ctx.stroke()
ctx.draw()
```

```
ctx  my
```

显示效果如下所示：



4.4.8.2.41. CanvasContext.strokeRect

绘制描边矩形。

说明： 用 `setFillStroke()` 设置矩形线条的颜色，如果没设置默认是 `black` 。

入参

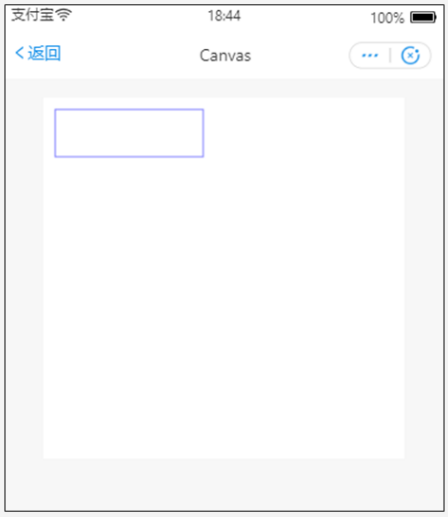
Object 类型，属性如下：

属性	类型	说明
x	Number	矩形左上角的 x 坐标。
y	Number	矩形左上角的 y 坐标。
width	Number	矩形路径宽度。
height	Number	矩形路径高度。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')
ctx.setStrokeStyle('blue')
ctx.strokeRect(20, 20, 250, 80)
ctx.draw()
```

显示效果如下图所示：



4.4.8.2.42. CanvasContext.toDataURL

获取画布指定区域的 data URL 数据。

入参

Object 类型，属性如下：

属性	类型	必填	默认值	说明
x	Number	否	0	将要被提取的矩形区域的左上角横坐标。
y	Number	否	0	将要被提取的矩形区域的左上角纵坐标。
width	Number	否	被提取的矩形区域的左上角到画布右下角的横向距离。	将要被提取的矩形区域的宽度。
height	Number	否	被提取的矩形区域的左上角到画布右下角的纵向距离。	将要被提取的矩形区域的高度。
destWidth	Number	否	默认等于 width	将要被提取的矩形区域提取后的宽度。
destHeight	Number	否	默认等于 height	将要被提取的矩形区域提取后的高度。
fileType	String	否	png	图片格式，值为 'jpg' 或 'png' 。
quality	Number	否	-	图片格式为 jpg 的情况下，data URL对应的图片的质量。取值范围是0到1，如果超出取值范围，将会默认该值为 1。其他图片格式该参数会被忽略。

返回值

Promise

类型	说明
Promise	提取的data URL字符串

示例代码

```
const ctx = my.createCanvasContext('awesomeCanvas');

ctx.setFillStyle('#108ee9');
ctx.arc(50, 50, 50, 0, Math.PI * 2, true);
ctx.fill();
ctx.draw();
ctx.toDataURL({
  x: 50,
  y: 50,
  width: 50,
  height: 50,
  destWidth: 100,
  destHeight: 100,
}).then(dataURL=>{
  ctx.drawImage(dataURL, 0, 0);
  ctx.draw();
})
```

4.4.8.2.43. CanvasContext.toTempFilePath

导出当前画布的内容为临时图片，返回文件路径。

入参

Object 类型，属性如下：

属性	类型	必填	默认值	说明
x	Number	否	0	画布 x 轴起点。
y	Number	否	0	画布 y 轴起点。
width	Number	否	默认为 canvas 宽度 x	画布宽度。
height	Number	否	默认为 canvas 高度 y	画布高度。
destWidth	Number	否	默认为 width	输出的图片宽度。
destHeight	Number	否	默认为 height	输出的图片高度。
fileType	String	否	1	目标文件的类型。合法值为jpg、png。
quality	Number	否		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		接口成功回调。

fail	Function	否		接口失败回调。
complete	Function	否		接口完成回调。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas');
ctx.toTempFilePath({
  success() {},
});
```

4.4.8.2.44. CanvasContext.transform

使用矩阵自由变换，矩阵由方法的参数进行描述。可以缩放、旋转、移动和倾斜上下文。

入参

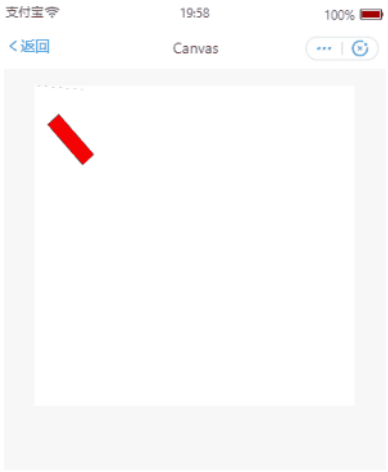
Object 类型，属性如下：

属性	类型	说明
scaleX	Number	水平缩放。
skewX	Number	水平倾斜。
skewY	Number	垂直倾斜。
scaleY	Number	垂直缩放。
translateX	Number	水平移动。
translateY	Number	垂直移动。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.rotate(45 * Math.PI / 180)
ctx.setFillStyle('red')
ctx.fillRect(70,0,100,30)

ctx.transform(1, 1, 0, 1, 0, 0)
ctx.setFillStyle('#000')
ctx.fillRect(0, 0, 100, 100)

ctx.draw()
```

显示效果如下图所示：



4.4.8.2.45. CanvasContext.translate

平移当前坐标系的原点，默认的坐标系原点是页面左上角 (0, 0)。

入参

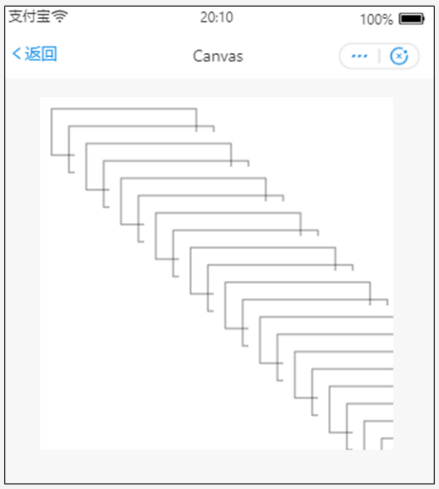
Object 类型，属性如下：

属性	类型	说明
x	Number	水平坐标平移量。
y	Number	竖直坐标平移量。

示例代码

```
//.js
const ctx = my.createCanvasContext('awesomeCanvas')

ctx.strokeRect(20, 20, 250, 80)
ctx.translate(30, 30)
ctx.strokeRect(20, 20, 250, 80)
ctx.translate(30, 30)
ctx.strokeRect(20, 20, 250, 80)

ctx.draw()
```

