蚂蚁金服金融科技产品手册

小程序

产品版本： V20200101

文档版本： V20200101

蚂蚁金服金融科技文档

商标声明

及其他蚂蚁金服金融科技服务相关的商标均为蚂蚁金服金融科技所有。
本文档涉及的第三方的注册商标，依法由权利人所有。

免责声明

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。蚂蚁金服金融科技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，并在蚂蚁金服金融科技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蚂蚁金服金融科技授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

目录

1 小程序简介 1
2 快速开发小程序 2
3 接入 Android 6
3.1 接入 Android 6
3.2 Android 小程序接入真机预览与调试 8
3.3 小程序右上角弹出菜单扩展 9
4 接入 iOS 11
4.1 版本 ≥ 10.1.60 12
4.1.1 iOS 小程序基础接入 12
4.1.2 iOS 小程序真机预览与调试 21
4.1.3 10.1.60 升级指南 22
4.2 版本 < 10.1.60 23
5 接入账户通 32
5.1 Android 小程序接入账户通与支付 32
5.2 iOS 小程序接入账户通 38
5.3 账户通接入服务端 45
6 小程序基础库说明 50
7 开发工具 53
7.1 概览 53
7.2 创建/选择环境 55
7.3 登录 57
7.4 编码 59
7.5 模拟器与调试器 62
7.6 真机预览与调试 64
7.7 上传 65
7.8 设置 66
7.9 ESLint 简介 67
8 框架 69
8.1 概述 69
8.2 应用 71
8.3 页面 77
8.4 视图层 82
8.5 事件 94
8.6 样式 98
8.7 小程序全局配置 99
8.7.1 小程序全局配置介绍 99
8.7.2 app.json 全局配置 100
8.7.3 app.acss 全局样式 103
8.7.4 app.js 注册小程序 103
8.7.5 getApp 方法 106
8.8 小程序页面 106
8.8.1 小程序页面介绍 106
8.8.2 页面配置 108
8.8.3 页面结构 108
8.8.4 页面样式 108
8.8.5 页面注册 109
8.8.6 getCurrentPages 方法 119
8.9 AXML 120
8.9.1 AXML 介绍 120
8.9.2 数据绑定 121
8.9.3 条件渲染 125
8.9.4 列表渲染 125
8.9.5 模板 128
8.9.6 引用 130

8.10 SJS 语法参考 . . . 131
8.10.1 SJS 介绍 . . . 132
8.10.2 变量 . . . 134
8.10.3 注释 . . . 135
8.10.4 运算符 . . . 135
8.10.5 语句 . . . 138
8.10.6 数据类型 . . . 140
8.10.7 基础类库 . . . 149
8.10.8 esnext . . . 152
8.11 ACSS 语法参考 . . . 154
8.12 事件系统 . . . 156
8.12.1 事件介绍 . . . 156
8.12.2 事件对象 . . . 157
8.13 自定义组件 . . . 159
8.13.1 自定义组件介绍 . . . 159
8.13.2 创建自定义组件 . . . 160
8.13.3 组件配置 . . . 160
8.13.4 组件模板和样式 . . . 161
8.13.5 组件对象 . . . 166
8.13.6 生命周期 . . . 170
8.13.7 mixins . . . 173
8.13.8 ref 获取组件实例 . . . 174
8.13.9 使用自定义组件 . . . 175
8.13.10 发布自定义组件 . . . 176
8.14 性能优化建议 . . . 178
9 组件 . . . 181
9.1 组件概述 . . . 181
9.2 组件常见问题 . . . 183
9.3 视图容器 . . . 184
9.3.1 view . . . 184
9.3.2 swiper . . . 185
9.3.3 scroll-view . . . 187
9.3.4 CoverView . . . 189
9.3.5 movable-view . . . 190
9.4 基础内容 . . . 191
9.4.1 text . . . 191
9.4.2 icon . . . 192
9.4.3 progress . . . 194
9.4.4 rich-text . . . 195
9.5 表单组件 . . . 198
9.5.1 button . . . 198
9.5.2 form . . . 201
9.5.3 label . . . 203
9.5.4 input . . . 205
9.5.5 textarea . . . 208
9.5.6 radio . . . 211
9.5.7 checkbox . . . 213
9.5.8 switch . . . 215
9.5.9 slider . . . 216
9.5.10 picker-view . . . 218
9.5.11 picker . . . 220
9.6 navigator . . . 223
9.7 媒体组件 . . . 223
9.7.1 图片 . . . 223
9.7.2 视频 . . . 236
9.8 canvas . . . 240
9.9 map . . . 243
9.10 开放组件 . . . 257
9.10.1 web-view . . . 257
10 扩展组件 . . . 260
10.1 概述 . . . 260
10.2 布局导航 . . . 261
10.2.1 列表（list） . . . 261

10.2.2 选项卡（tabs）. . . 263
10.2.3 纵向选项卡（vtabs）. . . 266
10.2.4 卡片（card）. . . 268
10.2.5 宫格（grid）. . . 269
10.2.6 步骤条（steps）. . . 271
10.2.7 页脚（footer）. . . 272
10.2.8 布局（flex）. . . 273
10.2.9 分页（pagination）. . . 275
10.2.10 折叠面板（collapse）. . . 276
10.3 操作浮层. . . 279
10.3.1 气泡（Popover）. . . 279
10.3.2 筛选（Filter）. . . 281
10.3.3 对话框（Modal）. . . 282
10.3.4 弹出菜单（Popup）. . . 284
10.4 结果类. . . 285
10.4.1 异常页（PageResult）. . . 285
10.4.2 结果页（Message）. . . 286
10.5 提示引导. . . 287
10.5.1 提示（Tips）. . . 287
10.5.2 通告栏（Notice）. . . 290
10.5.3 徽标（Badge）. . . 291
10.6 表单类. . . 293
10.6.1 文本输入（InputItem）. . . 293
10.6.2 选择输入（PickerItem）. . . 296
10.6.3 金额输入（AmountInput）. . . 298
10.6.4 搜索框（SearchBar）. . . 299
10.6.5 复选框（AMCheckBox）. . . 301
10.7 手势类. . . 303
10.7.1 可滑动单元格. . . 303
10.8 其他. . . 305
10.8.1 日历（Calendar）. . . 305
10.8.2 步进器（Stepper）. . . 306
10.8.3 图标（AMIcon）. . . 307
11 API. . . 308
11.1 概述. . . 308
11.2 界面. . . 309
11.2.1 导航栏. . . 309
11.2.2 TabBar. . . 318
11.2.3 路由. . . 320
11.2.4 交互反馈. . . 335
11.2.5 下拉刷新. . . 341
11.2.6 联系人. . . 342
11.2.7 选择日期. . . 343
11.2.8 动画. . . 344
11.2.9 画布. . . 347
11.2.10 地图. . . 347
11.2.11 键盘. . . 356
11.2.12 滚动. . . 356
11.2.13 节点查询. . . 356
11.2.14 选项选择器. . . 358
11.2.15 级联选择. . . 361
11.2.16 设置背景窗口. . . 362
11.2.17 设置页面是否支持下拉. . . 363
11.2.18 设置 optionMenu. . . 364
11.3 多媒体. . . 364
11.3.1 图片. . . 364
11.4 缓存. . . 368
11.5 文件. . . 374
11.6 位置. . . 377
11.7 网络. . . 382
11.8 设备. . . 393
11.8.1 canIUse. . . 393
11.8.2 获取基础库版本号. . . 393

11.8.3 系统信息 . . . 394
11.8.4 网络状态 . . . 395
11.8.5 剪贴板 . . . 397
11.8.6 摇一摇 . . . 398
11.8.7 振动 . . . 399
11.8.8 加速度计 . . . 400
11.8.9 陀螺仪 . . . 401
11.8.10 罗盘 . . . 401
11.8.11 拨打电话 . . . 402
11.8.12 用户截屏事件 . . . 402
11.8.13 屏幕亮度 . . . 403
11.8.14 添加手机联系人 . . . 405
11.8.15 扫码 . . . 407
11.8.16 接入蓝牙 . . . 408
11.8.17 蓝牙 API 列表 . . . 412
11.9 数据安全 . . . 428
11.10 分享 . . . 431
11.11 小程序当前运行版本类型 . . . 436
11.12 自定义分析 . . . 437
11.13 自定义 API . . . 438
11.14 小程序跳转 . . . 438
11.15 webview 组件控制 . . . 440
11.16 小程序 API 扩展 . . . 441
11.16.1 跳转支付宝卡包 . . . 441
11.16.2 用户授权 . . . 446
11.16.3 获取会员信息 . . . 449
11.16.4 芝麻认证 . . . 452
11.16.5 唤起支付 . . . 455
12 设计指南 . . . 459
12.1 设计原则 . . . 459
12.1.1 简单清晰 . . . 459
12.1.2 高效贴心 . . . 464
12.1.3 安全可控 . . . 474
12.2 视觉规范 . . . 482
12.2.1 颜色 . . . 482
12.2.2 字体 . . . 483
12.2.3 图标 . . . 483
12.3 组件规范 . . . 484
12.3.1 导航 . . . 484
12.3.2 信息录入 . . . 489
12.3.3 信息展示 . . . 507
12.3.4 交互反馈 . . . 518
12.3.5 手势 . . . 535
12.3.6 平台差异性设计 . . . 536
12.3.7 组件组合 . . . 543
13 代码示例 . . . 549
14 发布小程序 . . . 549

1 小程序简介

背景

支付宝小程序是一种全新的开发模式，融合了 H5 的易开发性、跨平台性、Native 性能，让开发者可以快速开发高性能的页面，提供优异的用户体验。通过使用支付宝小程序，开发者为支付宝开发了大量优质的小程序，丰富了支付宝生态能力。

为了开放更多的生态能力、帮助更多的企业客户快速丰富服务种类，mPaaS 小程序组件应运而生。

组件介绍

mPaaS 小程序是一款基于支付宝小程序的核心技术而独立开发并简化的产品。

通过使用 mPaaS 小程序，开发者不仅可以让支付宝里的小程序运行在利用 mPaaS 开发出来的 App 中，例如让支付宝中的共享单车小程序运行在利用 mPaaS 开发的上海地铁大都会 App 中。同时，开发者还可以利用小程序技术开发自己的 App 页面，并且将这种开发方式作为标准，让开发商为自己的 App 开发小程序，快速建立自己的生态。

mPaaS 小程序和支付宝小程序的区别如下：

- 支付宝小程序只是为支付宝开发小程序，而 mPaaS 小程序可以让开发出的小程序运行在其他客户端里。
- 虽然 mPaaS 小程序继承了支付宝小程序的技术能力，但其目的是为了在开发者自己的 App 中开发自己的小程序，并能够把支付宝小程序快速迁移到自己的 App 中。

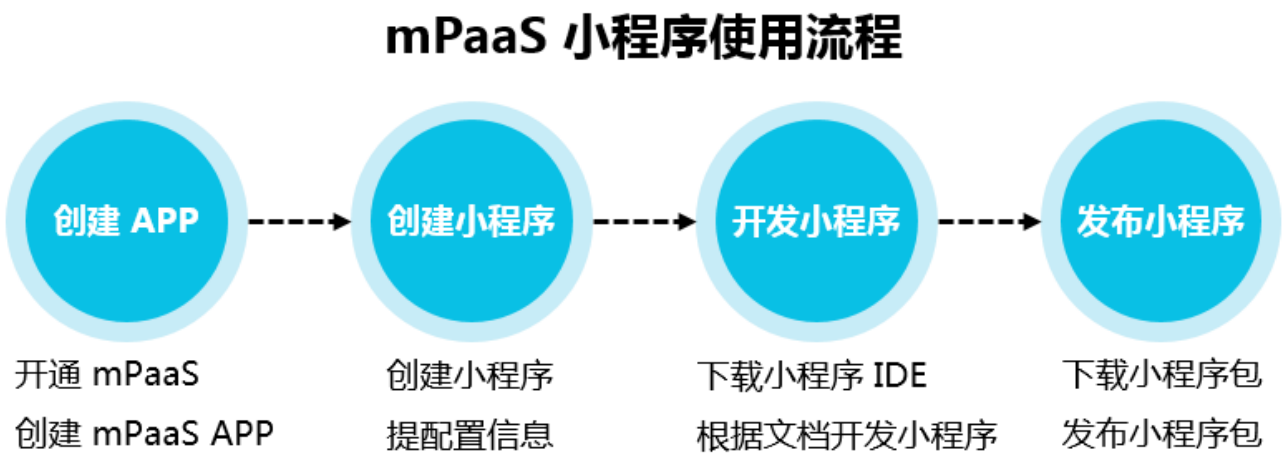
组件功能

mPaaS 小程序包含以下核心功能：

- 集成开发环境（IDE）
- 软件开发工具包（SDK）
- 客户端运行环境
- 小程序发布平台
- 小程序设计规范

使用方法

mPaaS 小程序的使用方法与支付宝小程序类似，如下图所示：



2 快速开发小程序

mPaaS 小程序是手机应用嵌入移动 App 客户端的一种方法，具备以下特点：

- 基于 Web 技术，学习成本低。
- 一套代码，同时支持 iOS 和 Android，接近原生体验。
- 提供丰富的组件和 API，比如获取用户信息、本地存储、支付功能等。

本文向您介绍 mPaaS 小程序的开发链路。通过本指南，您可以在数分钟内快速开发一个小程序：

1. 获取小程序 App ID
2. 安装 IDE
3. 创建项目
4. 编辑代码
5. 本地调试
6. 上传小程序包
7. 发布小程序包

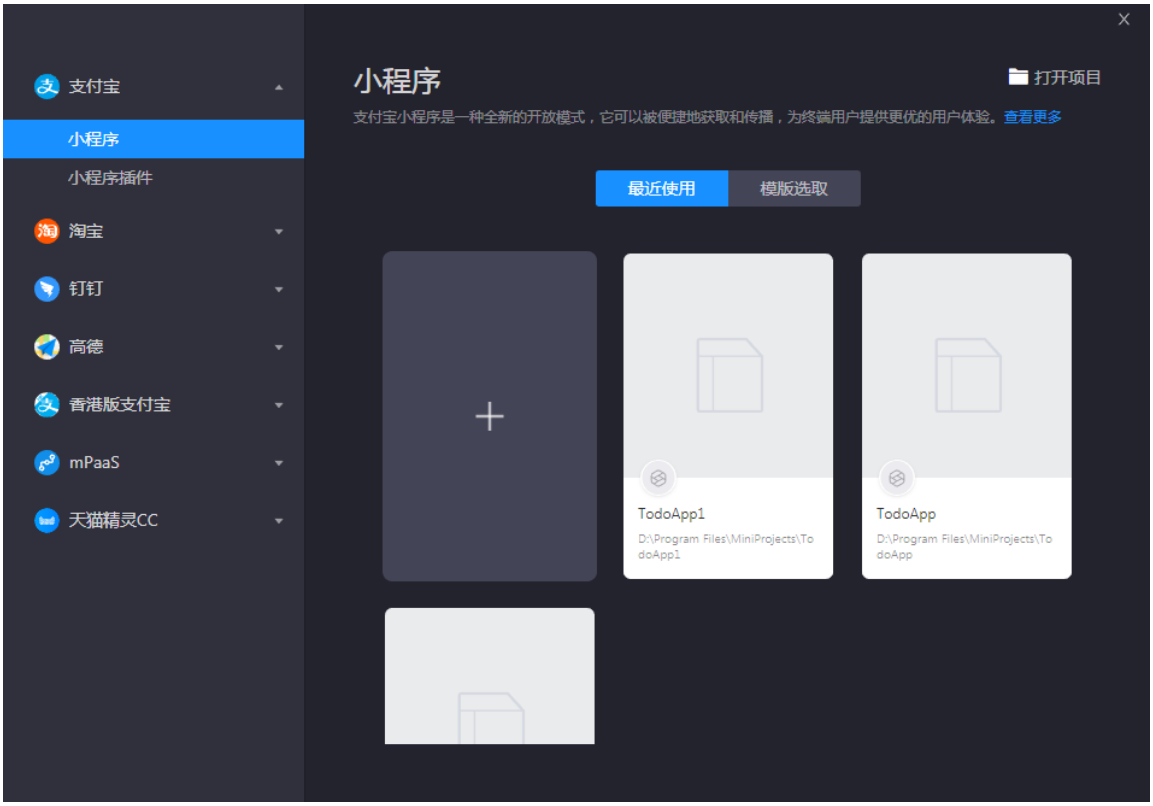
在控制台创建并获取小程序 App ID

1. 进入 mPaaS 控制台，点击左侧导航栏的 **实时发布** > **小程序包管理**。
2. 在 **离线包管理** 标签页中，点击 **小程序包列表** 右侧的 **新建**。

3. 在弹出的 **新建小程序** 窗口中，输入所要创建的小程序的 **ID** 和 **名称**。

安装 IDE

1. 下载 小程序开发工具。小程序开发工具（IDE）是一个辅助开发 mPaaS 小程序的本地应用工具，包含本地调试、代码编辑、上传小程序包等功能，覆盖了应用开发的完整流程。
2. 下载后，解压并安装 IDE。点击 **完成** 后，IDE 自动打开。



创建项目

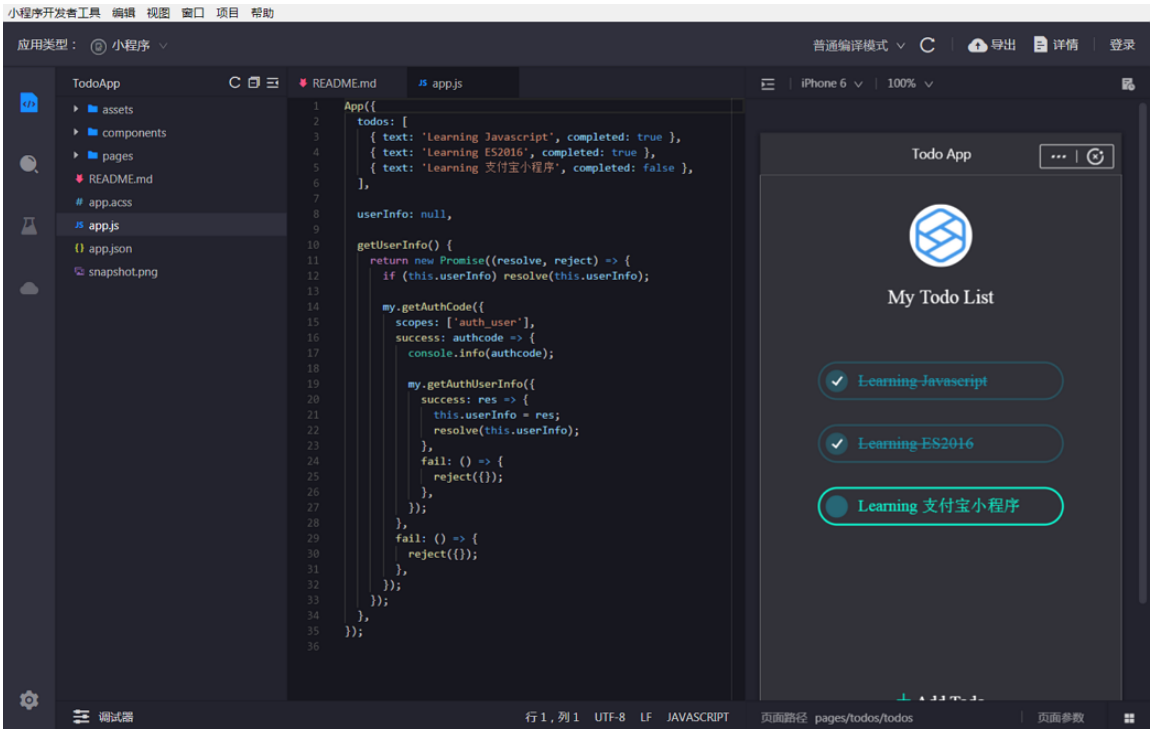
1. 在左侧导航栏，选择 **mPaaS** > **小程序**，您会看到以下界面：

2. 点击 **+** 创建项目。
3. 在 **新建项目** 栏，填写项目名称，选择项目路径，点击 **完成**。

编辑代码

打开项目，会默认进入代码编辑模式。从左到右依次是文件操作区、代码编辑区和预览区。

说明：在代码编辑区，您可以对当前项目编写代码，进行添加、删除以及重命名文件等操作。



2. 在 IDE 窗口右上角，点击 **登录**，使用 mPaaS 账号登录。

实时预览

在代码编辑区修改任何代码都会重新编译，然后自动刷新应用。

自动补全

工具针对 my 接口和 AXML 提供了大量的自动补全提示，以帮助开发者提高效率。

本地调试

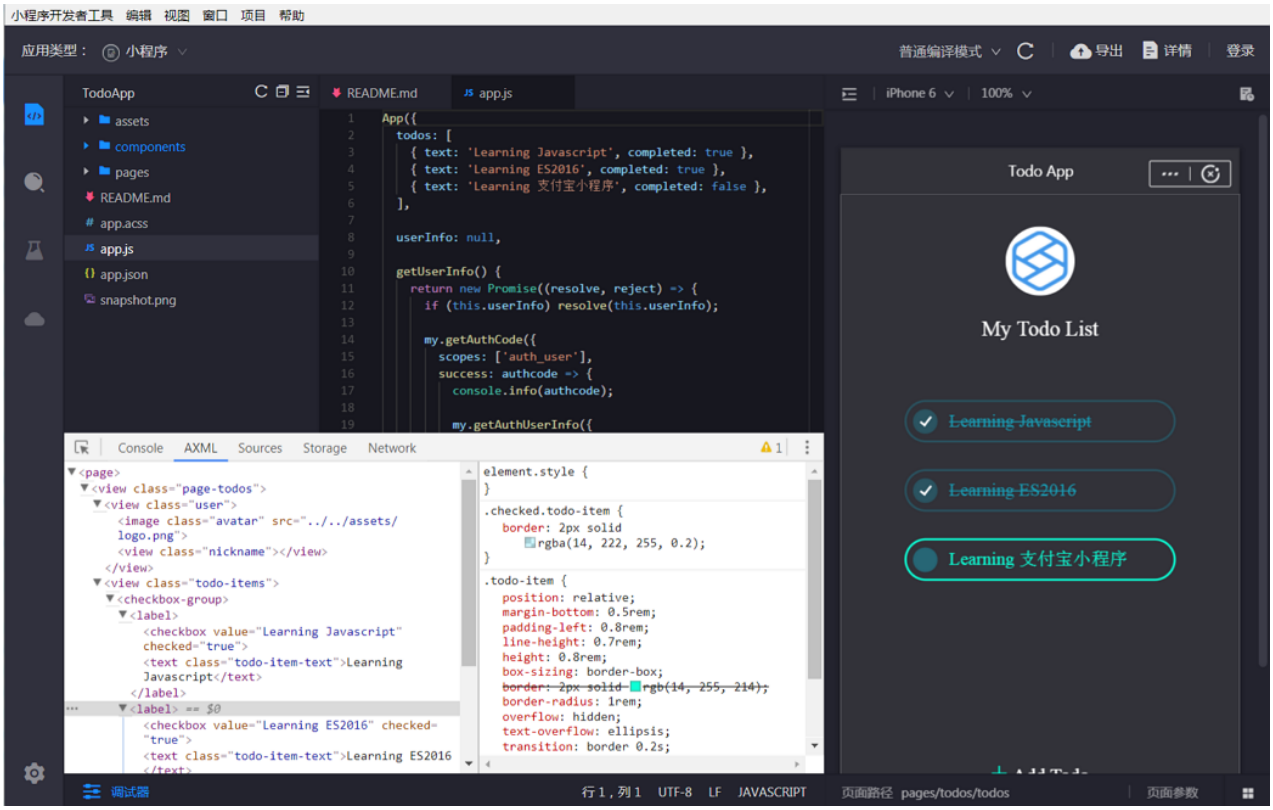
预览区

这里可真实模拟在 mPaaS 应用里的表现，并针对绝大部分的 API 提供了模拟功能。

调试模式

小程序调试工具提供了 AXML 和 acss 的支持，支持组件层级、属性回写等功能。同时，调试工具也包含了 Chrome 调试工具中的网络请求、DOM 元素检查、源码 debug 等。

按下 **Ctrl** + **Shift** + **I** 打开调试工具：



上传小程序包

点击 IDE 右上方的 **发布**，可将小程序代码的 .zip 包上传至 MDS 服务端。更多信息参见 上传 。

发布小程序包

完成小程序开发与上传后，您可以到 **mPaaS 控制台** > **实时发布** > **小程序包管理** 发布小程序。更多信息参见 小程序包管理 。

3 接入 Android

3.1 接入 Android

目前，小程序支持 **基于 mPaaS 框架** 与 **mPaaS Inside** 的接入方式。更多信息，请参考 接入方式简介 。

前置条件

您已参考 通用步骤说明 完成基础配置。

接入步骤

为了接入小程序，您需要完成以下步骤：

1. 添加 UC SDK 。
2. 添加 **小程序** SDK 依赖。
   使用 mPaaS 插件，分别在 Portal 和 Bundle 工程中添加 **小程序（Mini program）** 组件依赖。更

多信息，请参考 管理组件依赖 。

3. 加入小程序依赖的框架资源。
   小程序框架资源需要手动加入，参见 小程序基础库 。
4. 修改开关配置（可选）。
   小程序某些特定功能受配置开关影响，参见 H5 容器配置 。
5. 测试验证小程序。
   在 mPaaS 控制台，点击 **实时发布** > **小程序包管理** 上传并发布小程序文件后，在客户端打开该应用。启动小程序的示例代码如下：

```
String appId ="您的小程序的 appId";
Bundle bundle = new Bundle();
// 不设置就跳转发布小程序时的默认首页
bundle.putString(H5Param.PAGE,"跳转的页面路径");
LauncherApplicationAgent.getInstance().getMicroApplicationContext().startApp(null, appId, bundle);
```

说明：小程序和 H5 共用一套包管理机制。所以如需在客户端管理小程序包，可参考 管理离线包 。

小程序升级说明

SDK 版本升级至 10.1.60 后，变更如下。

API 变化

MPTinyHelper

新增类，用于配置小程序 API、真机预览以及调试所需的必要信息。

- 设置应用名称：小程序 getSystemInfo API 借此获取应用名称。

```
public void setAppName(String name)
```

- 设置应用版本号：小程序 getSystemInfo API 借此获取应用版本号。

```
public void setVersionName(String versionName)
```

- 设置小程序虚拟域名：真机调试依赖此虚拟域名解析请求。

```
public void setTinyAppVHost(String vhost)
```

工程配置要求

com.alipay.android:android-gradle-plugin 的最低版本要求为 3.0.0.8.0。

在工程主 module 的 build.gradle 文件的 portal 字段加入 enableNebulaMetaInfo 和 useMetaInfoClass，示例如下：

```
portal {
// 因篇幅限制，已有配置项在此处省略，请勿删除，也请不要重复添加 portal 配置在 gradle 文件中
enableNebulaMetaInfo true
useMetaInfoClass true
}
```

说明：上述配置对于 inside 模式接入同样适用。

相关链接

- 代码示例
- Android 小程序接入真机预览与调试
- Android 小程序接入账户通与支付

3.2 Android 小程序接入真机预览与调试

说明：仅在 mPaaS 10.1.60 及以上版本中支持。

接入真机预览和调试功能步骤如下：

在 H5 容器配置文件 中加入 h5_remote_debug_host 的值，此值为调试通信的服务器地址，您可在 mPaaS 控制台下载的 小程序 IDE 配置文件 中获得此地址。配置文件示例如下：

```
{
"login_url":"https://mappcenter.cloud.alipay.com/ide/login",
"uuid_url":"http://cn-hangzhou-mproxy.cloud.alipay.com/switch/uuid",
"debug_url":"wss://cn-hangzhou-mproxy.cloud.alipay.com",
"sign":"3decfd66c2924489204b4b0f38a9c228",
"upload_url":"https://mappcenter.cloud.alipay.com/ide/mappcenter/mds"
}
```

设置 h5_remote_debug_host 时，请使用配置文件中的 debug_url 字段，并在末尾加上 /host/, 示例如下：

```
[
{
"key":"h5_remote_debug_host",
"value":"wss://cn-hangzhou-mproxy.cloud.alipay.com/host/"
}
]
```

应用启动或启动小程序前调用 tinyHelper.setTinyAppVHost 方法设置小程序所使用的虚拟域名，示例代码如下：

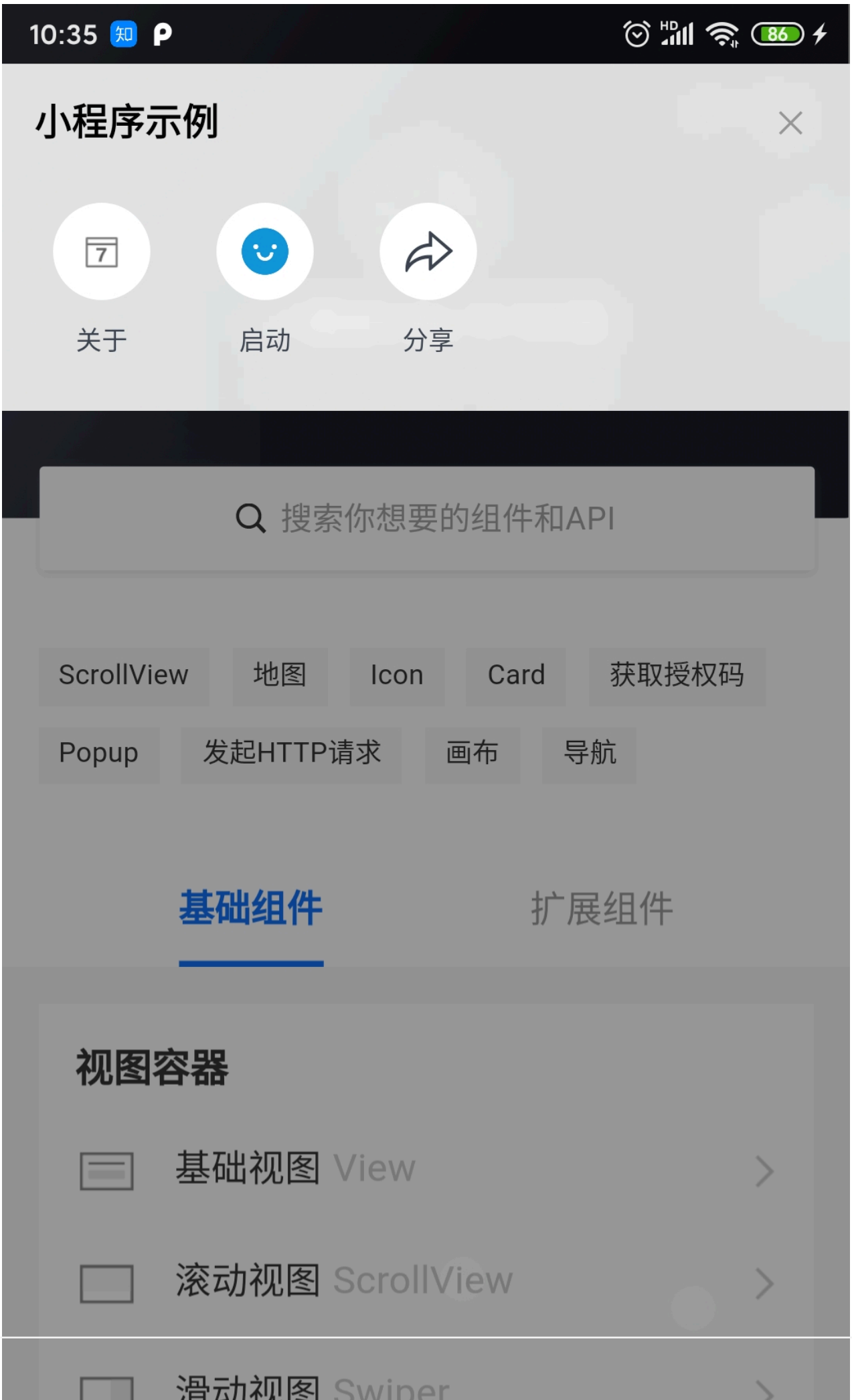
```
MPTinyHelper tinyHelper = MPTinyHelper.getInstance();
tinyHelper.setTinyAppVHost("h5app.com");
```

接入扫码组件并解析预览或调试的二维码，解析二维码并启动小程序的代码如下：

```
// uri 是二维码对应的内容
String scheme = uri.getScheme();
if ("mpaas".equals(scheme)) {
Bundle params = new Bundle();
String appId = uri.getQueryParameter("appId");
for (String key : uri.getQueryParameterNames()) {
if (!"appId".equalsIgnoreCase(key)) {
params.putString(key, uri.getQueryParameter(key));
}
}
LauncherApplicationAgent.getInstance().getMicroApplicationContext()
startApp(null, appId, params);
}
```

3.3 小程序右上角弹出菜单扩展

本文介绍如何在小程序弹出菜单上扩展自定义选项并响应用户点击事件。


10:35
小程序示例
关于
启动
分享
搜索你想要的组件和API
ScrollView
地图
Icon
Card
获取授权码
Popup
发起HTTP请求
画布
导航
基础组件
扩展组件
视图容器
基础视图 View
滚动视图 ScrollView
滑动视图 Swiper

操作步骤

1. 确保 mp_ta_showOptionMenu 配置已开启，详情参考 容器配置 。
2. 使用 TinyPopMenuItem.Builder 类创建 TinyPopMenuItem 对象。
    - Builder 类支持设置选项的名称、图标（支持 url 和 drawable ）及事件回调对象。
    - 调用 Builder 类 setId 方法须保证 ID 唯一性。
3. 事件回调对象的 onClick 方法的第二个参数携带小程序的 appId 以及弹出菜单所在小程序页面的具体路径。
4. 在打开小程序前将 TinyPopMenuProvider 实例对象设置到容器。

代码示例

```
H5Utils.setProvider(TinyPopMenuProvider.class.getName(), new TinyPopMenuProvider() {
@Override
public List<TinyPopMenuItem> fetchMenuItems(String appId) {
List<TinyPopMenuItem> items = new ArrayList<>();
TinyPopMenuItem urlItem = new TinyPopMenuItem.Builder()
.setId("1000")
.setIconUrl("https://gw-office.alipayobjects.com/basement_prod/3d46378a-6e4f-4aa1-820e-fd16da76b457.png")
.setName("关于")
.setCallback(new TinyPopMenuItem.TinyPopMenuItemClickListener() {
@Override
public void onClick(Context context, Bundle bundle) {
String appId = bundle.getString("appId");
String path = bundle.getString("page");
Toast.makeText(context,"应用ID="+ appId +",页面="+ path, Toast.LENGTH_LONG).show();
}
})
.build();
items.add(urlItem);
TinyPopMenuItem localItem = new TinyPopMenuItem.Builder()
.setId("1001")
.setIcon(getResources().getDrawable(R.drawable.smile))
.setName("启动")
.setCallback(new TinyPopMenuItem.TinyPopMenuItemClickListener() {
@Override
public void onClick(Context context, Bundle bundle) {
Toast.makeText(context,"启动"+ bundle.toString(), Toast.LENGTH_LONG).show();
}
})
.build();
items.add(localItem);
return items;
}
});
```

4 接入 iOS

4.1 版本 ≥ 10.1.60

4.1.1 iOS 小程序基础接入

要接入 iOS 小程序，您需要完成以下几大步骤：

1. 添加 SDK ：使用 Xcode 插件添加小程序组件。
2. 配置工程 ：要运行小程序的功能，需进行工程配置。
3. 发布小程序包 ：构建前端 .zip 包，并在控制台发布小程序包。
4. 加载小程序包 ：进入对应的页面时，加载小程序。
5. 预置小程序包 ：除了在线加载小程序包外，您还可以选择将小程序包预置到客户端中。

关于此任务

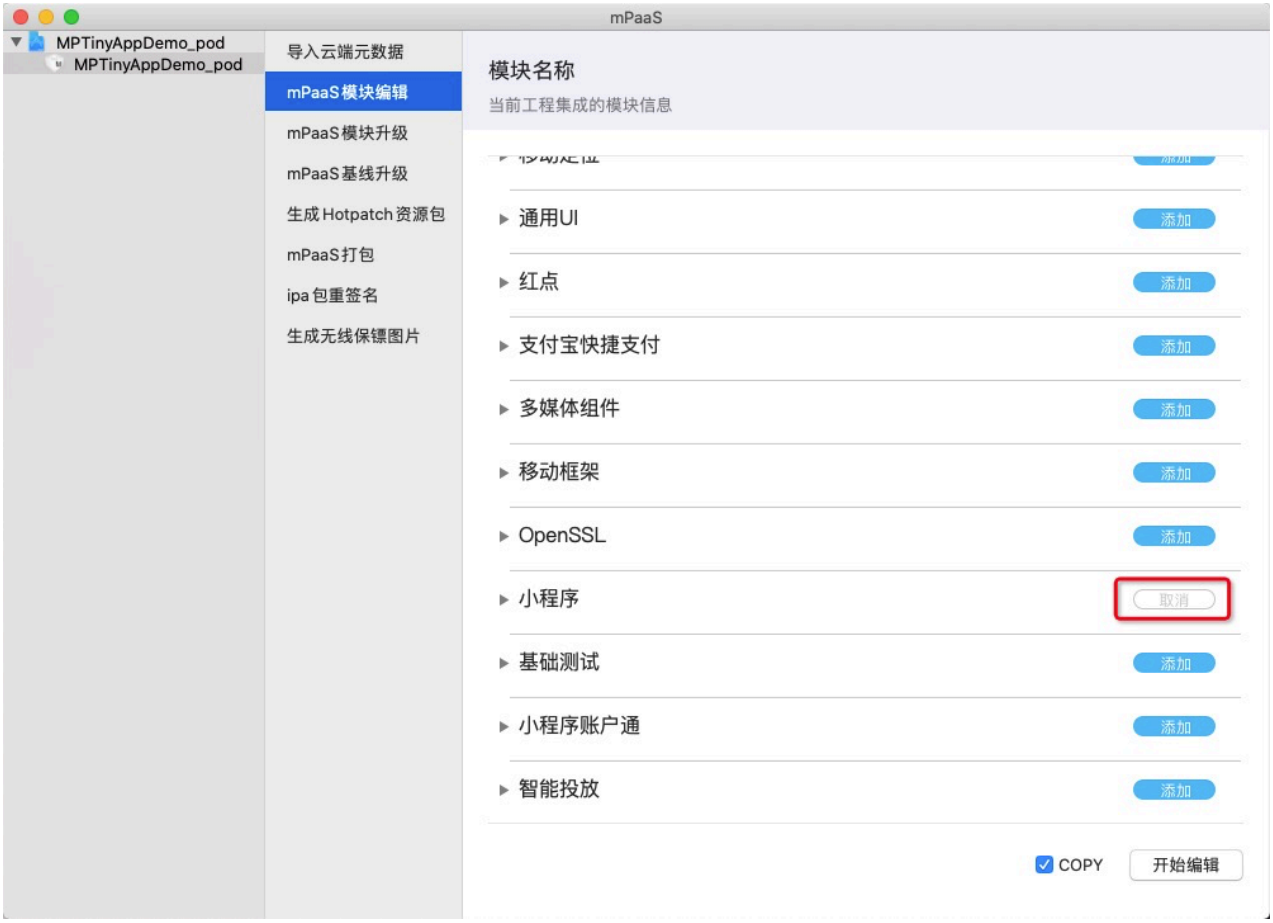
结合 代码示例 ，参照操作步骤，完成小程序接入 iOS 客户端。代码示例中包括小程序的依赖 SDK 和接入的代码示例。

添加 SDK

添加 SDK 分为 插件接入方式 与 cocoapods 接入方式 。

插件接入方式

使用 Xcode 插件添加小程序组件。



小程序自身会提供众多的 JSAPI 和 OpenAPI 能力，因此在插件中选择小程序组件后，相应的依赖组件也会默

认添加到工程中。

cocoapods 接入方式

1. 搭建 mPaaS cocoapods 环境，参见 基于原生框架且使用 CocoaPods 接入 。
2. 按照下方示例修改 podfile，引入小程序。

```
# mPaaS Pods Begin
plugin"cocoapods-mPaaS"
source"https://code.aliyun.com/mpaas-public/podspecs.git"
mPaaS_baseline '10.1.60'
platform :ios, '9.0'
target 'MPTinyAppDemo_pod' do
// 小程序
mPaaS_pod"mPaaS_TinyApp"
end
```