显示效果如下图所示：



4.4.9. 键盘

4.4.9.1. my.hideKeyboard

隐藏键盘。

示例代码

```
// API-DEMO page/API/keyboard/keyboard.json
{
  "defaultTitle": "键盘"
}
```

```
<!-- API-DEMO page/API/keyboard/keyboard.axml-->
<view class="page">
  <view class="page-description">输入框</view>
  <view class="page-section">
    <view class="form-row">
      <view class="form-row-label">密码键盘</view>
      <view class="form-row-content">
        <input class="input" password type="text" onInput="bindHideKeyboard" placeholder="输入 123 自动收起键盘" />
      </view>
    </view>

    <view class="form-row">
      <view class="form-row-label">数字键盘</view>
      <view class="form-row-content">
        <input class="input" type="digit" onInput="bindHideKeyboard" placeholder="输入 123 自动收起键盘" />
      </view>
    </view>
  </view>
</view>
```

```
// API-DEMO page/API/keyboard/keyboard.js
Page({
  bindHideKeyboard(e) {
    if (e.detail.value === "123") {
      // 收起键盘
      my.hideKeyboard();
    }
  },
});
```

4.4.10. 滚动

4.4.10.1. my.pageScrollTo

滚动到页面的目标位置。

注意：

- scrollTop 的优先级比 selector 高。
- 使用 my.pageScrollTo 跳转小程序顶部时，必须将 scrollTop 值设为大于 0，方可实现跳转。

扫码体验

示例代码

```
<!-- API-DEMO page/API/page-scroll-to/page-scroll-to.axml-->
<view class="page">
  <view class="page-description">页面滚动 API</view>

  <view class="page-section">
    <view class="page-section-title">
      my.pageScrollTo
    </view>
    <view class="page-section-demo">
      <input type="text" placeholder="key" name="key" value="{{scrollTop}}" onInput="scrollTopChange"></input>
    </view>
    <view class="page-section-btns">
      <view onTap="scrollTo">页面滚动</view>
    </view>
  </view>

  <view style="height:1000px"/>
</view>undefined
```

```
// API-DEMO page/API/page-scroll-to/page-scroll-to.js
Page({
  data: {
    scrollTop: 0,
  },
  scrollTopChange(e) {
    this.setData({
      scrollTop: e.detail.value,
    });
  },
  onPageScroll({ scrollTop }) {
    console.log('onPageScroll', scrollTop);
  },
  scrollTo() {
    my.pageScrollTo({
      scrollTop: parseInt(this.data.scrollTop),
            duration: 300,
    });
  },
});undefined
```

入参

为 Object 类型，属性如下：

属性	类型	默认值	必填	描述
scrollTop	Number	-	否	滚动到页面的目标位置，单位 px。使用 my.pageScrollTo 跳转小程序顶部时，必须将 scrollTop 值设为大于 0，方可实现跳转。
duration	Number	0	否	滚动动画的时长，单位ms。
selector	string	-	否	选择器。
success	function	-	否	接口调用成功的回调函数。
fail	function	-	否	接口调用失败的回调函数。
complete	function	-	否	接口调用结束的回调函数（调用成功、失败都会执行）。

selector语法

当传入 selector 参数，框架会执行 document.querySelector(selector) 以选取目标节点。

4.4.11. 设置窗口背景

4.4.11.1. my.setBackgroundColor

动态设置窗口的背景色。

示例代码

```
my.setBackgroundColor({
    backgroundColorTop: '#00ff00',
    backgroundColorBottom: '#ff00ff'
})
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
backgroundColor	HexColor	是	窗口的背景色
backgroundColorTop	HexColor	是	顶部窗口的背景色，仅 iOS 支持
backgroundColorBottom	HexColor	是	底部窗口的背景色，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）

4.4.11.2. my.setBackgroundTextStyle

动态设置下拉 loading 页面的文字颜色。

示例代码

```
my.setBackgroundTextStyle({
    textStyle: 'dark', // 下拉 loading 页面的文字颜色为dark
})
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
textStyle	String	是	下拉背景字体、loading 图的样式，仅支持 'dark', 'light'。
success	Function	否	接口调用成功的回调函数。
fail	Function	否	接口调用失败的回调函数。
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）。

4.4.12. 设置optionMenu

4.4.12.1. my.setOptionMenu

配置 optionMenu 导航栏额外图标，点击后触发 `onOptionMenuClick` 。

示例代码

```
my.setOptionMenu({
  icon: 'https://img.alicdn.com/tps/i3/T1OjaVFl4dXXa.JOZB-114-114.png',
});
```

入参

入参为 String 类型，属性如下：

属性	类型	必填	描述

icon	String	是	• 自定义 optionMenu 所用图标的 URL 或 base64 图片。 • 图片尺寸建议为 30*30 px。 • 更多使用限制请参见 下方 **icon 属性使用须知**。

icon 属性使用须知

- 由于 iOS 的 ATS 限制，icon URL 必须为 https 链接或 base64，http 链接会被忽略。
- icon 图标为 base64 格式时，只支持矢量图格式，且请勿写“data:image/png;base64”前缀。

4.4.13. 字体

4.4.13.1. my.loadFontFace

动态加载网络字体，文件地址需为下载类型。iOS 仅支持 https 格式文件地址。

阿里云小程序目前只支持 woff，otf，ttf，sfnt字体。

注意：阿里云小程序不支持 woff2 字体，原因是：

- 相对其他格式字体，对内存占用较高。
- 此字体支持对于内核 so size 有较大负担，目前阿里云使用的 u4 内核 3.0 将 woff2 格式支持给裁剪了，导致无法正常显示， 建议业务使用其他格式。

示例代码

```
<!-- .axml -->
<view class="page">
  <view class="page-description">动态加载网络字体</view>
  <view class="page-section">
    <view class="page-section-title">loadFontFace</view>
    <view class="page-section-demo">
      <button size="default" type="primary" onTap="loadFontFace">
        loadFontFace
      </button>
    </view>
  </view>
</view>
```

```
// .js
Page({
  data: {},
  onLoad() { },
  loadFontFace() {
    my.loadFontFace({
      family: 'Bitstream Vera Serif Bold',
      source: 'url("https://sungd.github.io/Pacifico.ttf")',
      success() {
        my.alert({
          title: 'loadfontface 成功!!!',
        })
      },
      fail: (err) => {
        my.alert({
          content: JSON.stringify(err),
        })
      },
    })
  },
})
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
family	String	是	字体名称
source	String	是	字体资源地址。
desc	Object	否	字体描述符
success	Function	否	调用成功的回调函数
fail	Function	否	调用失败的回调函数
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）

desc 结构

名称	类型	默认值	必填	描述
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

4.5. 缓存

4.5.1. 缓存 API 概览

开启本地缓存数据，进行存储、获取和删除等控制。

单个小程序的缓存总上限为 10MB。

同步方法会阻塞当前任务，直到同步方法处理返回。异步方法不会阻塞当前任务。

操作	同步	异步	描述
存储	my.setStorageSync	my.setStorage	数据存储在本地缓存中指定的 key 中的接口，会覆盖掉原来该 key 对应的数据
读取	my.getStorageSync	my.getStorage	获取缓存数据的接口
清除	my.clearStorageSync	my.clearStorage	清除本地数据缓存的接口
删除	my.removeStorageSync	my.removeStorage	删除缓存数据的接口
获取相关信息	my.getStorageInfoSync	my.getStorageInfo	获取当前 storage 的相关信息的接口

4.5.2. my.clearStorage

清除本地数据缓存的异步接口。清空内嵌 webview 的存储时不会同时清空当前小程序本身的存储数据。

扫码体验

示例代码