配置工程

在配置工程时，您需要：

- 初始化容器
- 配置小程序

如果您的 App 生命周期并没有交给 mPaaS 框架托管，您还需进行 非框架托管配置 。

初始化容器

启动容器

为了使用 Nebula 容器，您需要在程序启动完成后调用 SDK 接口，对容器进行初始化。必须在 DTFrameworkInterface 的 - (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions 中进行初始化。

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
// 初始化容器
[MPNebulaAdapterInterface initNebula];
}
```

若您需要使用 **预置离线包**、**自定义 JsApi** 和 **Plugin** 等功能，请将上方代码中的 initNebula 替换为下方代码中的 initNebulaWith 接口，传入对应参数对容器进行初始化。

- presetApplistPath：自定义的预置离线包的包信息路径。
- appPackagePath：自定义的预置离线包的包路径。
- pluginsJsapisPath：自定义 JsApi 和 Plugin 文件的存储路径。

```
- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
// 初始化容器
NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"MPCustomPresetApps.bundle/h5_json.json"] ofType:nil];
NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString
stringWithFormat:@"MPCustomPresetApps.bundle"] ofType:nil];
NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"Poseidon-
UserDefine-Extra-Config.plist"] ofType:nil];
[MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath
customPresetAppPackagePath:appPackagePath customPluginsJsapisPath:pluginsJsapisPath];
}
```

说明：initNebula 和 initNebulaWithCustomPresetApplistPath 是两个并列的方法，不要同时调用。

定制容器

如有需要，您可以通过设置 MPNebulaAdapterInterface 的属性值来定制容器配置。必须在 DTFrameworkInterface 的 - (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions 中设置，否则会被容器默认配置覆盖。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
// 定制容器
[MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
[MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
[MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
[MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];
}
```

属性含义如下：

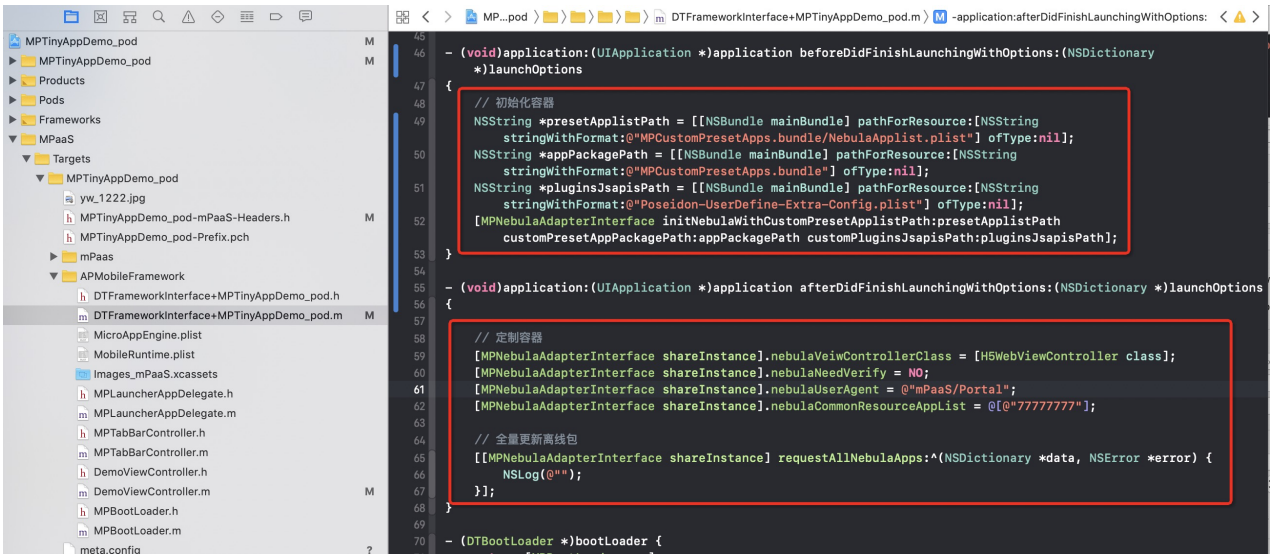
名称	含义	备注
nebulaVeiwControllerClass	H5 页面的基类	默认为 h5webviewcontroller，若需指定所有 H5 页面的基类，可直接设置此接口。
nebulaWebViewClass	WebView 的基类	默认为 UIWebView，若需指定 WebView 的基类，可直接设置此接口。
nebulaUserAgent	当前应用的 UserAgent	-
nebulaNeedVerify	是否验签，默认为 YES	若 配置离线包 时未上传私钥文件，此值需设为 NO，否则离线包加载失败。
nebulaPublicKeyPath	离线包验签的公钥	与 配置离线包 时上传的私钥对应的公钥。
nebulaCommonResourceAppList	公共资源包的 appId 列表	-
errorHtmlPath	当 H5 页面加载失败时展示的 HTML 错误页路径	默认读取 MPNebulaAdapter.bundle/error.html。

更新离线包

启动完成后，全量请求所有离线包信息，检查服务端是否有更新包。为了不影响应用启动速度，建议在 (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions 之后调用。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
// 定制容器
[MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
[MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
[MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
[MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];

// 全量更新离线包
[[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error) {
NSLog(@"");
}];
}
```

初始化完成后，效果如下：

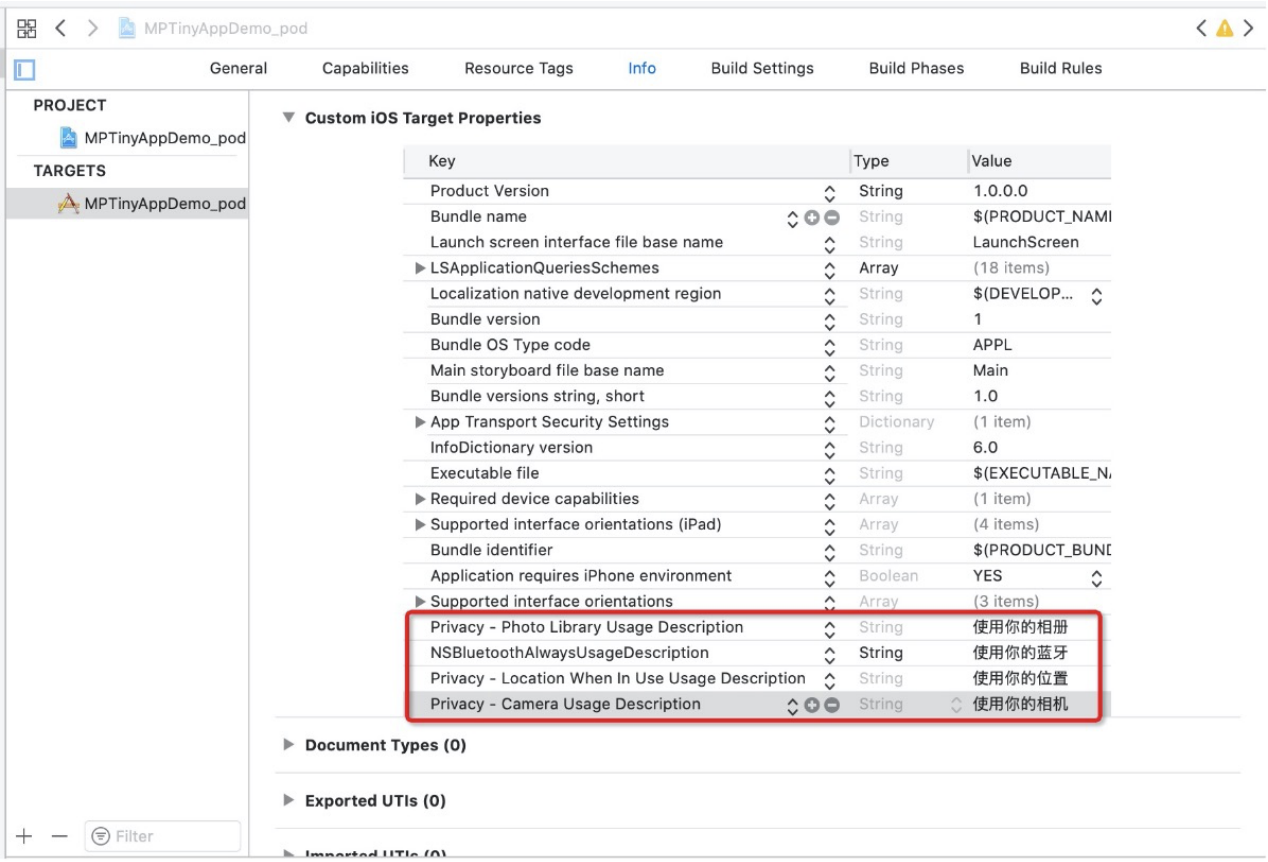


配置小程序

配置权限

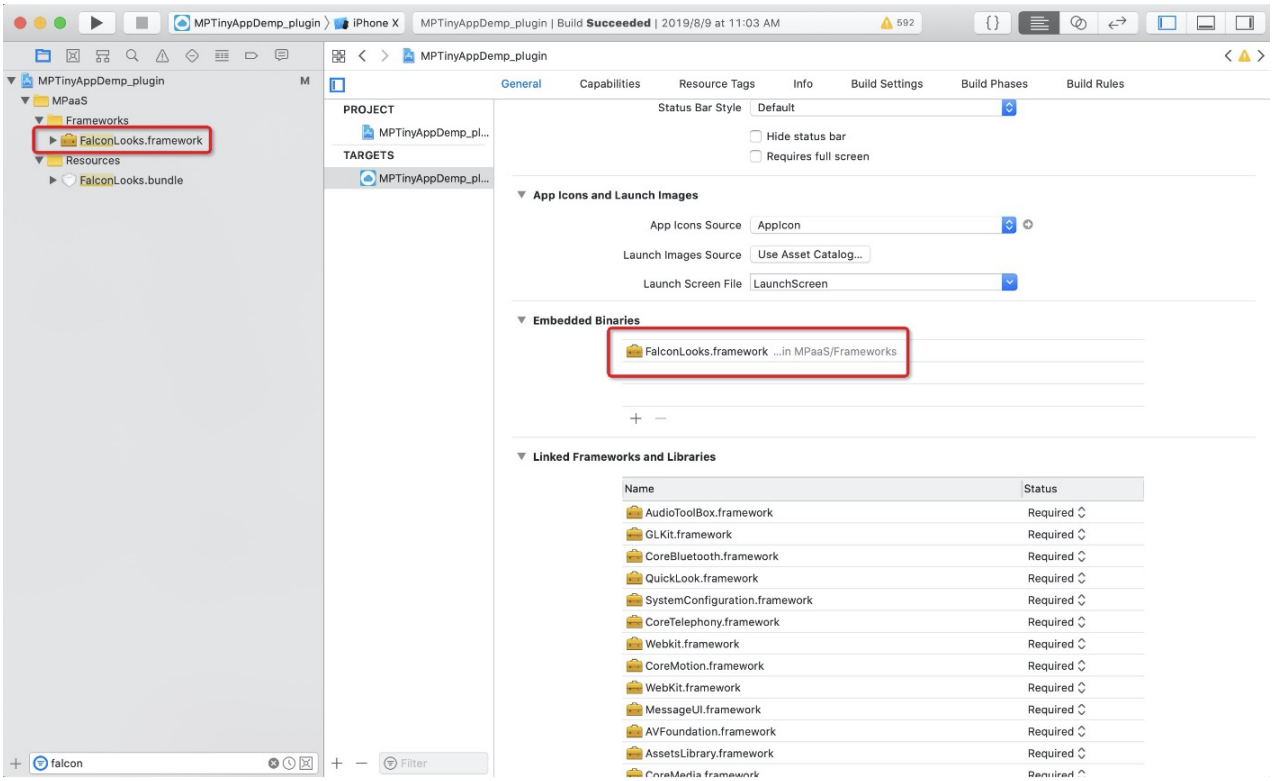
在 info.plist 中配置以下 App 权限：

- NSBluetoothAlwaysUsageDescription：蓝牙权限（iOS 13 中新增）。
- NSCameraUsageDescription：相机权限。
- NSPhotoLibraryUsageDescription：相册权限。
- NSLocationWhenInUseUsageDescription：定位权限。



配置动态库

在当前工程 target 的 **General** > **Embedded Binaries** 中添加 FalconLooks 库。

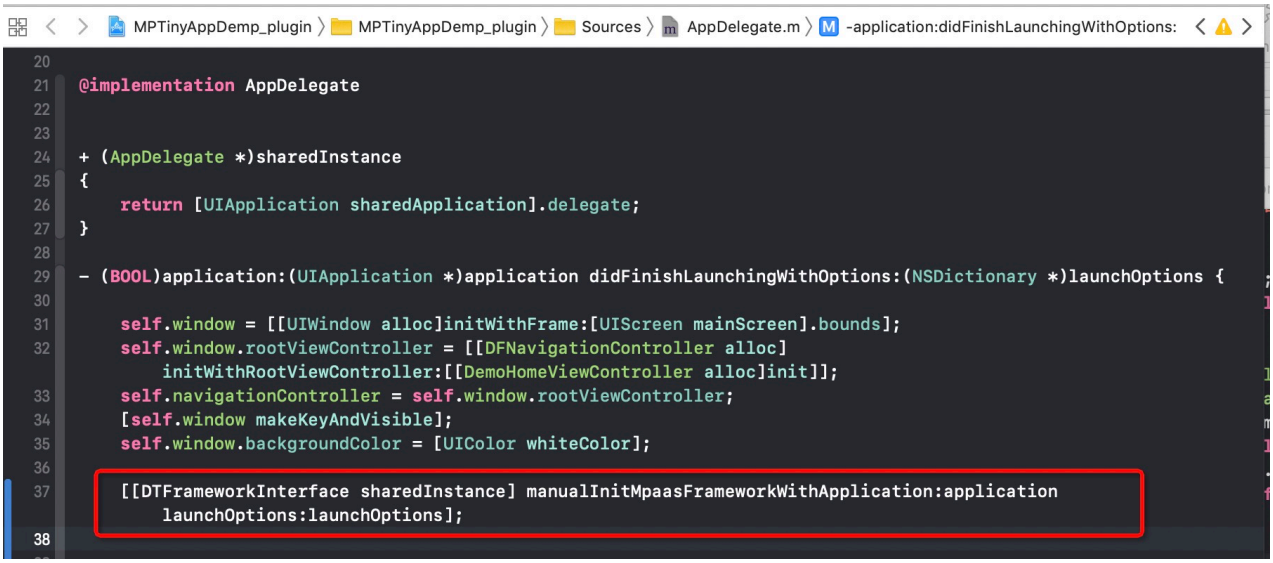


非框架托管配置

如下图所示，若您 App 的生命周期并没有交给 mPaaS 框架托管，而是指定为您自己定义的 delegate，那么您还需进行额外配置。

启动 mPaaS 框架

在当前应用的 didFinishLaunchingWithOptions 方法中调用 [[DTFrameworkInterface sharedInstance] manualInitMpaasFrameworkWithApplication:application launchOptions:launchOptions]; 来启动 mPaaS 框架。



说明：启动框架必须在当前应用 window 和 navigation 初始化完成后调用，否则无法生效。

创建应用启动器

创建继承 DTBootLoader 的子类，重写 createWindow 和 createNavigationController 方法，返回当前应用自己的 window 和 navigationControlle。

- 设置 window：当前应用的 keyWindow。
- 设置 navigationController：当前应用 keyWindow 的 rootviewcontroller，必须继承

DFNavigationController。

MPTinyAppDemp_plugin 〉 MPTinyAppDemp_plugin 〉 Sources 〉 Tinyapp 〉 MPBootLoaderImpl.h 〉 No Selection

```h
//
//  MPBootLoaderImpl.h
//  Portal
//
//  Created by yemingyu on 2019/6/24.
//  Copyright © 2019 Alibaba. All rights reserved.
//

#import <APMobileFramework/APMobileFramework.h>

NS_ASSUME_NONNULL_BEGIN

@interface MPBootLoaderImpl : DTBootLoader

@end

NS_ASSUME_NONNULL_END

```

MPTinyAppDemp_plugin 〉 MPTinyAppDemp_plugin 〉 Sources 〉 Tinyapp 〉 MPBootLoaderImpl.m 〉 No Selection

```m
//
//  MPBootLoaderImpl.m
//  Portal
//
//  Created by yemingyu on 2019/6/24.
//  Copyright © 2019 Alibaba. All rights reserved.
//

#import "MPBootLoaderImpl.h"
#import "AppDelegate.h"

@implementation MPBootLoaderImpl

- (UINavigationController *)createNavigationController
{
    return [AppDelegate sharedInstance].navigationController;
}

- (UIWindow *)createWindow
{
    return [AppDelegate sharedInstance].window;
}

@end

```

指定应用启动器

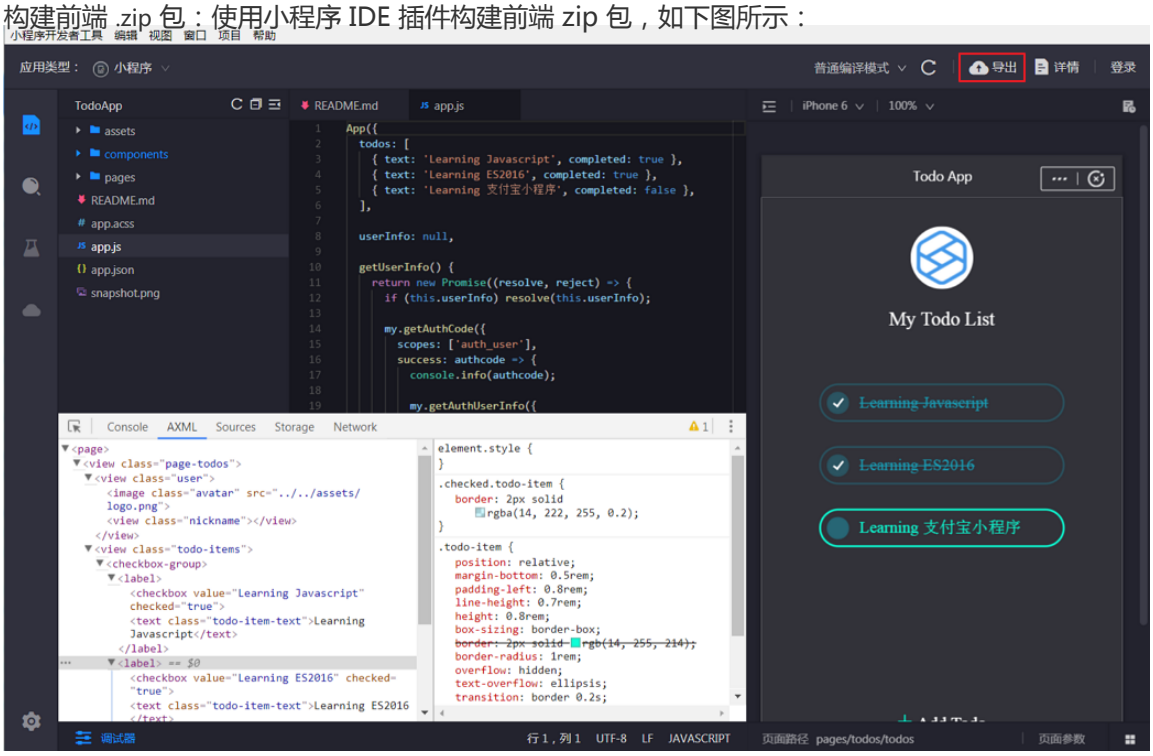
在 DTFrameworkInterface 的 category 中重写方法，指定当前应用自己的 bootloader，并隐藏 mPaaS 框架默认的 window 和 launcher 应用。

```objc
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 全量更新
    [[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error) {