```
my.clearStorage()
```

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

4.5.3. my.clearStorageSync

清除本地数据缓存的同步接口。

扫码体验

示例代码

```
my.clearStorageSync()
```

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

4.5.4. my.getStorage

获取缓存数据的异步接口。支持内嵌 webview 内缓与小程序缓存隔离，获取内嵌 webview 指定 key 的缓存不会同时返回小程序相同key下的缓存数据。

扫码体验

示例代码

```
my.getStorage({
  key: 'currentCity',
  success: function(res) {
    my.alert({content: '获取成功：' + res.data.cityName});
  },
  fail: function(res){
    my.alert({content: res.errorMessage});
  }
});
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
key	String	是	缓存数据的 key。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

success 回调函数

入参为 Object 类型，属性如下：

属性	类型	说明
data	Object/String	key 对应的内容。

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回。
- 覆盖安装阿里云(不是先删除再安装)，不会导致小程序缓存失效。
- 阿里云设置中心清除缓存不会导致小程序缓存失效。
- 小程序缓存默认具有阿里云账号和小程序 id 两级隔离。
- iOS 客户端支持 iTunes 备份。

4.5.5. my.getStorageInfo

获取当前 storage 的相关信息的异步接口。在内嵌 webview 内获取当前 storage 的相关信息，但不会获取到当前小程序 storage 的相关信息。

扫码体验

示例代码

```
my.getStorageInfo({
  success: function(res) {
    console.log(res.keys)
    console.log(res.currentSize)
    console.log(res.limitSize)
  }
})
```

入参

入参为 Object 类型，属性如下：

名称	类型	必填	描述
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

success 回调函数

入参为 Object 类型，属性如下：

属性	类型	说明
keys	String 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。

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；

- iOS 客户端支持 iTunes 备份。

4.5.6. my.getStorageInfoSync

获取当前 storage 相关信息的同步接口。

扫码体验

示例代码

```
var res = my.getStorageInfoSync()
console.log(res.keys)
console.log(res.currentSize)
console.log(res.limitSize)
```

返回值

名称	类型	说明
keys	String/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

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云南（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

4.5.7. my.getStorageSync

获取缓存数据的同步接口。

扫码体验

示例代码

```
let res = my.getStorageSync({ key: 'currentCity' });
 my.alert({
    content: JSON.stringify(res.data),
 });
```

入参

入参为 String 类型，属性如下：

属性	类型	必填	描述
key	String	是	缓存数据的 key

返回值

名称	类型	说明
data	Object/String	key 对应的内容

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

4.5.8. my.removeStorage

删除缓存数据的异步接口。移除内嵌 webview 的存储数据时不会移除当前小程序的存储数据。

扫码体验

示例代码

```
my.removeStorage({
  key: 'currentCity',
  success: function(){
    my.alert({content: '删除成功'});
  }
});
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
key	String	是	缓存数据的 key。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

4.5.9. my.removeStorageSync

删除缓存数据的同步接口。

扫码体验

示例代码

```
my.removeStorageSync({
  key: 'currentCity',
});
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
key	String	是	缓存数据的 key。

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

4.5.10. my.setStorage

将数据存储在本地缓存中指定的 key 中的异步接口，会覆盖掉原来该 key 对应的数据。支持内嵌 webview 的存储与小程序存储隔离，内嵌 webview 中指定 key 存储数据不会覆盖小程序自身相同 key 对应的数据。单条数据转换成字符串后，字符串长度最大为 200*1024。对于同一个阿里云用户的同一个小程序，缓存总上限为 10MB。

说明

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装）、阿里云设置中心清除缓存、关闭小程序，这三种操作均不会导致小程序缓存失效。
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

示例代码

```
my.setStorage({
  key: 'currentCity',
  data: {
    cityName: '杭州',
    adCode: '330100',
    spell: ' hangzhou',
  },
  success: function() {
    my.alert({content: '写入成功'});
  }
});
```

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
key	String	是	缓存数据的 key。
data	Object/String	是	要缓存的数据。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.5.11. my.setStorageSync

将数据存储在本地缓存中指定的 key 中的同步接口。

扫码体验

入参

入参为 Object 类型，属性如下：

属性	类型	必填	描述
key	String	是	缓存数据的 key。
data	Object/String	是	要缓存的数据。

示例代码

```
my.setStorageSync({
  key: 'currentCity',
  data: {
    cityName: '杭州',
    adCode: '330100',
    spell: ' hangzhou',
  }
});
```

其他信息

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回；
- 覆盖安装阿里云（不是先删除再安装），不会导致小程序缓存失效；
- 阿里云设置中心清除缓存不会导致小程序缓存失效；
- 小程序缓存默认具有阿里云账号和小程序 ID 两级隔离；
- iOS 客户端支持 iTunes 备份。

4.6. 网络

4.6.1. 网络 API 使用须知

在使用 网络 API 时，需要注意以下问题。

服务器域名白名单

服务器域名白名单是为了保证用户安全性，所做的限制性措施，使用方法如下：

- 请预先在 阿里云**小程序管理中心 > 小程序详情 > 设置 > 开发设置 > 服务器域名白名单** 中配置域名白名单。小程序在以下 API 调用时只能与白名单中的域名进行通讯：HTTP请求（my.request）、上传文件（my.uploadFile）、下载文件（my.downloadFile）和 WebSocket（my.connectSocket）。