    }];
}

- (DTBootLoader *)bootLoader {
    static MPBootLoaderImpl *_bootLoader;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        _bootLoader = [[MPBootLoaderImpl alloc] init];
    });
    return _bootLoader;
}

- (BOOL)shouldWindowMakeVisable {
    return NO;
}

- (BOOL)shouldShowLauncher {
    return NO;
}

@end

#pragma clang diagnostic pop
```

发布小程序包

小程序包的格式与离线包一样，同样也是 .amr 格式。完成以下步骤生成小程序包：

1. 构建前端 .zip 包：使用小程序 IDE 插件构建前端 zip 包，如下图所示：



2. 发布小程序包：
    - 登录 mPaaS 发布平台，上传生成的 .zip 包。具体操作步骤，参见 创建小程序包 。
    - 在 **小程序包管理** 页面，发布您上传的小程序包。具体操作步骤，参见 发布小程序包 。

加载小程序包

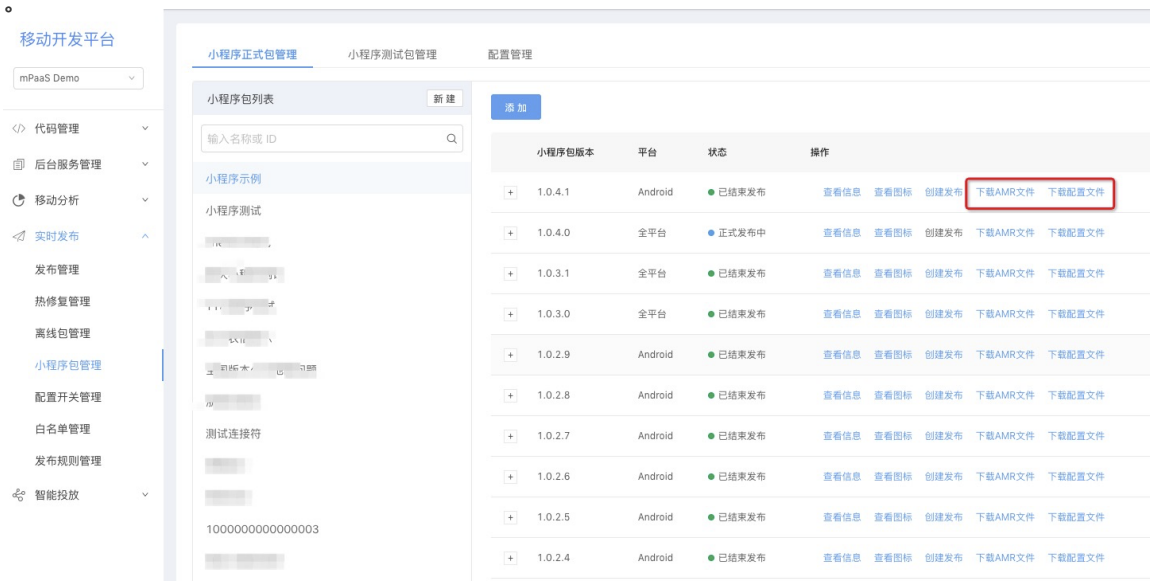
进入对应的页面时，调用框架提供的 startApplication 接口方法加载小程序。

```
[MPNebulaAdapterInterface startTinyAppWithId:appId params:dic];
```

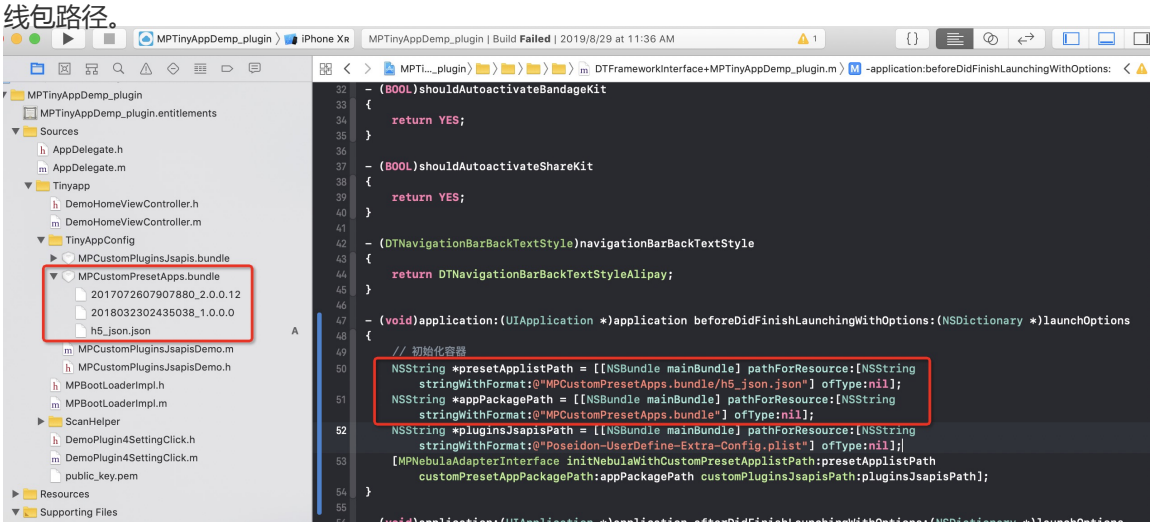
预置小程序包

针对业务上重要且不愿受网络限制的小程序包，您可直接将小程序包预置在客户端中进行加载，而不用去创建发布任务。主要有以下几步：

1. 进入 mPaaS 控制台，在 **实时发布** > **小程序包管理** 中下载您需要预置的小程序 .amr 包和配置文件



2. 将从控制台下载的小程序包和配置文件，添加到工程中，具体路径为您在初始化容器时指定的预置离线包路径。



3. 进入对应的页面时，调用框架提供的 startApplication 接口方法加载小程序。

```
[MPNebulaAdapterInterface startTinyAppWithId:appId params:dic];
```

4.1.2 iOS 小程序真机预览与调试

iOS 小程序接入真机预览与调试

说明：仅在 mPaaS 10.1.60 及以上版本中支持。

按照以下步骤接入预览和调试功能：

1. 根据 IDE 的 二维码 获取二维码内容字符串（如通过扫码）。
2. 调用小程序预览调试接口，传入二维码内容字符串。

```
[MPNebulaAdapterInterface startDebugTinyAppWithUrl:qrCode];
```

iOS 小程序保活

小程序保活指在 App 中打开小程序后，退出小程序但不退出 App 时，小程序会继续存活一段时间，再次打开小程序时，会回到上次退出时的状态，iOS 当前默认保活时间为 60s。

使用

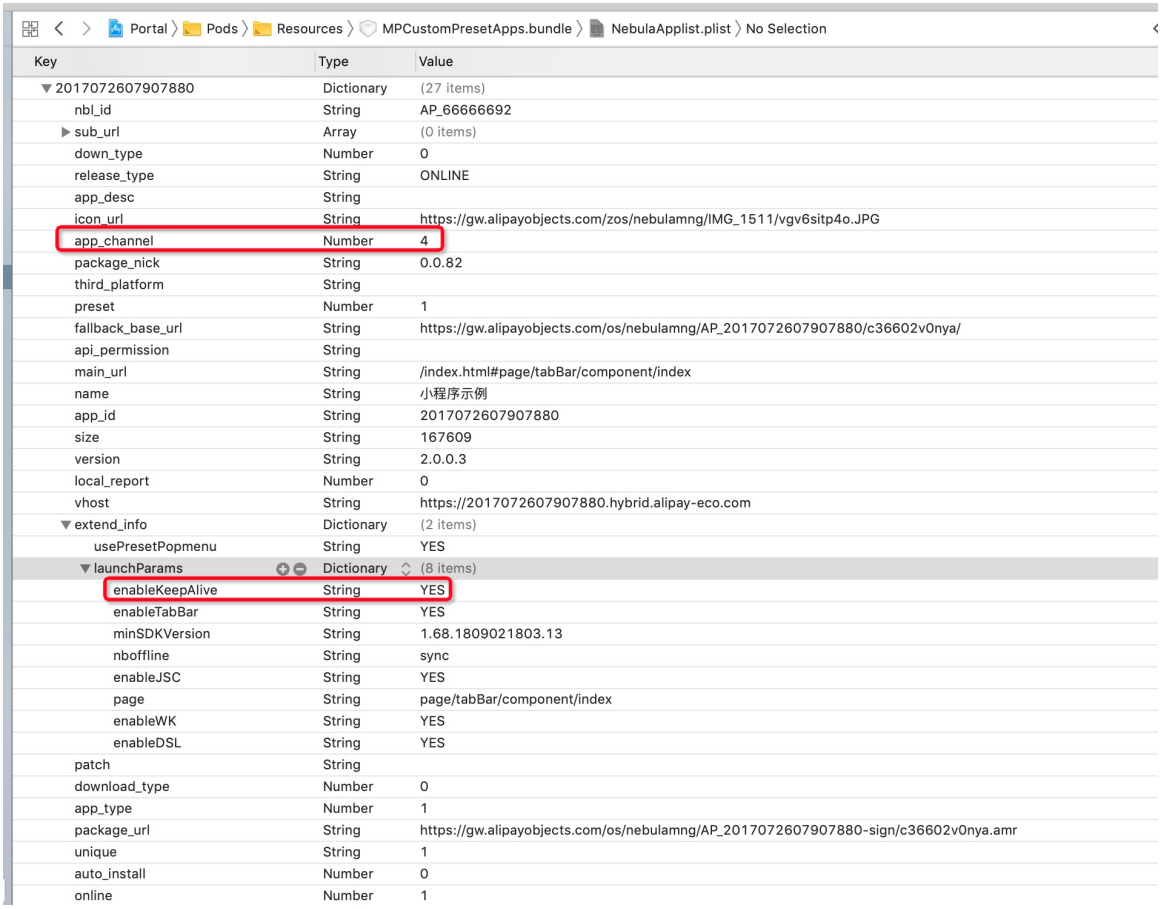
小程序中有一个 **场景** 的概念，指用户进入小程序的路径，场景值则是用来描述该路径的数值。小程序能否实现保活则主要取决于再次打开小程序的场景同退出小程序时的场景是否一致，以及再打开时的时间间隔是否超出了保活时间。

例如：扫码和搜索是两个不同的场景，假设扫码打开的小程序场景值为 A，退出后若再次通过扫码打开小程序则保活生效；但这时如果通过搜索打开小程序，假设搜索场景值为 B，则会清除小程序之前的缓存，此次保活不生效。

- 除了在 mPaaS 控制台配置小程序包时选择开启保活外，调用打开小程序函数时，还需要传入 chInfo，即 **场景值**，保活才会生效，代码示例如下：

```
[MPNebulaAdapterInterface startTinyAppWithId:item[0] params:@{@"chInfo": @"MPPortal_home"}]
```

- 预置包注意检查下面的配置是否正确：



注意事项

- 当账户发生变化时，需要告知小程序容器释放之前账户的保活小程序，需要调用下面的代码：

```
NSDictionary *userInfo = @{@"login_notifaction_changeAccount": @(YES)
};
[[NSNotificationCenter defaultCenter]
postNotificationName:@"APLoginControllerDidFinishNotification"object:nil userInfo:userInfo];
```

说明：如果不通知容器，可能会出现再次打开小程序时，由于保活的存在，会回到之前账户退出小程序时的状态。

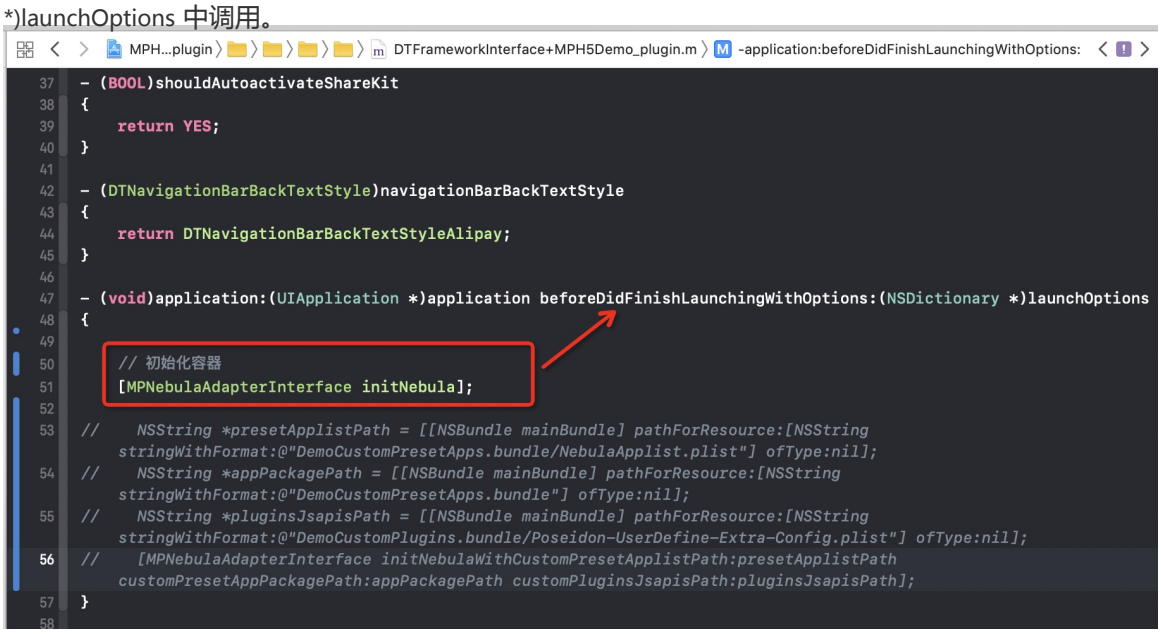
- 接入账户通后，在跳转支付宝授权登录时，由于账户发生变化会自动发出上面的通知，清除当前缓存的小程序，从而导致本次保活失效。
- 谨慎使用保活能力，一旦小程序出现问题，由于保活的存在，会造成可能需要重启 App 才能正常使用小程序。

4.1.3 10.1.60 升级指南

初始化配置

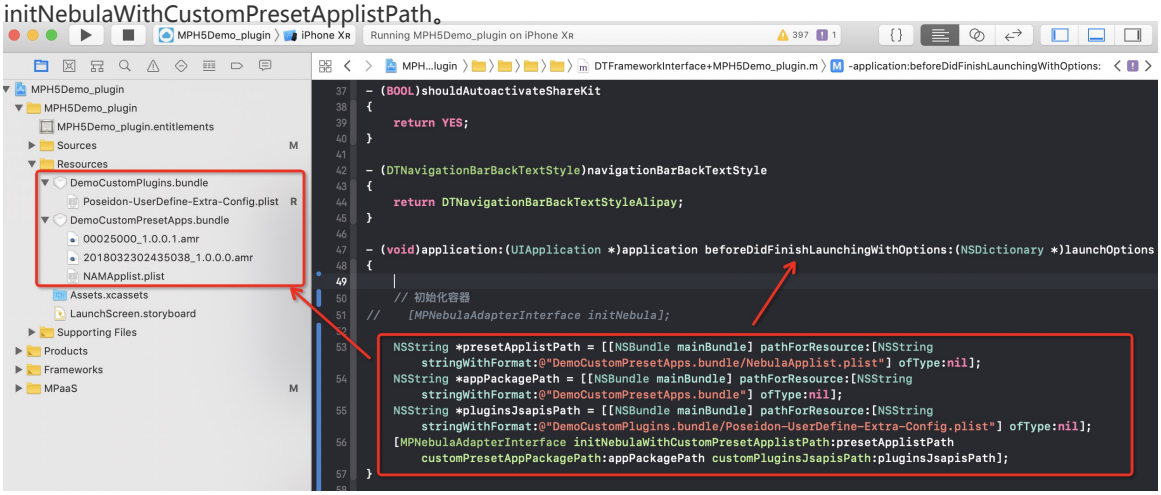
- 初始化时机：需在框架加载之前调用，必须在 DTFrameworkInterface 的 -(void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary

*)launchOptions 中调用。



```
- (BOOL)shouldAutoactivateShareKit
{
    return YES;
}

- (DTNavigationBarBackTextStyle)navigationBarBackTextStyle
{
    return DTNavigationBarBackTextStyleAlipay;
}

- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

    // 初始化容器
    [MPNebulaAdapterInterface initNebula];

//    NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPresetApps.bundle/NebulaApplist.plist"] ofType:nil];
//    NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPresetApps.bundle"] ofType:nil];
//    NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPlugins.bundle/Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
//    [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath customPresetAppPackagePath:appPackagePath customPluginsJsapisPath:pluginsJsapisPath];
}
```

- 若已有工程基线为 10.1.32，需修改自定义 JsApi 路径、预置离线包及包信息路径：
  必须在 DTFrameworkInterface 的 - (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions 中调用 initNebulaWithCustomPresetApplistPath。



```
- (BOOL)shouldAutoactivateShareKit
{
    return YES;
}

- (DTNavigationBarBackTextStyle)navigationBarBackTextStyle
{
    return DTNavigationBarBackTextStyleAlipay;
}

- (void)application:(UIApplication *)application beforeDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

    // 初始化容器
//    [MPNebulaAdapterInterface initNebula];

    NSString *presetApplistPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPresetApps.bundle/NebulaApplist.plist"] ofType:nil];
    NSString *appPackagePath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPresetApps.bundle"] ofType:nil];
    NSString *pluginsJsapisPath = [[NSBundle mainBundle] pathForResource:[NSString stringWithFormat:@"DemoCustomPlugins.bundle/Poseidon-UserDefine-Extra-Config.plist"] ofType:nil];
    [MPNebulaAdapterInterface initNebulaWithCustomPresetApplistPath:presetApplistPath customPresetAppPackagePath:appPackagePath customPluginsJsapisPath:pluginsJsapisPath];
}
```

4.2 版本 < 10.1.60

要接入 iOS 小程序，您需要完成以下几大步骤：

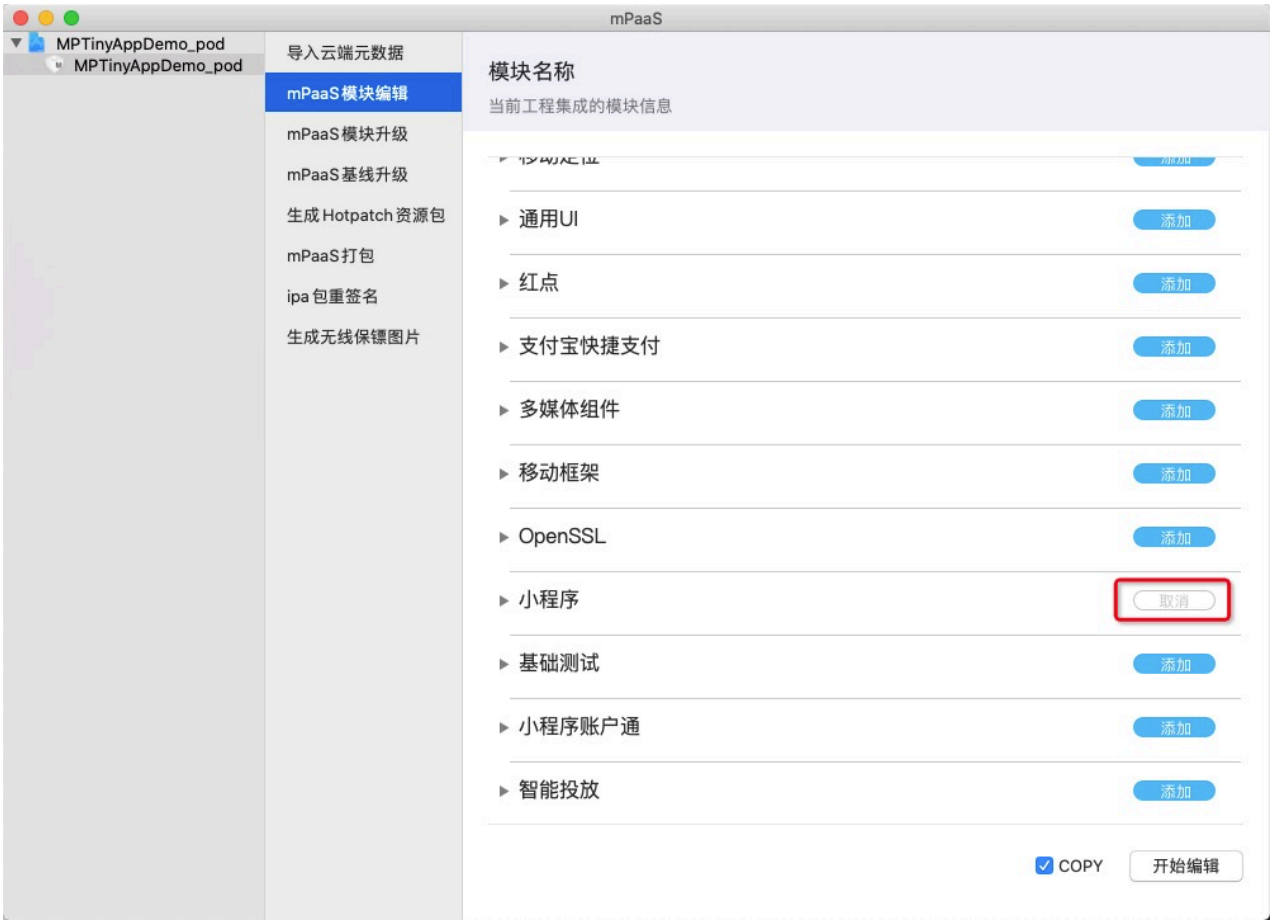
1. 添加 SDK ：使用 Xcode 插件添加小程序组件。
2. 配置工程 ：要运行小程序的功能，需进行工程配置。
3. 发布小程序包 ：构建前端 .zip 包，并在控制台发布小程序包。
4. 加载小程序包 ：进入对应的页面时，加载小程序。
5. 预置小程序包 ：除了在线加载小程序包外，您还可以选择将小程序包预置到客户端中。

添加 SDK

添加 SDK 分为 插件接入方式 与 cocoapods 接入方式 。

插件接入方式

使用 Xcode 插件添加小程序组件。



小程序自身会提供众多的 JSAPI 和 OpenAPI 能力，因此在插件中选择小程序组件后，相应的依赖组件也会默认添加到工程中。

cocoapods 接入方式

1. 搭建 mPaaS cocoapods 环境，参见 基于原生框架且使用 CocoaPods 接入 。
2. 按照下方示例修改 podfile，引入小程序。