添加服务器域名白名单后，需要重新打包上传生成体验版，服务器域名才会生效。

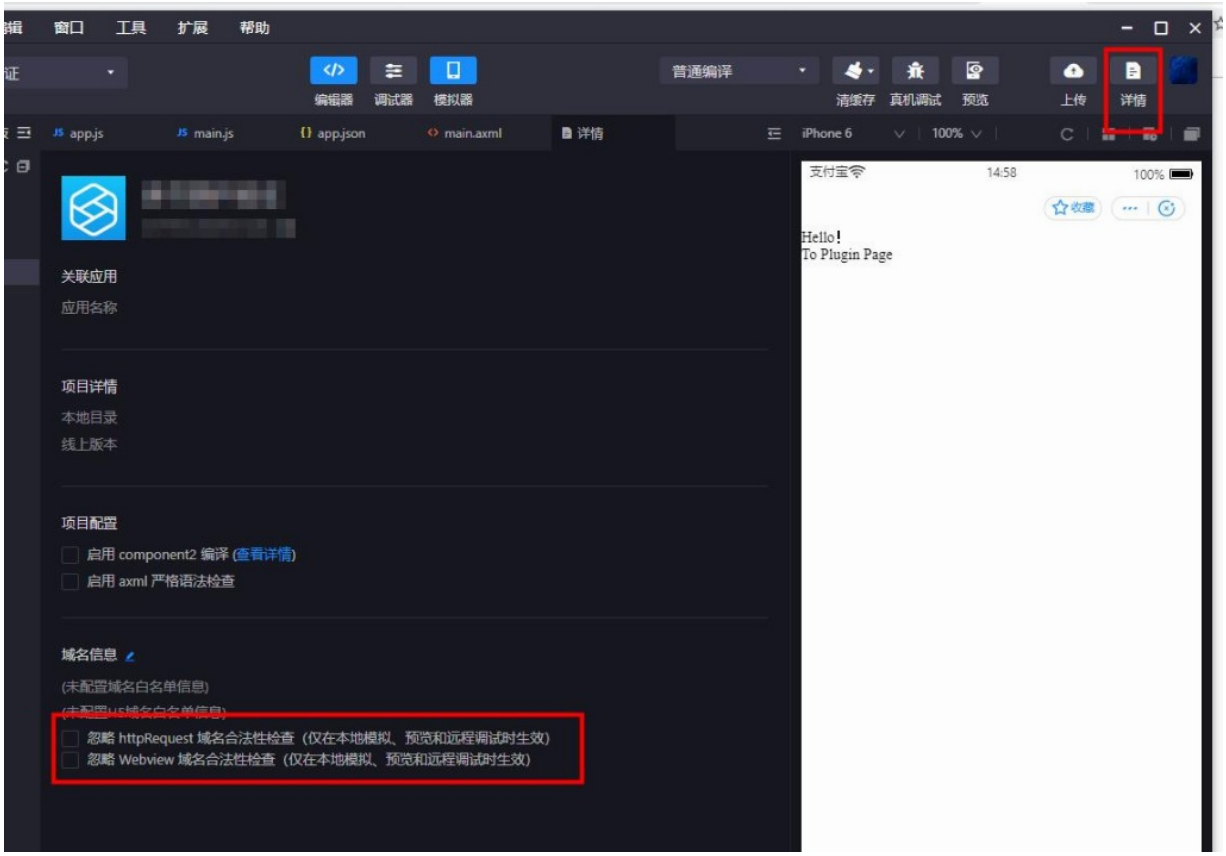
- 服务器域名白名单是以一级域名进行匹配的。

..1..2..1..2..3....

假如商户有 acom 和 bcom 两个域名，业务分别部署在
acom
acom
bcom
bcom
bcom
那么，在服务器域名白名单里面只需要配置 acom 和 bcom 即可，这里是一级域名匹配。

跳过域名检验

- 若在 IDE 中进行测试，请设置 **忽略 httpRequest 域名合法性检查**。IDE 中调试不受服务器域名白名单影响。



网络请求

- 默认超时时间为 30000 ms。
- 小程序会自带 HTTP Referer，暂时不支持自定义设置 HTTP Referer。
- 只要成功接收到服务器返回，无论 status 是多少，都会进入 success 回调函数。请开发者根据业务逻辑对返回值进行判断。

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。

小程序只支持 HTTPS 域名配置。 相比于 HTTP，HTTPS 可以提供更加优质保密的信息，保证了用户数据的安全性，此外 HTTPS 同时也一定程度上保护了服务端，使用恶意攻击和伪装数据的成本大大提高。

因此小程序强制使用 HTTPS，还在使用 HTTP 协议的开发者需要尽快对服务器进行升级。

为了方便开发者尽快进行 HTTPS 的配置，我们提供了免费的 SSL 证书。

4.6.2. my.closeSocket

关闭 WebSocket 连接。

示例代码

```
my.onSocketOpen(function() {
  my.closeSocket()
})

my.onSocketClose(function(res) {
  console.log('WebSocket 已关闭！')
})
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
success	Function	否	回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.6.3. my.connectSocket

创建一个 WebSocket 的连接。一个阿里云小程序同时只能保留一个 WebSocket 连接，如果当前已存在 WebSocket 连接，会自动关闭该连接，并重新创建一个新的 WebSocket 连接。

使用前提：

> *请预先在 阿里云**小程序管理中心 > 小程序详情 >设置 > 开发设置 > 服务器域名白名单** 中配置域名白名单。小程序在以下 API 调用时只能与白名单中的域名进行通讯：HTTP请求（my.request）、上传文件（my.uploadFile）、下载文件（my.downloadFile）和 WebSocket（my.connectSocket）。*

扫码体验

示例代码

```
my.connectSocket({
  url: 'test.php',
  data: {},
  header:{
    'content-type': 'application/json'
  },
});
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
url	String	是	目标服务器接口地址。 **注意**：部分新发布的小程序只支持 wss 协议。
data	Object	否	请求的参数。
header	Object	否	设置请求的头部。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

结果码

结果码	描述
1	未知错误
2	网络连接已经存在
3	URL 参数为空
4	无法识别的 URL 格式
5	URL 必须以 ws 或者 wss 开头
6	连接服务器超时
7	服务器返回的 https 证书无效
8	服务端返回协议头无效
9	WebSocket 请求没有指定 Sec-WebSocket-Protocol 请求头

10	网络连接没有打开，无法发送消息
11	消息发送失败
12	无法申请更多内存来读取网络数据

4.6.4. my.downloadFile

下载文件资源到本地。

使用前提：

> *请预先在 阿里云**小程序管理中心 > 小程序详情 > 设置 > 开发设置 > 服务器域名白名单** 中配置域名白名单。小程序在以下 API 调用时只能与白名单中的域名进行通讯：HTTP 请求（my.request）、上传文件（my.uploadFile）、下载文件（my.downloadFile）和 WebSocket（my.connectSocket）。*