```
# mPaaS Pods Begin
plugin"cocoapods-mPaaS", :show_all_specs => true
source"https://code.aliyun.com/mpaas-public/podspecs.git"
#use_pod_for_mPaaS!
mPaaS_baseline '10.1.32' # 请将 x.x.x 替换成真实基线版本
# mPaaS Pods End
platform :ios, '9.0'
target 'MPTinyAppDemo_pod' do
// 小程序
mPaaS_pod"mPaaS_TinyApp"
end
```

配置工程

在配置工程时，您需要：

- 初始化容器
- 配置小程序

如果您的 App 生命周期并没有交给 mPaaS 框架托管，您还需进行 非框架托管配置 。

初始化容器

启动容器

为了使用 Nebula 容器，您需要在程序启动完成后调用 SDK 接口，对容器进行初始化。必须在 DTFrameworkInterface 的 - (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions 中进行初始化。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

// 初始化容器
[MPNebulaAdapterInterface initNebula];
}
```

若您需要使用 **预置离线包**、**自定义 JsApi** 和 **Plugin** 等功能，相关信息只能存储在以下默认 bundle 中，否则无法生效。

名称	含义
MPCustomPlugins.bundle	自定义的 JSAPI 和 Plugin 路径
MPCustomPresetApps.bundle	预置的离线包路径

定制容器

如有需要，您可以通过设置 MPNebulaAdapterInterface 的属性值来定制容器配置。必须在 DTFrameworkInterface 的 - (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions 中设置，否则会被容器默认配置覆盖。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
// 初始化容器
[MPNebulaAdapterInterface initNebula];

// 定制容器
[MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
[MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
```

```
[MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
[MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];
}
```

属性含义如下：

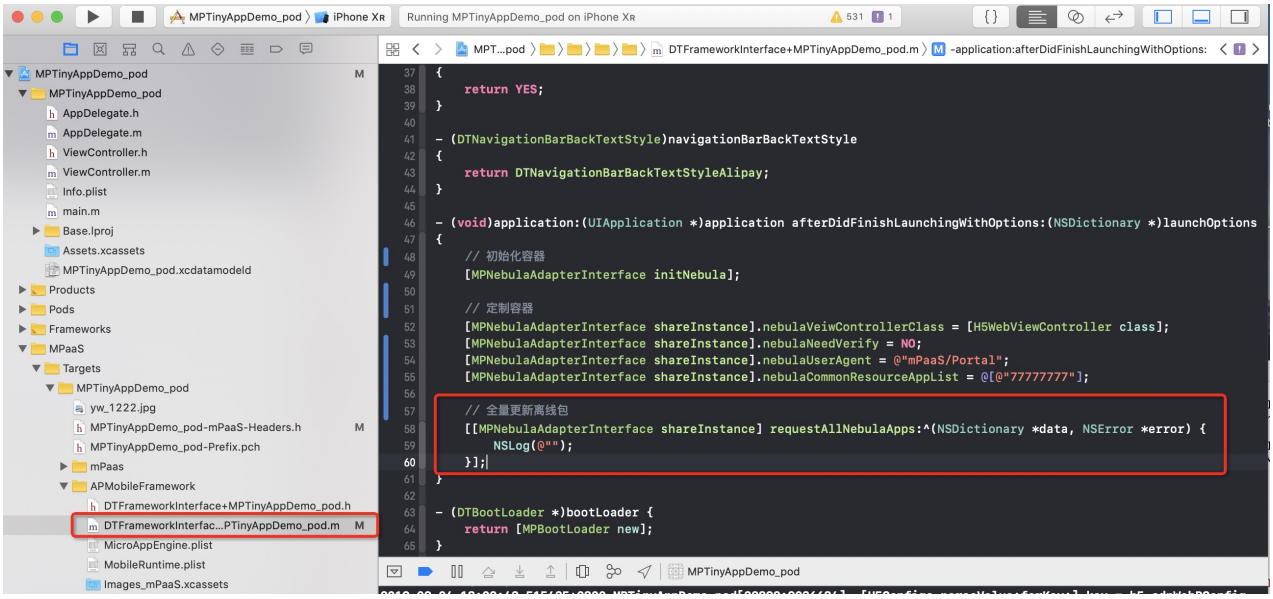
名称	含义	备注
nebulaVeiwControllerClass	H5 页面的基类	默认为 h5webviewcontroller，若需指定所有 H5 页面的基类，可直接设置此接口。
nebulaWebViewClass	WebView 的基类	默认为 UIWebView，若需指定 WebView 的基类，可直接设置此接口。
nebulaUserAgent	当前应用的 UserAgent	-
nebulaNeedVerify	是否验签，默认为 YES	若 配置离线包 时未上传私钥文件，此值需设为 NO，否则离线包加载失败。
nebulaPublicKeyPath	离线包验签的公钥	与 配置离线包 时上传的私钥对应的公钥。
nebulaCommonResourceAppList	公共资源包的 appId 列表	-
errorHtmlPath	当 H5 页面加载失败时展示的 HTML 错误页路径	默认读取 MPNebulaAdapter.bundle/error.html。

更新离线包

启动完成后，全量请求所有离线包信息，检查服务端是否有更新包。为了不影响应用启动速度，建议在 (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions 之后调用

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
// 初始化容器
[MPNebulaAdapterInterface initNebula];

// 定制容器
[MPNebulaAdapterInterface shareInstance].nebulaVeiwControllerClass = [MPH5WebViewController class];
[MPNebulaAdapterInterface shareInstance].nebulaNeedVerify = NO;
[MPNebulaAdapterInterface shareInstance].nebulaUserAgent = @"mPaaS/Portal";
[MPNebulaAdapterInterface shareInstance].nebulaCommonResourceAppList = @[@"77777777"];

// 全量更新离线包
[[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error) {
NSLog(@"");
}];
}
```

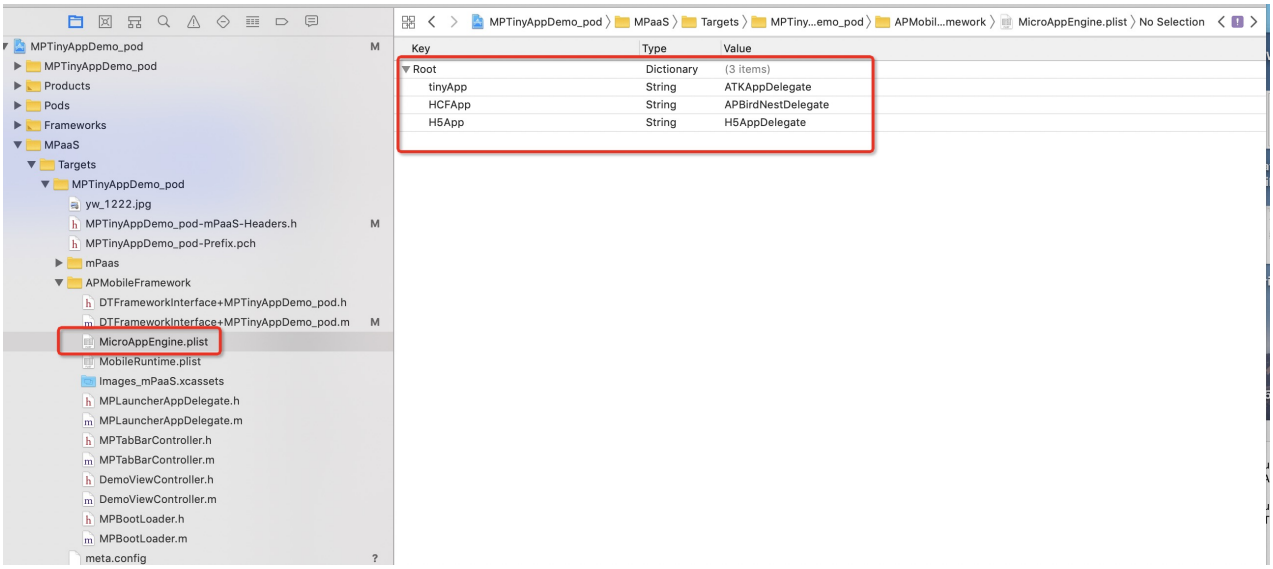
初始化完成后，效果如下：



配置小程序

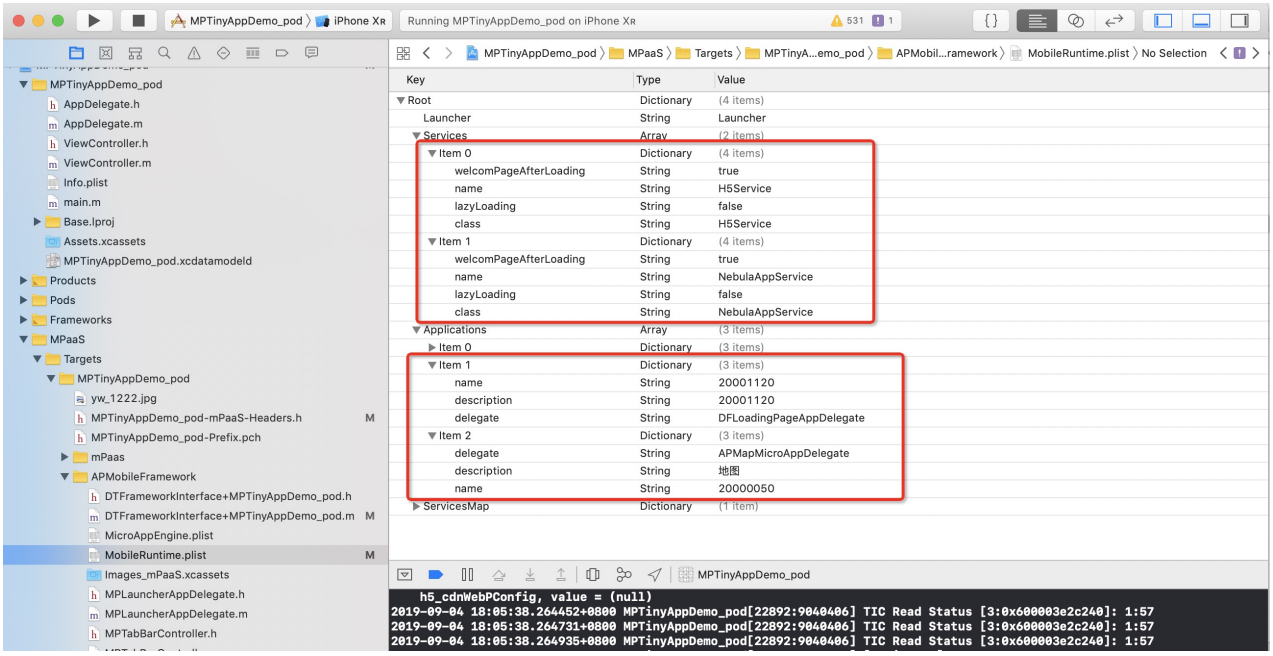
配置 MicroAppEngine.plist

创建 MicroAppEngine.plist，配置小程序内核处理的 App。



配置 MobileRuntime.plist

在 MobileRuntime.plist 中添加小程序依赖的微应用和微服务。

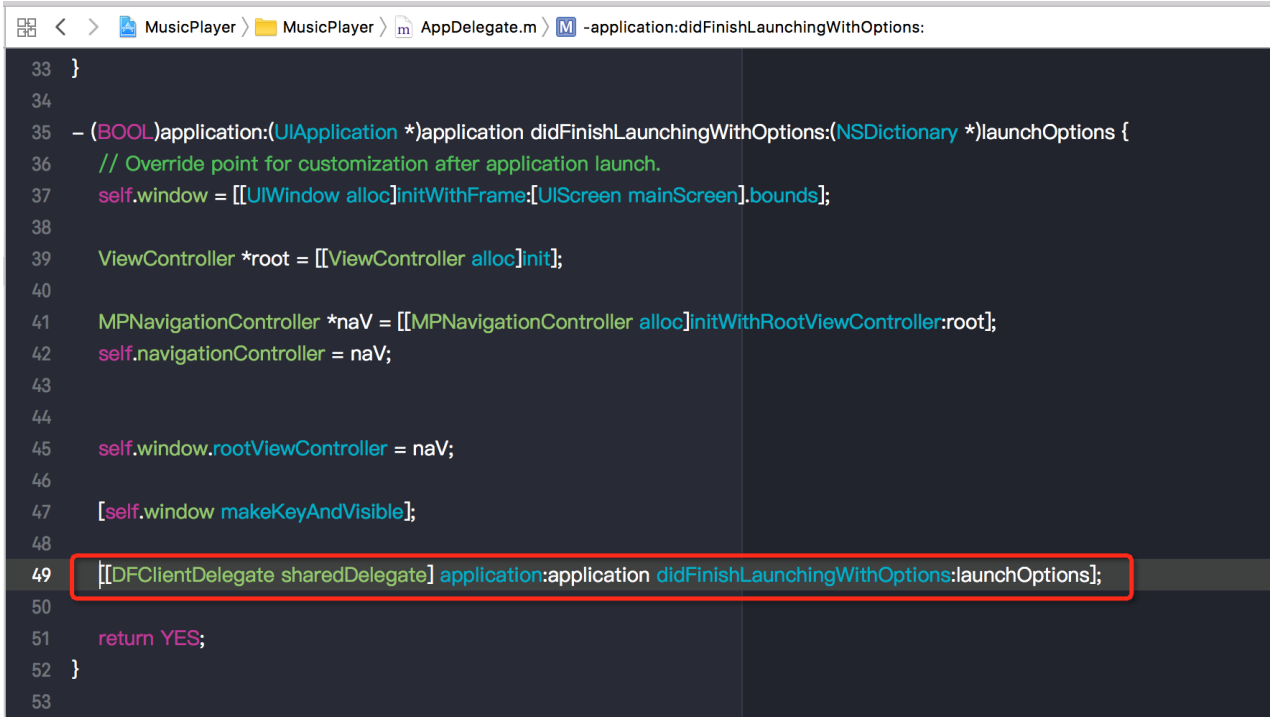


非框架托管配置

如下图所示，若您 App 的生命周期并没有交给 mPaaS 框架托管，而是指定为您自己定义的 delegate，您还需进行额外配置。

启动 mPaaS 框架

在当前应用的 didFinishLaunchingWithOptions 方法中调用 [[DFClientDelegate sharedDelegate] application:application didFinishLaunchingWithOptions:launchOptions]; 来启动 mPaaS 框架。



创建应用启动器

创建继承 DTBootLoader 的子类，重写 createWindow 和 createNavigationController 方法，返回当前应用自己的 window 和 navigationController。

- 设置 window：当前应用的 keyWindow
- 设置 navigationController：当前应用 keyWindow 的 rootviewcontroller，必须继承 DFNavigationController。

MPTinyAppDemp_plugin 〉 MPTinyAppDemp_plugin 〉 Sources 〉 Tinyapp 〉 MPBootLoaderImpl.m 〉 No Selection

```m
//
//  MPBootLoaderImpl.m
//  Portal
//
//  Created by yemingyu on 2019/6/24.
//  Copyright © 2019 Alibaba. All rights reserved.
//

#import "MPBootLoaderImpl.h"
#import "AppDelegate.h"

@implementation MPBootLoaderImpl

- (UINavigationController *)createNavigationController
{
    return [AppDelegate sharedInstance].navigationController;
}

- (UIWindow *)createWindow
{
    return [AppDelegate sharedInstance].window;
}

@end

```

指定应用启动器

在 DTFrameworkInterface 的 category 中重写方法，指定当前应用自己的 bootloader，并隐藏 mPaaS 框架默认的 window 和 launcher 应用。

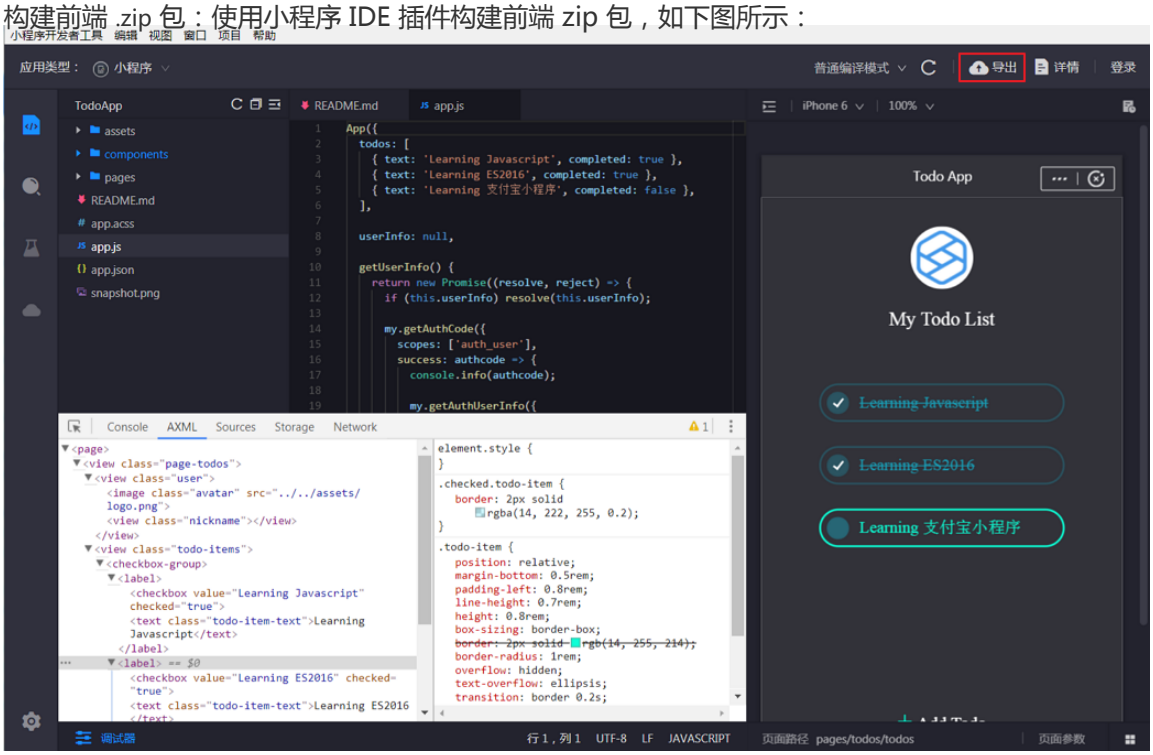
MPTinyAppDemp_plugin 〉 ... 〉 T...s 〉 ... 〉 A...k 〉 DTFrameworkInterface+MPTinyAppDemp_plugin.m 〉 -bootLoader

```m
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // 全量更新
    [[MPNebulaAdapterInterface shareInstance] requestAllNebulaApps:^(NSDictionary *data, NSError *error) {

    }];
}

- (DTBootLoader *)bootLoader {
    static MPBootLoaderImpl *_bootLoader;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        _bootLoader = [[MPBootLoaderImpl alloc] init];
    });
    return _bootLoader;
}

- (BOOL)shouldWindowMakeVisable {
    return NO;
}

- (BOOL)shouldShowLauncher {
    return NO;
}

@end

#pragma clang diagnostic pop

```

发布小程序包

小程序包的格式与离线包一样，同样也是 .amr 格式。完成以下步骤生成小程序包：

1. 构建前端 .zip 包：使用小程序 IDE 插件构建前端 zip 包，如下图所示：



2. 发布小程序包：
    - 登录 mPaaS 发布平台，上传生成的 .zip 包。具体操作步骤，参见 创建小程序包 。
    - 在 **小程序包管理** 页面，发布您上传的小程序包。具体操作步骤，参见 发布小程序包 。

加载小程序包

小程序包发布后，您可以再客户端进行加载。进入对应的页面时，调用框架提供的 startApplication 接口方法加载小程序。

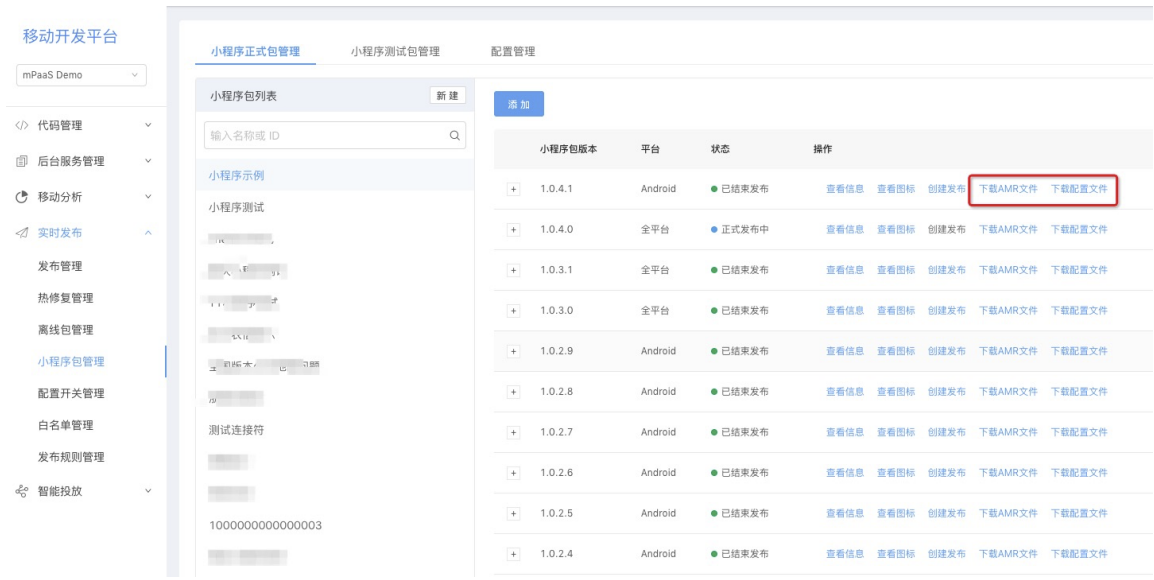
- appId：小程序的 appId
- param：小程序的启动参数，具体内容参见 内置 JSAPI 启动参数

```
[DTContextGet() startApplication:appId params:param animated:YES]
```

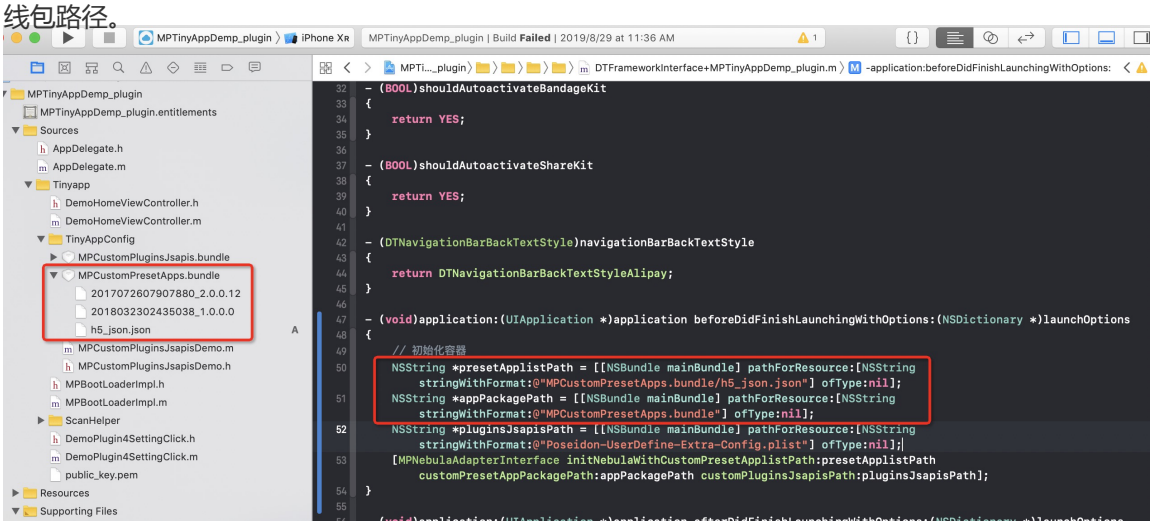
预置小程序包

针对业务上重要且不愿受网络限制的小程序包，您可直接将小程序包预置在客户端中进行加载，而不用去创建发布任务。主要有以下几步：

1. 进入 mPaaS 控制台，在 **实时发布** > **小程序包管理** 中下载您需要预置的小程序 .amr 包和配置文件
    。



2. 将从控制台下载的小程序包和配置文件，添加到工程中，具体路径为您在初始化容器时指定的预置离线包路径。



3. 进入对应的页面时，调用框架提供的 startApplication 接口方法加载小程序。

```
[DTContextGet() startApplication:appId params:param animated:YES]
```

5 接入账户通

5.1 Android 小程序接入账户通与支付

说明：本文档仅适用于版本为 10.1.60 的基线。

接入前准备

在接入前，您需要提供以下信息：

- 在支付宝开放平台申请的 APPID，详情参见 创建应用。

- 应用名称，不可修改。
- 携带签名的 apk 包，如果线上和测试使用的签名不同，请分别提供相应的 apk 包。

接入条件

- 已获得账户通和快捷支付 SDK。

  说明：
  - 在提供上述信息后，您将获得相应的 SDK。
  - SDK 不要集成到 Bundle 工程中。
  - 如果您当前的应用已集成支付宝快捷支付 SDK，请先将其移除。
- 已将基线升级至 10.1.60 版本。
- 已接入小程序。

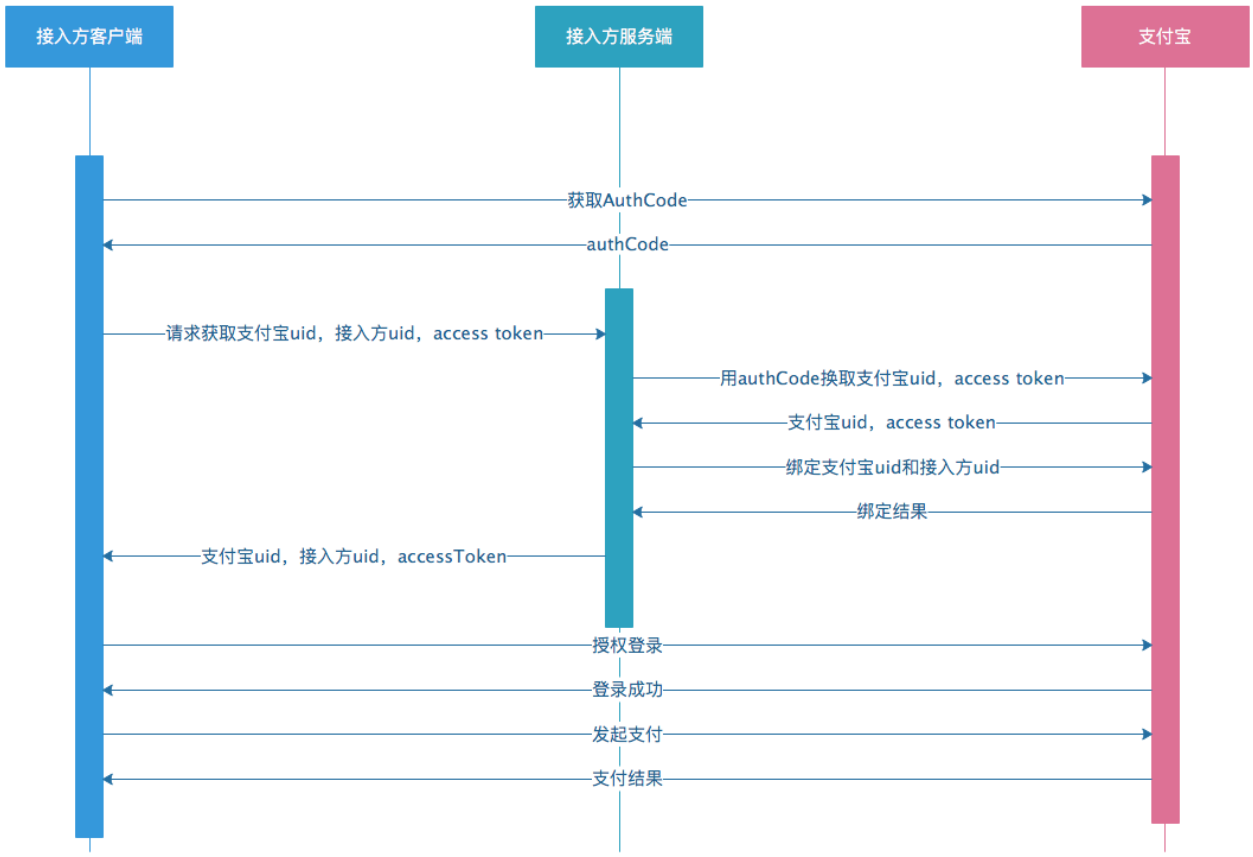
SDK 混淆配置

如果应用开启了混淆编译，需将以下配置加入到混淆配置文件中：

```
# for inside
-keep class com.alipay.android.phone.inside.** { *; }

# for rpc
-keep class com.alipay.inside.** { *; }
-keep class org.json.alipay.inside.* { *; }

# for login
-keep class com.ali.user.** { *; }
-keep class com.alipay.** { *; }

# for minipaysdk
-keep class com.alipay.** { *; }
-keep class com.flybird.** { *; }
-keep class org.iffa.** { *; }

# for securitysdk
-keep class com.alipay.** { *; }

# for securityguard sdk
-keep class com.alibaba.** {*;}
-keep class com.taobao.** {*;}

# for thirdparty lib: okio
-keep class okio.** {*;}

# for thirdparty lib: utdid
-keep class com.ut.** {*;}
-keep class com.ta.** {*;}

# for thirdparty lib: pb
-keep class com.squareup.** {*;}
```

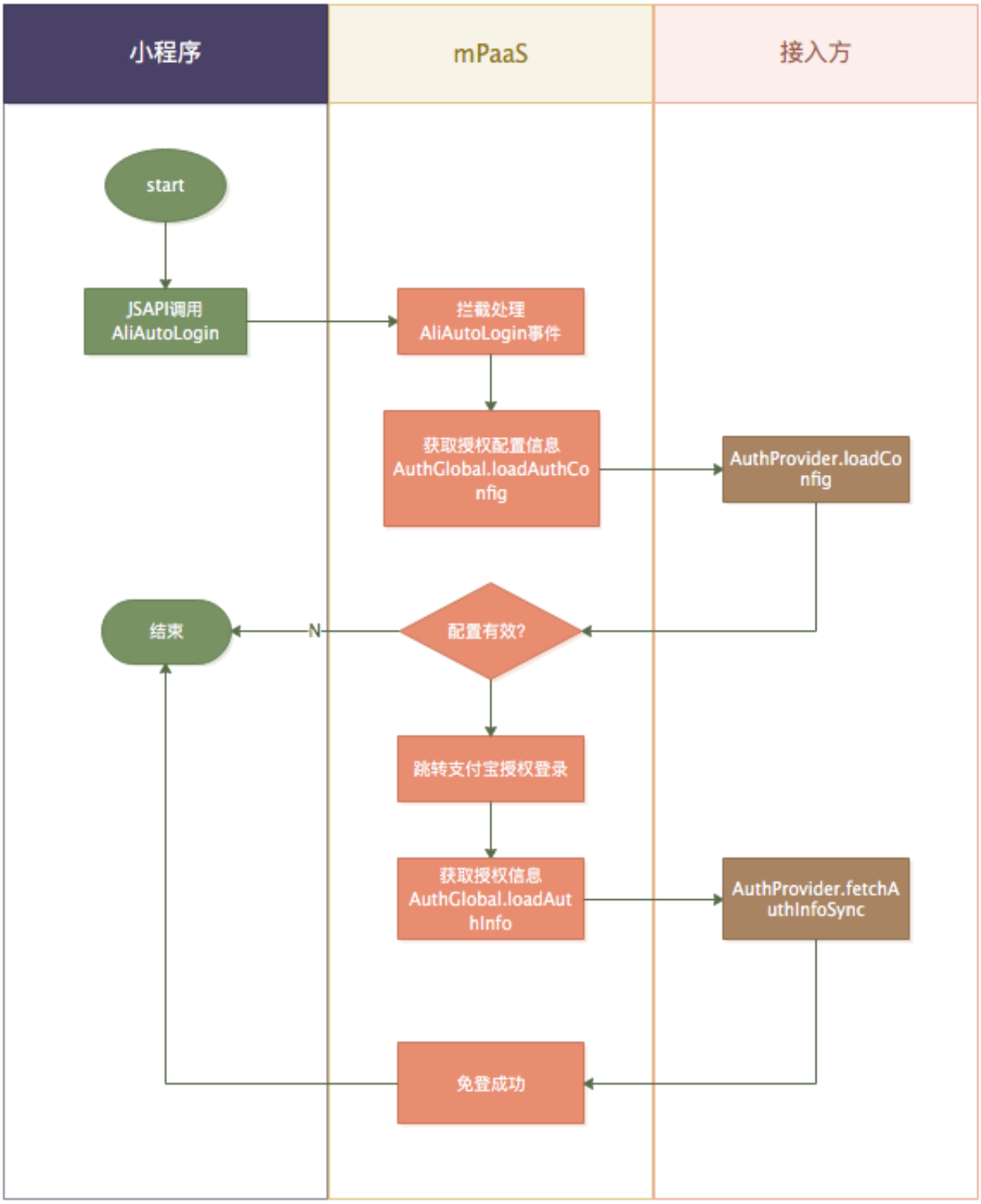
账户授权及支付流程

账户授权及支付流程如下：



客户端授权登录流程

客户端授权登录流程如下：



客户端退出登录或切换用户

接入方退出登录或者切换用户时，需要清除客户端支付宝登录态，否则会造成小程序仍使用前个用户的支付宝登录信息。接入方需在退出登录或者切换用户处调用 AuthGlobal.getInstance().logout 方法。

支付配置

接入方对于支付配置有要求时，请 提交工单提交工单 获取活动标识和业务场景标识。获取标识后，请在客户端

代码中设置，代码示例如下：

```
MPOpenBizHelper.getInstance().setBizSceneCode("小程序应用ID","业务场景标识");
MPOpenBizHelper.getInstance().setCampaignIds("小程序应用ID","活动标识);
```

客户端接入 API

AuthProvider.java

```
package com.mpaas.nebula.adapter.alipay;

/**
* 接入方需实现的接口，用于获取应用的授权配置和授权信息
*/
public interface AuthProvider {

/**
* 获取授权配置
* @return 授权配置
*/
AuthConfig loadConfig();

/**
* 获取授权信息用于客户端免登，一般在此方法需要访问自己的服务端获取信息。
* 此方法运行在非主线程上，因此接入方不需要创建新线程发送网络请求。
* @param authCode 支付宝返回的授权码
* @return 授权信息用于免登
*/
AuthInfo fetchAuthInfoSync(String authCode);

/**
* 获取缓存的授权信息，接入方需要自行缓存支付宝 userId 和 accessToken
* @return 授权信息
*/
AuthInfo getCachedAuthInfo();
}
```

AuthConfig.java

```
package com.mpaas.nebula.adapter.alipay;

/**
* 授权配置，
*/
public class AuthConfig {

/**
* 授权页面请求 url
*/
private String authUrl;
}
```

AuthInfo.java

```
package com.mpaas.nebula.adapter.alipay;

public class AuthInfo {

/**
* 支付宝用户 ID
*/
private final String aliUid;

/**
* 接入端用户统一标识 userId
*/
private final String mcUid;

/**
* 访问令牌
*/
private final String accessToken;

public AuthInfo(String aliUid, String mcUid, String accessToken) {
this.aliUid = aliUid;
this.mcUid = mcUid;
this.accessToken = accessToken;
}
}
```

AuthGlobal.java

```
package com.mpaas.nebula.adapter.alipay;

public class AuthGlobal {

public static AuthGlobal getInstance();

public void setAuthProvider(AuthProvider authProvider);


/**
* 获取支付宝授权auth code
* 必须在子线程中调用
* @param context
* @return
*/
public AuthResult getAuthCode(Context context);

/**
* 清除支付宝登录态或解绑支付宝账户，可在主线程中调用
* @param context
*/
public void logout(Context context, boolean unbindAlipay);
```

示例代码

```
AuthGlobal.getInstance().setAuthProvider(new AuthProvider() {
@Override
public AuthConfig loadConfig() {
return new AuthConfig.Builder()
.setAuthUrl("https://openapi.alipay.com/gateway.do?alipay_sdk=alipay-sdk-java-
3.7.4.ALL&app_id=2019040163782051&biz_content=%7B%22auth_type%22%3A%22MY_PASS_OAUTH%22%2C%22
scopes%22%3A%5B%22auth_user%22%5D%2C%22state%22%3A%2210%22%2C%22is_mobile%22%3A%22true%2
2%7D&charset=UTF-
8&format=json&method=alipay.user.info.auth&return_url=http%3A%2F%2Fzhanghutong.yuguozhou.online%2Ffirs
t&sign=RHLcR%2BbfgW50JgNr5e6MTT08Bnnb3%2Fyt%2B0YIObm%2Fdpq2yJtYzHKgmS2ciVrgFEk6DUKtEmipoLb8
xJ8ErFQAtSS7p8AvXGGY63D95N4lm6yasUVCg2kGoofeB9OPk7GBkLkud1CY3oCbK4HgbHHnHIc43GtXuKt0QLMPivZj
Kgqb5u1zt%2FKscdCt8JrLG4L5vOOFGKRuh3cFq%2BVL%2Bdvaufwbut6B%2B85GjOsnvONICif8r9cxpdzlsRFoSVmYu
%2F7AUM34diatlQPvKs5NOeeAg2W8QkBbQYza0f84KYrNAAeX9ITbzvc7ntiL9606qEB1OWj%2Flccm%2B1TSKQjUUjj
C6A%3D%3D&sign_type=RSA2&timestamp=2019-04-28+17%3A28%3A04&version=1.0")
.build();
}
@Override
public AuthInfo fetchAuthInfoSync(String authCode) {
try {
URL url = new
URL("http://zhanghutong.yuguozhou.online/first?isv_app_id=2019040163782051&app_id=2019040163782051&aut
h_code="+ authCode
+"&state=10&scope=auth_user");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("GET");
connection.connect();
int responseCode = connection.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK) {
InputStream inputStream = connection.getInputStream();
final String resp = readStream(inputStream);
if (null != resp) {
JSONObject jsonObject = JSON.parseObject(resp).getJSONObject("alipay_system_oauth_token_response");
String aliUid = jsonObject.getString("user_id");
String mcUid = jsonObject.getString("mc_user_id");
String accessToken = jsonObject.getString("access_token");
return new AuthInfo(aliUid, mcUid, accessToken);
}
}
} catch (Exception e) {
e.printStackTrace();
}
return null;
}

@Override
public AuthInfo getCachedAuthInfo() {
return new AuthInfo(aliUid, mcUid, accessToken);
}
});
```