扫码体验

示例代码

```
// API-DEMO page/API/download-file/download-file.json
{
    "defaultTitle": "下载文件"
}
```

```
<!-- API-DEMO page/API/download-file/download-file.axml-->
<view class="container">
  <button onTap="download">下载图片并显示</button>
</view>
```

```
// API-DEMO page/API/download-file/download-file.js
Page({
  download() {
    my.downloadFile({
      url: 'http://img.alicdn.com/tfs/TB1x669SXXXXXbdaFXXXXXXXXXX-520-280.jpg',
      success({ apFilePath }) {
        my.previewImage({
          urls: [apFilePath],
        });
      },
      fail(res) {
        my.alert({
          content: res.errorMessage || res.error,
        });
      },
    });
  },
})
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
url	String	是	下载文件地址。
header	Object	否	HTTP 请求 Header。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

success 回调函数

入参为 Object 类型，属性如下：

名称	类型	描述
apFilePath	String	文件临时存放的位置。

结果码

结果码	描述
12	下载失败。
13	没有权限。

4.6.5. my.offSocketClose

取消监听 WebSocket 关闭事件。

示例代码

```
Page({
  onLoad() {
  my.onSocketClose(this.callback);
  },
  onUnload() {
    my.offSocketClose(this.callback);
    //    my.offSocketClose();
  },
  callback(res) {
  my.alert({content: '连接已关闭！'});
      this.setData({
        sendMessageAbility: false,
        closeLinkAbility: false,
      });
  },
})
```

是否需要传 callback 值

- 不传递 callback 值，则会移除监听所有的事件监听回调。示例代码如下：

myoffSocketClose

- 传递 callback 值，只移除对应的 callback 事件。示例代码如下：

myoffSocketClosethiscallback

4.6.6. my.offSocketOpen

取消监听 WebSocket 连接打开事件。

示例代码

```
Page({
  onLoad() {
    this.callback = this.callback.bind(this);
    my.onSocketOpen(this.callback);
  },
  onUnload() {
    my.offSocketOpen(this.callback);
  },
  callback(res) {
  },
})
```

是否需要传 callback 值

- 不传递 callback 值，则会移除监听所有的事件监听回调。示例代码如下：

```
myoffSocketOpen
```

- 传递 callback 值，只移除对应的 callback 事件。示例代码如下：

```
myoffSocketOpenthiscallback
```

4.6.7. my.offSocketError

取消监听 WebSocket 错误。

示例代码

```
Page({
  onLoad() {
    this.callback = this.callback.bind(this);
    my.onSocketError(this.callback);
  },
  onUnload() {
    my.offSocketError(this.callback);
  },
  callback(res) {
  },
})
```

是否需要传 callback 值

- 不传递 callback 值，则会移除监听所有的事件监听回调。示例代码如下：

```
myoffSocketError
```

- 传递 callback 值，只移除对应的 callback 事件。示例代码如下：

```
myoffSocketErrorthiscallback
```

4.6.8. my.onSocketClose

监听 WebSocket 关闭。

示例代码

```
onLoad() {
    // 注意： 回调方法的注册在整个小程序启动阶段只要做一次，调多次会有多次回调
    my.onSocketClose((res) => {
      my.alert({content: '连接已关闭！'});
      this.setData({
        sendMessageAbility: false,
        closeLinkAbility: false,
      });
    });
    // 注意： 回调方法的注册在整个小程序启动阶段只要做一次，调多次会有多次回调
    my.onSocketOpen((res) => {
      my.alert({content: '连接已打开！'});
      this.setData({
        sendMessageAbility: true,
        closeLinkAbility: true,
      });
    });

    my.onSocketError(function(res){
      my.alert('WebSocket 连接打开失败，请检查！' + res);
    });

    // 注意： 回调方法的注册在整个小程序启动阶段只要做一次，调多次会有多次回调
    my.onSocketMessage((res) => {
      my.alert({content: '收到数据！' + JSON.stringify(res)});
    });
  }

connect_start() {
    my.connectSocket({
      url: '服务器地址', // 开发者服务器接口地址，必须是 wss 协议，且域名必须是后台配置的合法域名
      success: (res) => {
        my.showToast({
          content: 'success', // 文字内容
        });
      },
      fail:()=>{
        my.showToast({
          content: 'fail', // 文字内容
        });
      }
    });
  },
```

入参

Obejct 类型，属性如下：

参数	类型	必填	描述
callback	Function	是	WebSocket 连接关闭事件的回调函数。

4.6.9. my.onSocketError

监听 WebSocket 错误。

示例代码

```
my.connectSocket({
  url: '开发者的服务器地址'
});

my.onSocketOpen(function(res){
  console.log('WebSocket 连接已打开！');
});

my.onSocketError(function(res){
  console.log('WebSocket 连接打开失败，请检查！');
});
```

入参

Object 类型，属性如下：

参数	类型	必填	说明
callback	Function	是	WebSocket 错误事件的回调函数。

4.6.10. my.onSocketMessage

监听 WebSocket 接受到服务器的消息事件。

回调返回值

属性	类型	描述
data	String / ArrayBuffer	服务器返回的消息：普通的文本 String 或者经 base64 编码后的 String。
isBuffer	Boolean	如果此字段值为 `true`，`data` 字段表示接收到的经过了 base64 编码后的 String，否则 `data` 字段表示接收到的普通 String 文本。

示例代码

```
my.connectSocket({
  url: '服务器地址'
})

my.onSocketMessage(function(res) {
  console.log('收到服务器内容：' + res.data)
})
```

4.6.11. my.onSocketOpen

监听 WebSocket 连接打开事件。

示例代码

```
my.connectSocket({
  url: 'test.php',
});

my.onSocketOpen(function(res) {
  console.log('WebSocket 连接已打开！');
});
```

入参

Object 类型，属性如下：

属性	类型	必填	说明
callback	Function	是	WebSocket 连接打开事件的回调函数。

4.6.12. my.request

版本要求：基础库 1.11.0 或更高版本；阿里云客户端 10.1.32 或更高版本，若版本较低，建议做 兼容处理。

发起网络请求：

- my.request 目前支持 GET/POST/PUT/DELETE（其中 PUT、DELETE 请求在阿里云客户端 10.1.82 或更高版本支持）。
- my.request 目前只支持 https 协议的请求。

更多问题请参见 常见问题。

> *使用说明：*
>
> *请预先在 阿里云**小程序管理中心 > 小程序详情 > 设置 > 开发设置 > 服务器域名白名单** 中配置域名白名单。小程序在以下 API 调用时只能与白名单中的域名进行通讯：HTTP请求（my.request）、上传文件（my.uploadFile）、下载文件（my.downloadFile）和 WebSocket（my.connectSocket）。*
>
> 
>
>
> *添加服务器域名白名单后，需要重新打包上传生成体验版，服务器域名才会生效。*
>
> *在 IDE 上进行调试时，请使用真机预览调试。*
>
> *阿里云客户端已不再维护 my.httpRequest，建议使用 my.request。另外，钉钉客户端尚不支持 my.request。若在钉钉客户端开发小程序，则需要使用 my.httpRequest。*

扫码体验

重要：

- 小程序开发过程中，可在开发工具内

  详情 > 域名信息 > 忽略 httpRequest 域名合法性检查

中选择是否忽略域名合法性检查，如果选择忽略，则在模拟器、预览以及真机调试场景不会校验域名合法性，但小程序上线前必须确保通讯域名在白名单内，否则在正式版本无法调用。

- my.request 的请求头默认值为 {'content-type': 'application/json'}，而不是{'content-type': 'application/x-www-form-urlencoded'}。此外，请求头对象里面的 key 和 value 必须是 String 类型。

示例代码

```
// dataType 为 json 示例
my.request({
  url: 'https://httpbin.org/post',
  method: 'POST',
  data: {
    from: '阿里云',
    production: 'AlipayJSAPI',
  },
  headers:{
    'content-type':'application/json'  //默认值
  },
  dataType: 'json',
  success: function(res) {
    my.alert({content: 'success'});
  },
  fail: function(res) {
    my.alert({content: 'fail'});
  },
  complete: function(res) {
    my.hideLoading();
    my.alert({content: 'complete'});
  }
});

// dataType 为 base64 示例
my.request({
  url: 'https://gw.alipayobjects.com/mdn/miniapp_de/afts/img/A*G1kWSJbe2zEAAAAAAAAAAABjARQnAQ',
  method: 'GET',
  dataType: 'base64',
  success: (resp) => {
    console.log('resp data length', resp.data.length);
    console.log('resp data', resp.data); // 返回格式类似于：...
  },
  fail: (err) => {
    console.log('error', err);
  },
});undefined
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
url	String	是	目标服务器 URL。
headers	Object	否	设置请求的 HTTP 头对象，默认 {'content-type': 'application/json'}，该对象里面的 key 和 value 必须是 String 类型。
method	String	否	默认 GET，目前支持 GET/POST/PUT/DELETE。

data	Object	否	详见 **data 参数说明** 。
timeout	Number	否	超时时间，单位 ms，默认 30000。
dataType	String	否	期望返回的数据格式，默认 JSON，支持 JSON、text、base64、arraybuffer（10.1.70 版本开始支持）
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

data 参数说明

传给服务器的数据最终会是 String 类型，如果 data 不是 String 类型，会被转换成 String 。转换规则如下：

- 若方法为

  `GET`

  ，会将数据转换成 query string：
  encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...
- 若方法为

  `POST`

  且

  `headers['content-type']`

  为

  `application/json`

  ，会对数据进行 JSON 序列化
- 若方法为

  `POST`

  且

  `headers['content-type']`

  为

  `application/x-www-form-urlencoded`

  ，会将数据转换成 query string：
  encodeURIComponent(k)=encodeURIComponent(v)&encodeURIComponent(k)=encodeURIComponent(v)...

success 回调函数

入参为 Object 类型，属性如下：

属性	类型	描述
data	String	响应数据，格式取决于请求时的 dataType 参数，
status	Number	响应码。
headers	Object	响应头。

错误码

错误码	描述
error: 1	{error: 1, message: "not implemented!"} 请求没有结束，就跳转到了另一个页面。
error: 2	参数错误。
error: 11	无权跨域。
error: 12	网络出错。
error: 13	超时。
error: 14	解码失败。
error: 15	小程序页面传参如果做urlencode需要把整体参数进行编码。
error: 19	HTTP错误。
error: 20	请求已被停止/服务端限流。
error: 23	代理请求失败。

以下几种情况，会返回 error 为 14 的错误（遇到此错误时，请先检查 dataType 的设置是否正确）：

入参 dataType 值	返回 error 14 的原因
JSON	小程序框架对返回结果做 JSON.prase 操作时解析失败。
text	返回的内容格式不符。
base64	转换失败。

若 request 调用返回 **无权调用该接口**，则需要在 阿里云**小程序管理中心 > 设置 > 开发设置 > 服务器域名白名单** 中配置域名白名单。

返回值 RequestTask

网络请求任务对象。调用 `my.request` 后返回的请求对象。

RequestTask.abort()

中断请求任务。

示例代码

// 返回 RequestTask，可以调用 abort 方法取消请求const=.request({:'https://httpbin.org/post'}).abort();

```
 task  myurl
task
```

常见问题

小程序只支持 https 域名配置吗？

小程序只支持 https 域名配置。

相比于 http，https 可以提供更加优质保密的信息，保证了用户数据的安全性，此外 https 同时也一定程度上保护了服务端，使用恶意攻击和伪装数据的成本大大提高。