5.2 iOS 小程序接入账户通

账户通组件的作用是使支付宝内部小程序在客户 App（即使用 mPaaS 框架开发的 App） 中运行时，可通过

scheme 协议跳转支付宝授权，获取支付宝登录态，进而正常使用各项功能。例如：1688 小程序在客户 App 中通过账户通组件获取支付宝登录态，进而可以进行购买等业务操作。

在跳转时，会根据用户手机上安装支付宝 App 与否出现以下处理流程：

- 当使用客户 App 的用户手机上同时也安装了支付宝 App 时：
  - 客户 App 在需要支付宝登录态时会通过 scheme 协议跳转打开支付宝申请授权。
  - 授权成功后，回到客户 App 就可以正常使用了。
- 如果没有安装支付宝，则需要注册或者在小程序中输入账号密码登录。

前置条件

您已按照小程序接入文档接入小程序框架，并且各项功能验证正常。

说明：目前账户通支持的是小程序中获取支付宝授权登录，插件接入账户通时不要选择接入快捷支付组件，账户通依赖中有快捷支付的解决冲突问题版本。

通用接入配置

添加 SDK

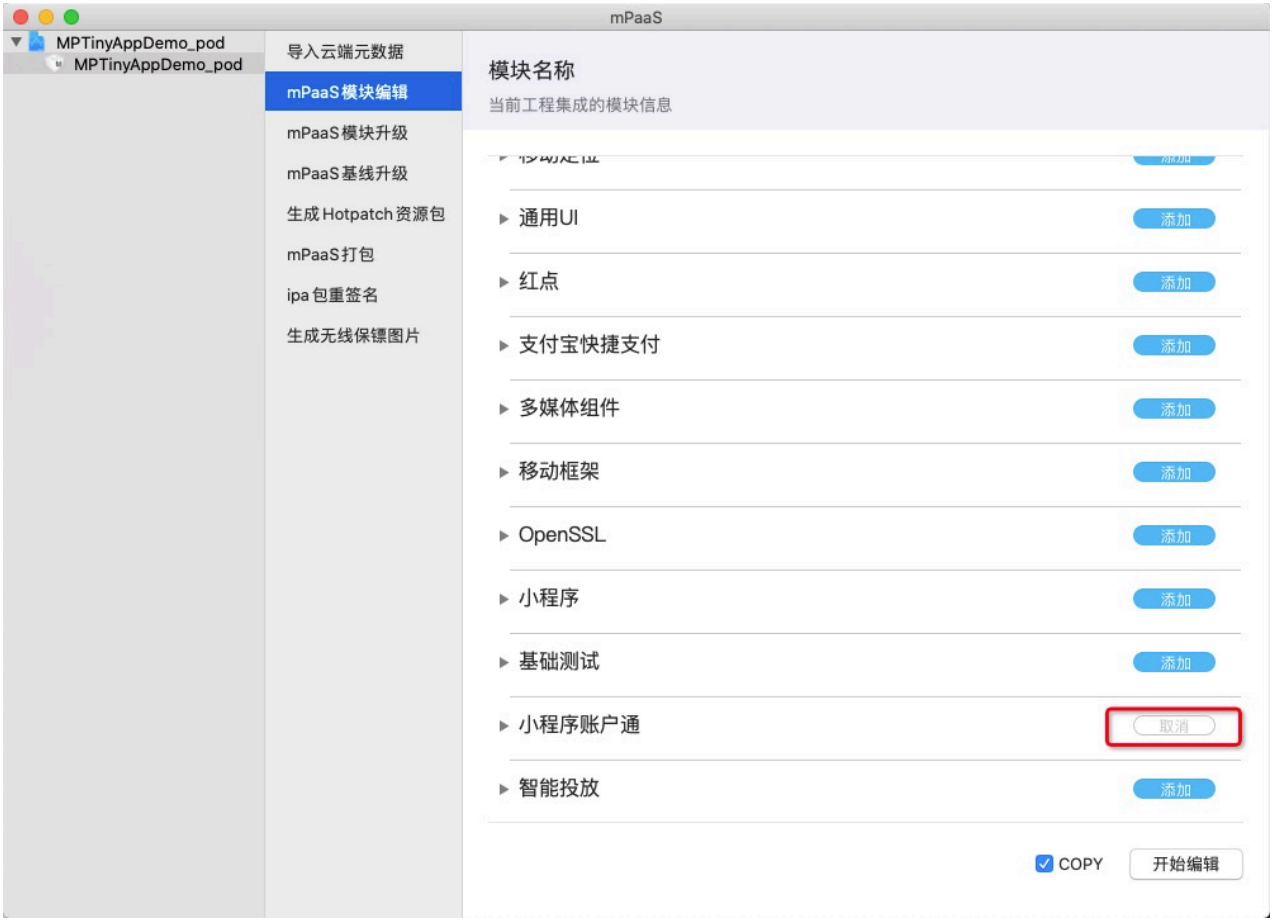
如果您已使用 mPaaS 插件集成过快捷支付 SDK，则需要在按如下步骤操作：

1. 点击右侧的 **取消** 快捷支付 SDK。
2. 点击 **开始编辑**。
3. 点击 **小程序账户通** 右侧的 **添加**，集成 SDK。

如果在添加小程序账户通前不取消快捷支付，则有可能出现符号冲突问题。

插件接入方式

使用 Xcode 插件添加账户通组件。



小程序自身会提供众多的 JSAPI 和 OpenAPI 能力，因此在插件中选择小程序组件后，相应的依赖组件也会默认添加到工程中。

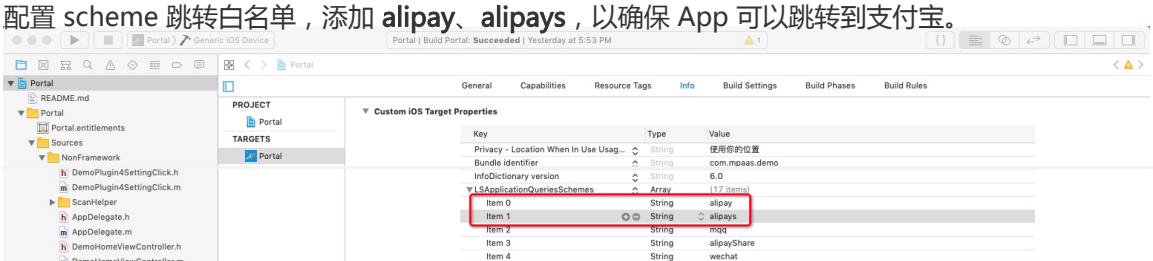
cocoapods 接入方式

1. 搭建 mPaaS cocoapods 环境，参见 基于原生框架且使用 CocoaPods 接入 。
2. 按照下方示例修改 podfile，引入账户通。

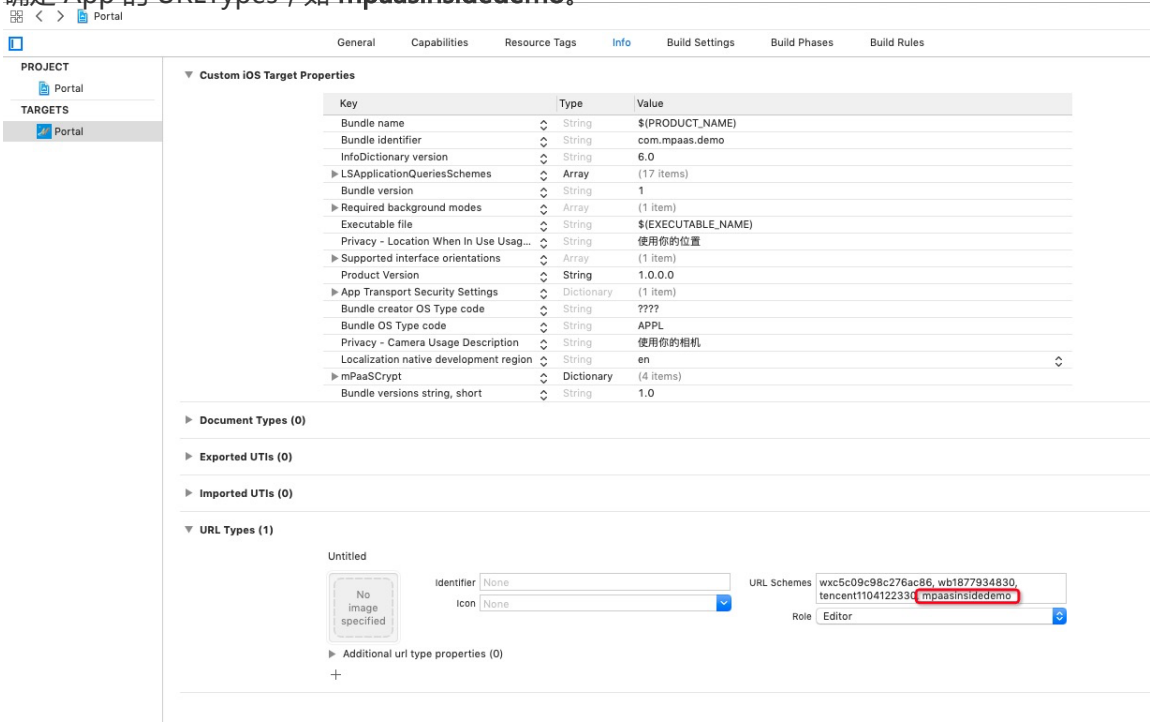
```
# mPaaS Pods Begin
plugin"cocoapods-mPaaS", :show_all_specs => true
source"https://code.aliyun.com/mpaas-public/podspecs_test.git"
#use_pod_for_mPaaS!
mPaaS_baseline '10.1.60' # 请将 x.x.x 替换成真实基线版本
# mPaaS Pods End
platform :ios, '9.0'
target 'MPTinyAppDemo_pod' do
// 小程序
mPaaS_pod"mPaaS_TinyApp"
// 账户通
mPaaS_pod"mPaaS_AliAccount"
end
```

配置 scheme 协议跳转

1. 配置 scheme 跳转白名单，添加 **alipay**、**alipays**，以确保 App 可以跳转到支付宝。

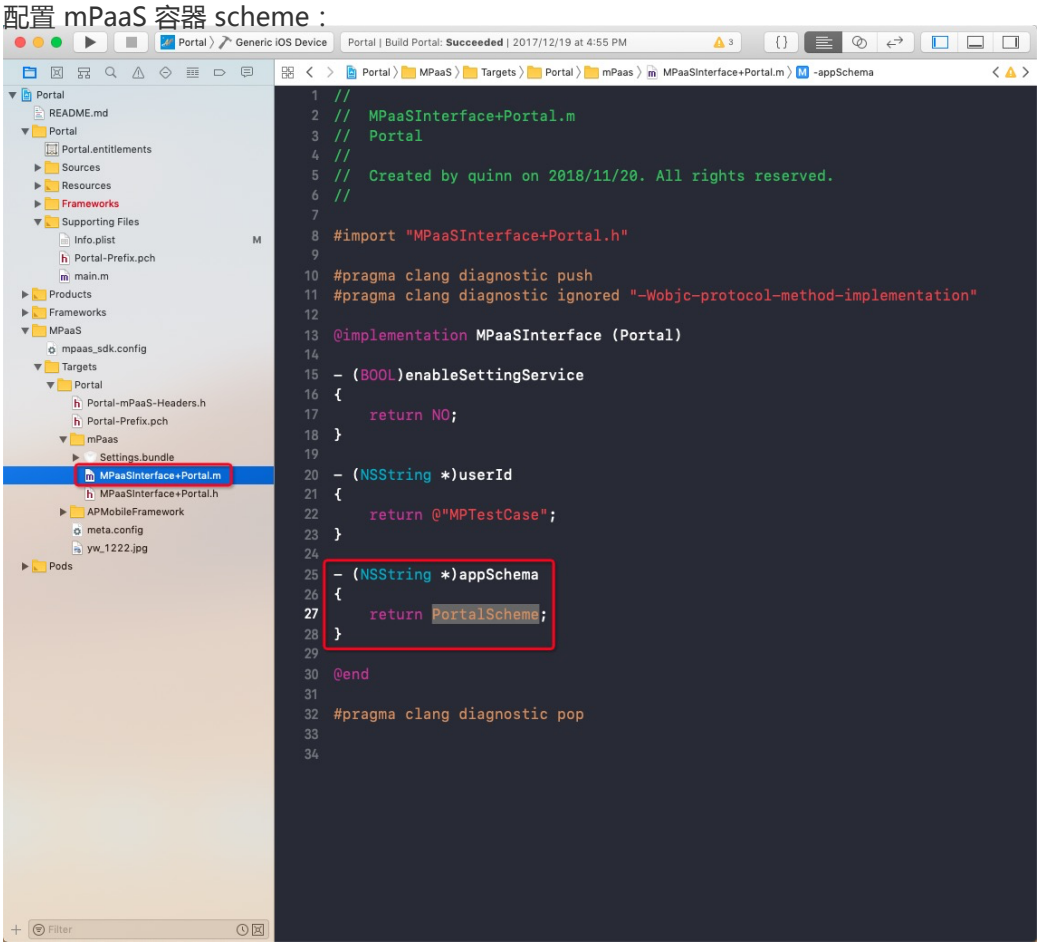


2. 确定 App 的 URLTypes，如 **mpaasinsidedemo**。



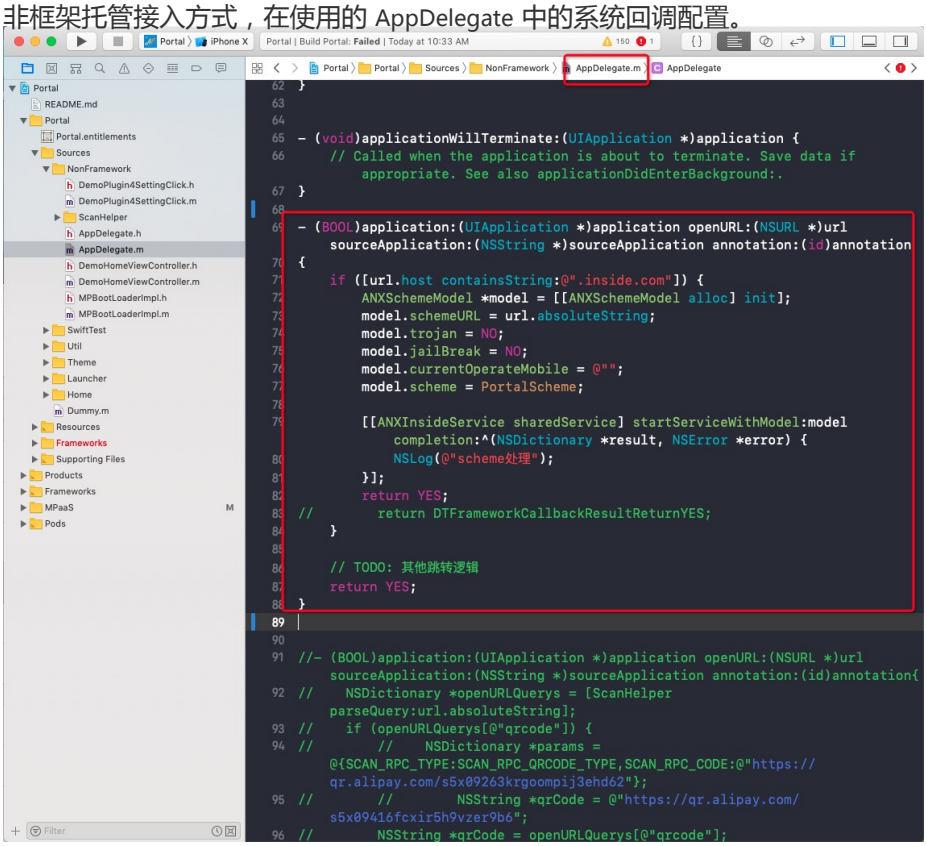
3. 配置 scheme 协议相关。

- 配置 mPaaS 容器 scheme：



说明：下文中涉及到的所有 scheme 值都需要和此处一致。

- scheme 系统回调配置：
- 非框架托管接入方式，在使用的 AppDelegate 中的系统回调配置。



- 框架托管接入方式，在 mPaaS 框架提供的回调中配置。

4. 账户通其他配置。
   - 提交工单 或联系售后，生成新的 ANX_ALIPAY_INSIDE_CONFIG，覆盖 InsideDataConfig.bundle 中的原有配置。
   - 提交工单 或联系售后，根据 mPaaS 无线保镖图片生成新的无线保镖图片并替换。

需要账户通的小程序打开方式。

当要打开的小程序需要账户通能力时，**不要** 使用原有小程序打开接口，而需要使用以下专门的账户通小程序打开接口：

```
#import <NBInsideAccountAdaptor/NBIAuthService.h>
#import <InsideAccountOpenAuth/ANXAccountOpenAuthModel.h>
#import <InsideService/ANXInsideService.h>

[NBIAuthService shareInstance].delegate = self;
[[NBIAuthService shareInstance] startTinyApp:item[0] uId:nil params:nil];
```

- 且需要实现 NBIAuthDelegate 这个 delegate：

```
- (void)authModelForMode:(NBIAuthMode)mode extendParams:(NSDictionary *)extendParams
callback:(NBIAuthCallback)callback {
// NeedRefreshToken == YES;
// 账户通来保障是串型的，如果一直是 NeedRefreshToken，那么就是要不断跳授权
// TODO：此处跳转支付宝获取授权然后获取 accessToken 等以及从本地持久化获取 accessToken 均为样例参考，实际情况接入方自定义
if (YES == [[extendParams objectForKey:@"NeedRefreshToken"] boolValue]) {
[self getOnlineTokenWithMode:mode callback:callback];
} else {
NBIAuthModel *model = [self getLocalTokenModelWithMode:mode];
if (nil == model) {
[self getOnlineTokenWithMode:mode callback:callback];
return;
}
if(mode == NBIAuthModePlatformOnly) {
callback(model);
}
}
}
```

获取数据 model

小程序需要支付宝登录态时，会回调到上文中的 delegate 函数，在这个函数中需要获取对应数据 model 并通过 callback 返回给小程序容器，容器会使用这个数据 model 获取支付宝登录态。

- 数据 model 有三个部分：
  - uid（支付宝用户 ID）
  - accessToken（支付宝授权 Token）
  - mcUid（App 用户 ID）
- 获取数据 model 的流程为：
  - 回调中通过 AuthURL 跳转支付宝获取授权。
  - 授权成功后获得 authCode。
  - 将 authCode 传给接入方的接口获取 model，调用 delegate 中的回调将数据传给小程序容器进行获取支付宝登录态操作。
- 通常 accessToken 存在一个有效期，为了避免每次打开小程序都跳转支付宝要求授权，常规做法为：首次获取支付宝授权并获取数据 model 后，将 model 缓存下来，当 model 没有过期时，将 model 返回给小程序容器即可。