因此小程序强制使用 https，还在使用 http 协议的开发者需要尽快对服务器进行升级。

为了方便开发者尽快进行 https 的配置，我们提供了免费的 SSL 证书。

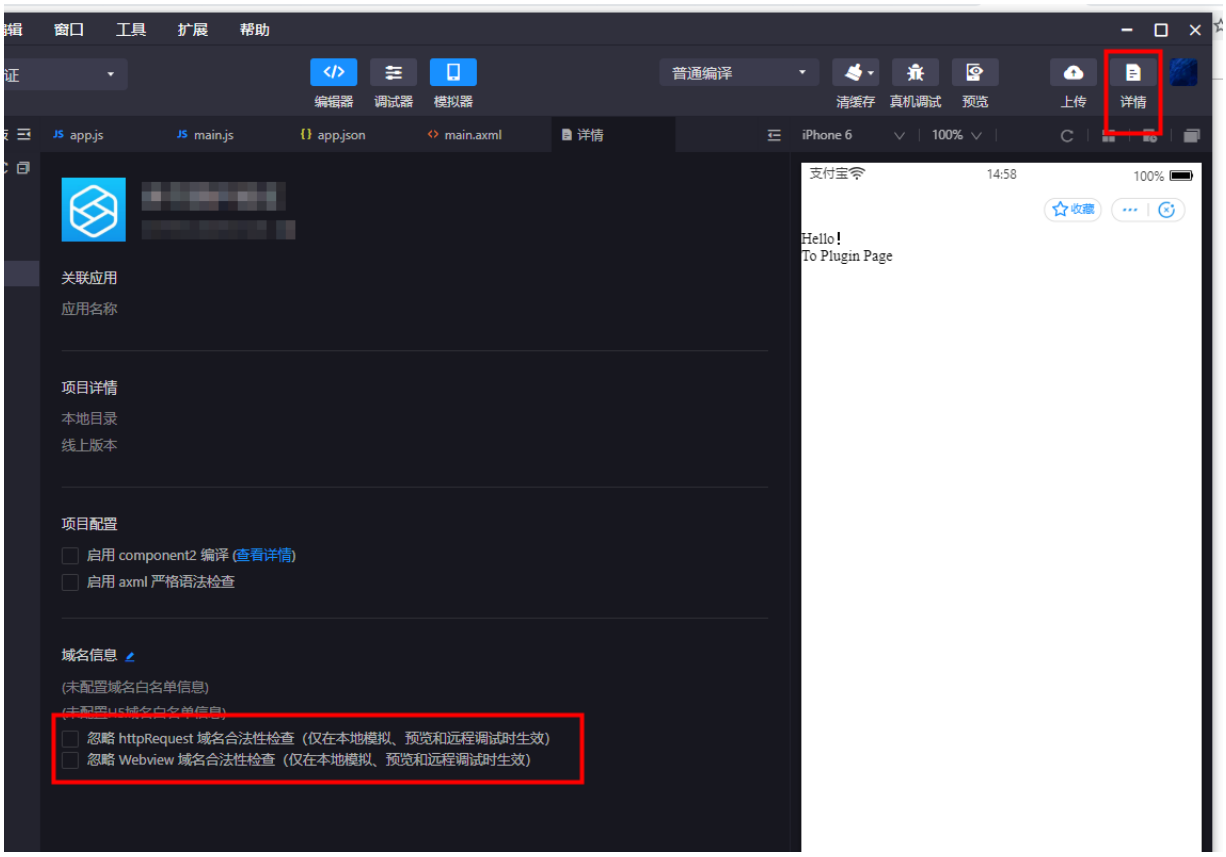
my.request 请求报错“[API 调用] 请配置安全域名”，如何处理？

此报错是未添加服务器域名白名单所致，请检查并确保已添加服务器域名白名单。

my.request 请求报错“无权跨域调用”，如何处理？

步骤一：在小程序管理中心配置服务器域名白名单。

步骤二：若在 IDE 中进行测试，请设置 **忽略 httpRequest 域名合法性检查**。



my.request 请求报错 error: 19，如何处理？

这是 SSL 证书不正确导致的，建议更换网站 SSL 证书。

同时发送多个 https 请求会造成页面卡顿，每秒传输帧数（FPS）低吗？

不会的，这个无影响。

在 IDE 预览模式中测试 my.request 为何没有反应？

在 IDE 中测试 my.requet，需在 IDE 真机模式下，且 my.request 请求地址必须要使用 HTTPS 地址。

添加服务器域名白名单，为什么不生效呢？

添加服务器白名单后需要重新打包上传生成体验版。

RequestTask.abort() 方法是在小程序前端进行请求打断还是直接中止了服务端请求？

RequestTask.abort() 方法只是强制断掉了和后端的链接，并不是阻止请求。

my.request 请求报错 {data: "请求超时异常", error: 14, headers: {...}, status: 13, errorMessage: "JSON parse data error"}，如何处理？

因返回数据格式 text 与入参 dataType 值 JSON 不一致而导致接口报错，请修改后台返回数据格式为 JSON。

小程序后端可以从请求头部，获取到小程序 appId 吗？

小程序后端无法从请求头部获取到小程序 appId。可以通过 my.getAppIdSync 同步获取 appId。

小程序通过 my.request 访问后台，为什么后台获取不到前端 data 中携带参数的值？

建议排查后台获取参数值时，与前端携带参数的key是否保持一致。

使用 my.request 传参，为何参数出现乱码？

您可以选择以下两种方法中的一种进行解决：

- 使用 POST 请求。
- 在前端使用 encodeURIComponent("你好呀")，后端使用URLDecoder.decode(aa, "UTF-8") 解析。

是否可以本地测试小程序请求服务端呢？

不可以本地测试小程序请求服务端。

my.request 可以同步执行吗？

my.request 是异步执行的，无法同步执行。

小程序网络请求支持设置 HTTP REFERER 吗？

小程序会自带 HTTP Referer，暂时不支持自定义设置 HTTP Referer。

my.request 与 my.httpRequest 的区别是什么呢？

1. my.request 是 my.httpRequest 的升级优化，都是小程序端与服务器端交互发起网络请求，体验上没有大的区别。
2. 阿里云客户端已不再维护 my.httpRequest，建议使用 my.request。另外，钉钉客户端尚不支持 my.request。若在钉钉客户端开发小程序，则需要使用 my.httpRequest。
3. my.httpRequest 的请求头默认值为{'content-type': 'application/x-www-form-urlencoded'}。my.request的请求头默认值为{'content-type': 'application/json'}。此外，请求头对象里面的 key 和 value 必须是 String 类型。
4. 版本要求：基础库 1.11.0 或更高版本；阿里云客户端 10.1.32 或更高版本。

my.request 是否支持 IP 地址请求？是否支持 IPv6 域名？

1. my.request 暂不支持 IP 地址请求，不支持 http 请求，请使用 https 请求。
2. my.request 暂不支持 IPv6 域名。
3. my.request 不建议使用关于 127.0.0.1 的测试请求。

my.request 为何显示请求失败，提示 error：2 入参有误？

链接过长导致，建议参数放在 data 中处理。

4.6.13. my.sendSocketMessage

通过 WebSocket 连接发送数据，需要先使用 my.connectSocket 发起建连，并在 my.onSocketOpen 回调之后再发送数据。

示例代码

```
my.sendSocketMessage({
    data: this.data.toSendMessage, // 需要发送的内容
    success: (res) => {
        my.alert({content: '数据发送！' + this.data.toSendMessage});
    },
});
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
data	String	是	需要发送的内容：普通的文本内容 String 或者经 base64 编码后的 String。
isBuffer	Boolean	否	如果需要发送二进制数据，需要将入参数据经 base64 编码成 String 后赋值 `data`，同时将此字段设置为true，否则如果是普通的文本内容 String，不需要设置此字段。

success	Function	否	回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

4.6.14. my.uploadFile

上传本地资源到开发者服务器。

> *使用前提：*
>
> *请预先在 阿里云**小程序管理中心** > 小程序详情 >**设置** > **开发设置** > **服务器域名白名单** 中配置域名白名单。小程序在以下 API 调用时只能与白名单中的域名进行通讯：HTTP请求（my.request）、上传文件（my.uploadFile）、下载文件（my.downloadFile）和 WebSocket（my.connectSocket）。*
>
> 
>

扫码体验

示例代码

```
//  API-DEMO page/upload-file/upload-file.json{    "defaultTitle": "Upload File"}undefined
```

```
<!-- API-DEMO page/upload-file/upload-file.axml --><view class="page">
  <button type="primary" onTap="uploadFile">上传图片</button>
</view>
```

```
// API-DEMO page/API/upload-file/upload-file.js
Page({
  uploadFile() {
    my.chooseImage({
      chooseImage: 1,
      success: res => {
        const path = res.apFilePaths[0];
        console.log(path);
        my.uploadFile({
          url: 'http://httpbin.org/post',
          fileType: 'image',
          fileName: 'file',
          filePath: path,
          success: res => {
            my.alert({ title: '上传成功' });
          },
          fail: function(res) {
            my.alert({ title: '上传失败' });
          },
        });
      },
    });
  },
});
```

上传文件的后端代码。

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

    String path =  req.getParameter("filePath");
    //得到要下载的文件名
    String fileName = URLEncoder.encode(req.getParameter("fileName"),"utf-8");
    String fileType = path.substring(path.lastIndexOf('.')+1,path.length());
    FileInputStream fis = new FileInputStream(path);

    System.out.println("debugFileName: "+ fileName);
    //下载文件存放路径
    String localPath = "";

    FileOutputStream fs = new FileOutputStream(localPath + fileName +"."+fileType);

    resp.setHeader("content-disposition", "attachment;filename="+fileName);
    resp.setHeader("content-type", fileType );

    //执行fileOutputStream的输出操作
    int len = 1;
    byte[] b = new byte[1024];
    while((len=fis.read(b))!=-1){
        fs.write(b, 0, len);
    }
    fs.close();
    fis.close();
}
```

入参

Object 类型，属性如下：

属性	类型	必填	描述
url	String	是	开发者服务器地址。

filePath	String	是	要上传文件资源的本地定位符。
fileName	String	是	文件名，即对应的 key，开发者在服务器端通过这个 key 可以获取到文件二进制内容。
fileType	String	是	文件类型支持图片、视频、音频（image / video / audio）
header	Object	否	HTTP 请求 Header。
formData	Object	否	HTTP 请求中其他额外的 form 数据。
success	Function	否	调用成功的回调函数。
fail	Function	否	调用失败的回调函数。
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

success 回调函数

入参为 Object 类型，属性如下：

属性	类型	描述
data	String	服务器返回的数据。
statusCode	String	HTTP 状态码。
header	Object	服务器返回的 header。

结果码

结果码	描述
11	文件不存在。
12	上传文件失败。
13	没有权限。

UploadTask

版本要求：阿里云客户端 10.1.35 及以上版本，低版本需做 兼容处理。

监听上传进度变化，取消上传任务的对象。

方法

方法	描述
UploadTask.abort()	中断上传任务
UploadTask.onProgressUpdate(function callback)	监听上传进度变化事件

常见问题 FAQ

小程序上传图片可以自动转成 base64 (基于 64 个可打印字符来表示二进制数据的方法)吗?

小程序暂不支持图片转成 base64。

my.uploadFile 如何获取服务器返回的错误信息呢?

解决方案:

1. 可以通过 success 回调中的 data 参数获取。
2. 可以在服务端增加一个日志获取接口。如果上传失败，就请求到日志获取接口获取详细的失败日志。

my.uploadFile 默认超时时间是多少？是否可以设置默认延长时间?

my.uploadFile 默认超时时间是 30s，目前无法设置默认延长时间。

使用 my.uploadFile 上传文件，为何报错 error:12?

上传失败导致报错 error:12 ，造成上传失败的可能原因有:

1. 文件过大
2. 上传时间超过 30s
3. 没有权限

使用 my.uploadFile 上传图片至后台，接收的是二进制图片，再从后台发送小程序前台对应的二进制图片，小程序前台是如何解析呢?

上传图片是后端通过二进制流接受图片，之后后端只需提供对应的图片在服务器上的位置地址就可以。

4.7. 扩展能力

4.7.1. my.aliyun.request

发起网络请求，仅适用于深度集成模式

注意:

- 由于开发者工具版本限制，目前 my.aliyun.request 接口暂不支持在开发者工具调试，仅支持真机预览，请在阿里云客户端扫码查看效果。
- 此 API 暂仅支持深度集成模式的小程序使用。

示例代码

```
my.aliyun.request({
  data: {
    product: 'isvCode',
    action: 'actionName',
    params: JSON.stringify({})
  },
  success: (data) => {
        my.alert({ content: JSON.stringify(data, null, 2)})
  },
  fail: () => {
    // 根据自己的业务场景来进行错误处理
  },
});
```

入参

Object 类型，属性如下：

名称	类型	必填	描述
data	Object	是	请求的入参
success	Function	否	调用成功的回调函数
fail	Function	否	调用失败的回调函数
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）

data

名称	类型	必填	描述
product	String	是	isv的服务code
action	String	是	具体的API名称
params	String	否	请求额外携带的参数

success 返回

名称	类型	描述
data	String	服务端返回的数据

4.7.2. my.aliyun.navigateToAliyunPage

跳转到阿里云客户端内的官方业务或云产品相关页面。

注意：仅能够跳转到阿里云 App 提供的页面

示例代码

```
// 跳转到 ecs 列表页
my.aliyun.navigateToAliyunPage({
    appCode: 'ecsList',
    appParams: {},
    success:(res) => {
        my.alert(JSON.stringify(res));
    },
    fail:(res) => {
        my.alert(JSON.stringify(res));
    }
});

// 跳转到 ecs 详情
my.aliyun.navigateToAliyunPage({
    appCode: 'ecsDetail',
    appParams: {
      instanceId: 'i-xxxxxxxx',
      regionId: 'cn-shanghai'
    },
    success:(res) => {
        my.alert(JSON.stringify(res));
    },
    fail:(res) => {
        my.alert(JSON.stringify(res));
    }
});
```

参数

名称	类型	必填	描述
appCode	String	是	云产品控制台页的标识，目前支持的见下表
appParams	Object	看code	页面所需要的参数
success	Function	否	调用成功的回调函数
fail	Function	否	调用失败的回调函数
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）

success 返回

名称	类型	描述
data	String	服务端返回的数据

appCode 列表

ecsList

ECS的列表页

ecsDetail

ECS的详情页

名称	类型	必填	描述
instanceId	String	是	实例ID
regionId	String	是	实例所在地域

ssh

SSH功能

名称	类型	必填	描述
host	String	是	主机IP
subTitle	String	否	副标题

4.7.3. my.aliyun.getOpenUserInfo

此接口可获取阿里云用户的基础信息（昵称、邮箱、电话、UID）

使用注意

- `my.aliyun.getOpenUserInfo`

  只能在真机上调试（无法在 IDE 进行调试），以获取阿里云用户的基础信息。

示例代码

唤起授权框，推荐兼容方案如下：

```
my.aliyun.getOpenUserInfo({
  success: (res) => {
    my.alert({ content: JSON.stringify(res)});
  },
  fail: (res) => {
    my.alert({ content: JSON.stringify(res)})
  }
});
```

success 回调函数

入参为 res 对象，Object 类型，属性被解析后如下表所示：

名称	类别	类型	描述
aliUid	基础类型	String	阿里云UID
aliyunName	基础类型	String	昵称
email	基础类型	String	邮箱
userPhone	基础类型	String	用户手机号