判断 model 有无过期的标志是 delegate 回调携带的参数 extendParams 中的 NeedRefreshToken 标识，NeedRefreshToken 表示 accessToken 等过期，需要重新跳转支付宝授权，这时就可以重新走获取数据 model 的流程。
示例如下：

```
- (void)getOnlineTokenWithMode:(NBIAuthMode)mode callback:(NBIAuthCallback)callback
{
ANXAccountOpenAuthModel *model = [[ANXAccountOpenAuthModel alloc] init];
model.scheme = PortalScheme;
model.thirdAuth = YES;
// TODO: 接入方提供 AuthURL
model.authURL = @"xxx";
model.phoneNum = nil;

[[ANXInsideService sharedService] startServiceWithModel:model completion:^(NSDictionary<ANXCallbackKey *,id>
*result, NSError *error) {
if ([result[ANXProductConfigResultCodeKey] isEqualToString:@"account_open_auth_9000"]) {
//授权成功，可以拿到authcode、app_id
NSString *authcode = result[ANXProductConfigResultKey][@"auth_code"];

NSDictionary *userInfo = @{@"behaviorCode": @"AccountOpenAuth",
@"params1": result[ANXProductConfigResultKey]
};
[[NSNotificationCenter defaultCenter] postNotificationName:@"ANX_Login_log"object:nil userInfo:userInfo];

NBIAuthModel *model = [[NBIAuthModel alloc] init];
model.uid = @"xxx";
model.token = @"xxx";
model.extraInfo = @{@"mcUid": @"xxx"};
if(mode == NBIAuthModePlatformOnly) {
callback(model);
}
}
}];
}
```

说明：AuthURL 与根据 authCode 获取 uid、accessToken、mcUid 的接口都是由接入方提供。

接入支付

小程序需要支付宝快捷支付能力时，需要配置 scheme 系统回调。

- 对于非 mPaaS 框架托管模式，添加如下代码：

```
#import <AlipaySDK/AlipaySDK.h>
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString
*)sourceApplication annotation:(id)annotation
{
if ([url.host containsString:@"safepay"])
{
[[AlipaySDK defaultService] processOrderWithPaymentResult:url standbyCallback:nil];
}
// TODO: 其他跳转逻辑
return YES;
```

```
}
```

- 对于 mPaaS 框架托管模式，在框架类的 Category 中添加：

```
#import <AlipaySDK/AlipaySDK.h>
- (DTFrameworkCallbackResult)application:(UIApplication *)application openURL:(NSURL *)url newURL:(NSURL
**)newURL sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
if ([url.host containsString:@"safepay"])
{
[[AlipaySDK defaultService] processOrderWithPaymentResult:url standbyCallback:nil];
}
// TODO: 其他跳转逻辑
return YES;
}
```

说明：上方两端代码中的 AlipaySDK 调用的 callback 需要为 nil，这样小程序中调用 tradePay jsapi 时使用的 AlipaySDK 在支付完成时才能拿到回调。

客户端退出登录或切换用户

接入方退出登录或者切换用户时，需要清除客户端支付宝登录态，否则会造成小程序仍使用前用户的支付宝登录信息。接入方需在退出登录或者切换用户处调用下面的方法，并且注意在使用账户通功能时，清除 authcode、accesstoken 等的持久化或缓存。

```
// 当商户账号退出或切换账号时，都需要调用账号退出登录函数，告知账户通退出登录，然后再次进入账户通时重新授权和绑
定
ANXMCAccountStatusChangeModel *model = [ANXMCAccountStatusChangeModel new];
model.status = MCAccountLogout; // 账号退出登录
//model.status = MCAccountUnbind; // 账号解绑支付宝
[[ANXInsideService sharedService] startServiceWithModel:model completion:nil];
// TODO: 注意在使用账户通功能时，清除 authcode、accesstoken 等的持久化或缓存
```

支付配置

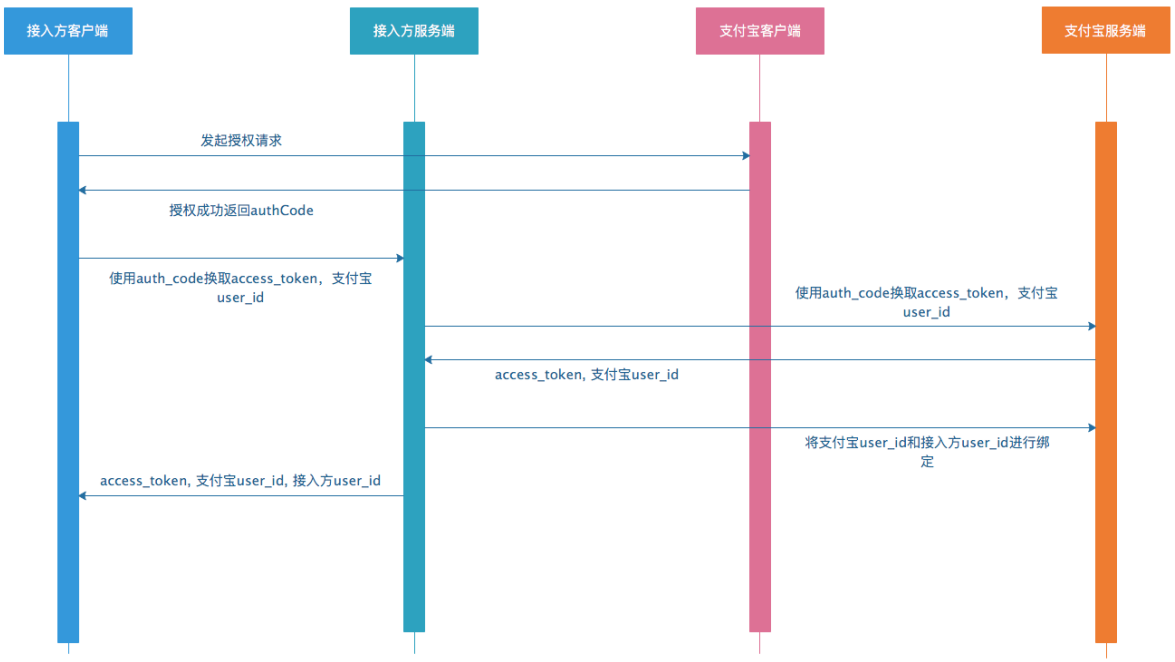
接入方对于支付配置有要求时，可通过 提交工单 获取活动标识和业务场景标识。获取标识后，在客户端 info.plist 中进行设置，设置参考如下：

```
<key>MPAuthIdentity</key>
<dict>
<key>instBizSceneCode</key>
<string>业务场景标识（替换为实际值）</string>
<key>instCampaignIds</key>
<string>活动标识（替换为实际值）</string>
</dict>
```

5.3 账户通接入服务端

账户通服务用于在非支付宝客户端上实现支付宝用户登录态获取并访问支付宝相关的服务，如授权、支付等。

除了需要在客户端接入相关 SDK 之外，接入方还需要在服务端实现相关服务以完成支付宝三方授权流程。支付宝三方授权的流程如下：



服务端接入流程包含以下几步：

1. 应用配置 。
2. 生成授权链接 。
3. 获取 。
4. 同步账号绑定关系 。

应用配置

接入开始前，需要在支付宝开放平台上创建、配置并上线应用。详情参考 创建应用。

创建说明：

- 创建应用时选择 **自定义接入**，应用类型选择 **移动应用**。
- 添加应用功能，选择 **获取会员信息**。
- 所填写的授权回调地址必须与后续接入中使用的回调地址一致。

生成授权请求链接

为帮助开发者调用支付宝开放平台接口，我们提供开放平台服务端 SDK，用于封装签名验签、HTTP 请求等基础功能。您可前往 下载页面 下载选择所需语言的版本。在下文中会通过 JAVA 版本的 SDK 做接入用于举例说明。

授权请求链接构造分为两个步骤：

1. 接口调用初始化
2. 构造业务请求参数

接口调用初始化

代码如下：

```
AlipayClient alipayClient = new DefaultAlipayClient(ALIPAY_GATEWAY_URL, APP_ID,
APP_PRIVATE_KEY, FORMAT, CHARSET, ALIPAY_PUBLIC_KEY, SIGN_TYPE);
```

参数说明：

配置参数	参数说明	参数值或获取方式
ALIPAY_GATEWAY_URL	支付宝网关，此为固定值。	固定值，填写内容为：https://openapi.alipay.com/gateway.do
APP_ID	创建应用时获得的 APPID。	参考 创建应用 > 查看 APPID
APP_PRIVATE_KEY	开发者私钥，由开发者自己生成。	参考 创建应用 > 配置应用环境
FORMAT	参数返回格式，只支持 json。	固定值，填写内容为：json
CHARSET	编码格式，支持 GBK/UTF-8。	开发者按需选择
ALIPAY_PUBLIC_KEY	支付宝公钥。	参考 生成公钥
SIGN_TYPE	开发者生成签名所使用的算法类型，支持 RSA 与 RSA2，推荐使用 RSA2。	RSA2

构造业务请求参数

参数名称	是否必须	参数说明
return_url	是	回调地址，必须与配置应用时设置的的回调地址一致。
scopes	是	接口权限值，如：auth_user、auth_base 等，请求格式为："scopes":["auth_user","auth_base"]。
state	是	• 自定义参数，用户授权后，重定向到 redirect_uri 时会原样回传给开发者。 • 为防止 CSRF 攻击，建议开发者请求授权时传入 state 参数，该参数要做到既不可预测，又可以证明客户端和当前第三方网站的登录认证状态存在关联。 • 只允许 Base64 字符（长度小于等于 100）。
auth_type	是	auth_type，用于标识授权类别。值为：MY_PASS_OAUTH
origin	是	调用来源，填写一个宿主标识，**由我方分配**。
is_mobile	是	true

代码示例

```
AlipayClient alipayClient = new DefaultAlipayClient("https://openapi.alipay.com/gateway.do", appId,
appPrivateKey,"json","UTF-8", alipayPublicKey,"RSA2");
```

```
// 创建 API 对应的 request
AlipayUserInfoAuthRequest alipayRequest = new AlipayUserInfoAuthRequest();

// 设置授权回调地址，在开放平台后台配置
alipayRequest.setReturnUrl("授权回调 url 地址");

// 构造 scope 列表
List<String> scopes = new ArrayList<String>();
scopes.add("auth_base");
scopes.add("auth_user");

Map<String, Object> bizContent = new HashMap<String, Object>();
bizContent.put("scopes", scopes);

bizContent.put("auth_type","MY_PASS_OAUTH"); // 固定值
bizContent.put("origin","XXXX"); // 调用来源，例如 AMAP、UC_BROSWER、NAPOS 等
bizContent.put("is_mobile","true"); // 固定值

// 请求唯一随机标识，用于防 CSRF 攻击，只允许 Base64 字符（长度小于等于 100）
bizContent.put("state","xxxxxx");

// 填充业务参数
alipayRequest.setBizContent(JSONObject.toJSON(bizContent).toString());

AlipayUserInfoAuthResponse response = alipayClient.pageExecute(alipayRequest,"GET");
if (response.isSuccess()) {
System.out.println("调用成功");
System.out.println(response.getBody());
} else {
System.out.println("调用失败");
System.out.println(response.getSubCode() +":"+ response.getSubMsg());
}
```

生成的授权链接样例如下：

https://openapi.alipay.com/gateway.do?alipay_sdk=alipay-sdk-java-3.7.4.ALL&app_id=2019040163782051&biz_content=%7B%22auth_type%22%3A%22MY_PASS_OAUTH%22%2C%22scopes%22%3A%5B%22auth_user%22%5D%2C%22state%22%3A%2210%22%2C%22is_mobile%22%3A%22true%22%7D&charset=UTF-8&format=json&method=alipay.user.info.auth&return_url=http%3A%2F%2Fzhanghutong.yuguozhou.online%2Ffirst&sign=RHLcR%2BbfgW50JgNr5e6MTT08Bnnb3%2Fyt%2B0YIObm%2Fdpq2yJtYzHKgmS2ciVrgFEk6DUKtEmipoLb8xJ8ErFQAtSS7p8AvXGGY63D95N4lm6yasUVCg2kGoofeB9OPk7GBkLkud1CY3oCbK4HgbHHnHIc43GtXuKt0QLMPivZjKgqb5u1zt%2FKscdCt8JrLG4L5vOOFGKRuh3cFq%2BVL%2Bdvaufwbut6B%2B85GjOsnvONICif8r9cxpdzlsRFoSVmYu%2F7AUM34diatlQPvKs5NOeeAg2W8QkBbQYza0f84KYrNAAeX9ITbzvc7ntiL9606qEB1OWj%2Flccm%2B1TSKQjUUjjC6A%3D%3D&sign_type=RSA2&timestamp=2019-04-28+17%3A28%3A04&version=1.0

此授权链接可重复使用，客户端使用此授权链接向支付宝发起授权请求获取 auth_code。

获取 access_token 和支付宝 user_id

通过 alipay.system.oauth.token 接口获取 access_token 和支付宝 user_id。

接口调用示例：

```
AlipayClient alipayClient = new DefaultAlipayClient("https://openapi.alipay.com/gateway.do", APP_ID,
APP_PRIVATE_KEY,"json", CHARSET, ALIPAY_PUBLIC_KEY,"RSA2");

AlipaySystemOauthTokenRequest request = new AlipaySystemOauthTokenRequest();
request.setCode("2e4248c2f50b4653bf18ecee3466UC18");
request.setGrantType("authorization_code");
try {
AlipaySystemOauthTokenResponse oauthTokenResponse = alipayClient.execute(request);
System.out.println(oauthTokenResponse.getAccessToken());
} catch (AlipayApiException e) {
e.printStackTrace();
}
```

同步账户绑定关系

获取 access_token 和支付宝 user_id 后，需要进一步调用账户绑定关系同步接口，把机构的用户唯一标识与支付宝 user_id 进行绑定。

接口定义

重要：

- 绑定关系未落成功会影响登录态的创建，从而导致用户无法访问业务。
- 由于网络超时或其他原因，如果接口返回超时或者未知异常，接入方务必保障重试直到成功为止。

接口名称	alipay.user.antpaas.role.relation.save
接口描述	保存账号绑定关系，涵盖增、删、改操作
入参说明	• userId：接入方站点用户 UserId。 • userSource：接入方站点名，**由我方分配**。 • alipayUserId：需要绑定的支付宝用户 ID。 • userOccupiedAutoDelete：填 true 或 false，若用户已被其它支付宝用户绑定，则自动删除已有关系，仅在 opType=enable 时有效。 • alipayUserOccupiedAutoDelete：填 true 或 false，若支付宝用户已被其它接入方站点用户绑定，则自动删除已有关系，仅在 opType=enable 时有效。 • opType：操作类型，可选 enable 或 delete，表示 **存储** 或 **删除**。
出参说明	• code：结果码 • msg：结果信息
错误码	INVALID_PARAMETER SYSTEM_ERROR USER_OCCUPIED ALIPAY_USER_OCCUPIED

调用示例

```
AlipayClient alipayClient = new DefaultAlipayClient("https://openapi.alipay.com/gateway.do","app_id","your
private_key","json","GBK","alipay_public_key","RSA2");
AlipayUserAntpaasRoleRelationSaveRequest request = new AlipayUserAntpaasRoleRelationSaveRequest();
request.setBizContent("{"+
"\"user_id\":\"287346876344\","+
"\"user_source\":\"FINTECH_TEST\","+"\"alipay_user_id\":\"2088131231323456\","+
"\"op_type\":\"enable\","+
"\"user_occupied_auto_delete\":true,"+"\"alipay_user_occupied_auto_delete\":true"+
"}");
AlipayUserAntpaasRoleRelationSaveResponse response = alipayClient.execute(request);
if (response.isSuccess()) {
System.out.println("调用成功");
} else {
System.out.println("调用失败");
}
```

6 小程序基础库说明

基础库与客户端的关系

小程序能力需要客户端来支撑：

- 每一版基础库新增能力都需要运行在特定版本及以上的客户端中。
- 高版本基础库的某些新能力无法兼容低版本客户端。

关于基础库兼容方法，参见 兼容基础库 。您可以通过 my.SDKVersion 查看当前基础库版本号。

兼容基础库

现阶段，小程序组件和 API 能力正在逐步完善和丰富，但是老版本客户端并不支持这些新增的能力，因此建议开发者做对应的兼容性处理。

您可通过接口 my.canIUse(String) 实现兼容性判断，详见 接口说明 。

兼容示例

新增 API 兼容性处理

对于新增 API，可以参照下面的代码来判断当前基础库是否支持该 API：

```
if (my.getLocation) {
my.getLocation();
} else {
// 如果希望用户在最新版本的客户端上体验您的小程序，可以这样提示
my.alert({
title: '提示',
content: '当前版本过低，无法使用此功能，请升级最新版本'
});
}
```

API 新增参数兼容性处理

```
if (my.canIUse('getLocation.object.type')) {
// ...
} else {
console.log('当前版本不支持该参数')
}
```

API 新增返回值兼容性处理

```
if (my.canIUse('getSystemInfo.return.storage')) {
// ...
} else {
console.log('当前版本不支持该返回值')
}
```

组件新增属性兼容性处理

组件新增属性在旧版本客户端上无法实现，但也不会报错。若要对属性做降级处理可参见以下代码：

```
Page({
data: {
canIUse: my.canIUse('button.open-type.share')
}
})
```

基础库版本

基础库版本	基础库对应基线版本	
1.9.0	10.1.32	下载地址
1.14.1	10.1.60	下载地址

基础库集成说明

iOS

iOS 中无需进行基础库集成。

Android

实现 H5AppCenterPresetProvider 接口类。代码示例如下：

```
package com.mpaas.demo.nebula;

import com.alipay.mobile.nebula.appcenter.H5PresetInfo;
import com.alipay.mobile.nebula.appcenter.H5PresetPkg;
import com.alipay.mobile.nebula.provider.H5AppCenterPresetProvider;

import java.io.File;
```

```
import java.io.InputStream;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class H5AppCenterPresetProviderImpl implements H5AppCenterPresetProvider {
private static final String TAG ="H5AppCenterPresetProviderImpl";

// 设置小程序专用资源包，此 ID 固定，不可设置其他值
private static final String TINY_COMMON_APP ="66666692";

// 预置包的 asset 目录
private final static String NEBULA_APPS_PRE_INSTALL ="nebulaPreset"+ File.separator;

// 预置包集合
private static final Map<String, H5PresetInfo> NEBULA_LOCAL_PACKAGE_APP_IDS = new HashMap();

static {

H5PresetInfo h5PresetInfo2 = new H5PresetInfo();
// 内置目录的文件名称
h5PresetInfo2.appId = TINY_COMMON_APP;
h5PresetInfo2.version ="1.0.0.0";
h5PresetInfo2.downloadUrl ="";

NEBULA_LOCAL_PACKAGE_APP_IDS.put(TINY_COMMON_APP, h5PresetInfo2);

}

@Override
public Set<String> getCommonResourceAppList() {
Set<String> appIdList = new HashSet<String>();
appIdList.add(getTinyCommonApp());
return appIdList;
}

@Override
public H5PresetPkg getH5PresetPkg() {
H5PresetPkg h5PresetPkg = new H5PresetPkg();
h5PresetPkg.setPreSetInfo(NEBULA_LOCAL_PACKAGE_APP_IDS);
h5PresetPkg.setPresetPath(NEBULA_APPS_PRE_INSTALL);
return h5PresetPkg;
}

/**
* 设置可以降级的资源包 ID
*/
@Override
public Set<String> getEnableDegradeApp() {
return null;
}

@Override
public String getTinyCommonApp() {
return TINY_COMMON_APP;
```

```
}

@Override
public InputStream getPresetAppInfo() {
return null;
}

@Override
public InputStream getPresetAppInfoObject() {
return null;
}
}
```

根据客户端集成的版本下载相应的小程序基础库，将基础库复制到上一步中指定的 asset 目录下并重新命名。基于上一步的代码示例，小程序基础库路径为assets/nebulaPreset/66666692。

在启动时设置该 Provider 实例。代码示例如下：

```
H5Utils.setProvider(H5AppCenterPresetProvider.class.getName(), new H5AppCenterPresetProviderImpl());
```

说明：

- 如果客户端中有使用 H5 公共资源包，请将公共资源包相关代码合并至该 H5AppCenterPresetProvider 的实例类中。
- 如有更多问题，参见 代码示例 。

7 开发工具

7.1 概览

开发一个小程序通常包括以下步骤：

1. 创建
2. 编码
3. 调试
4. 测试
5. 上传
6. 发布

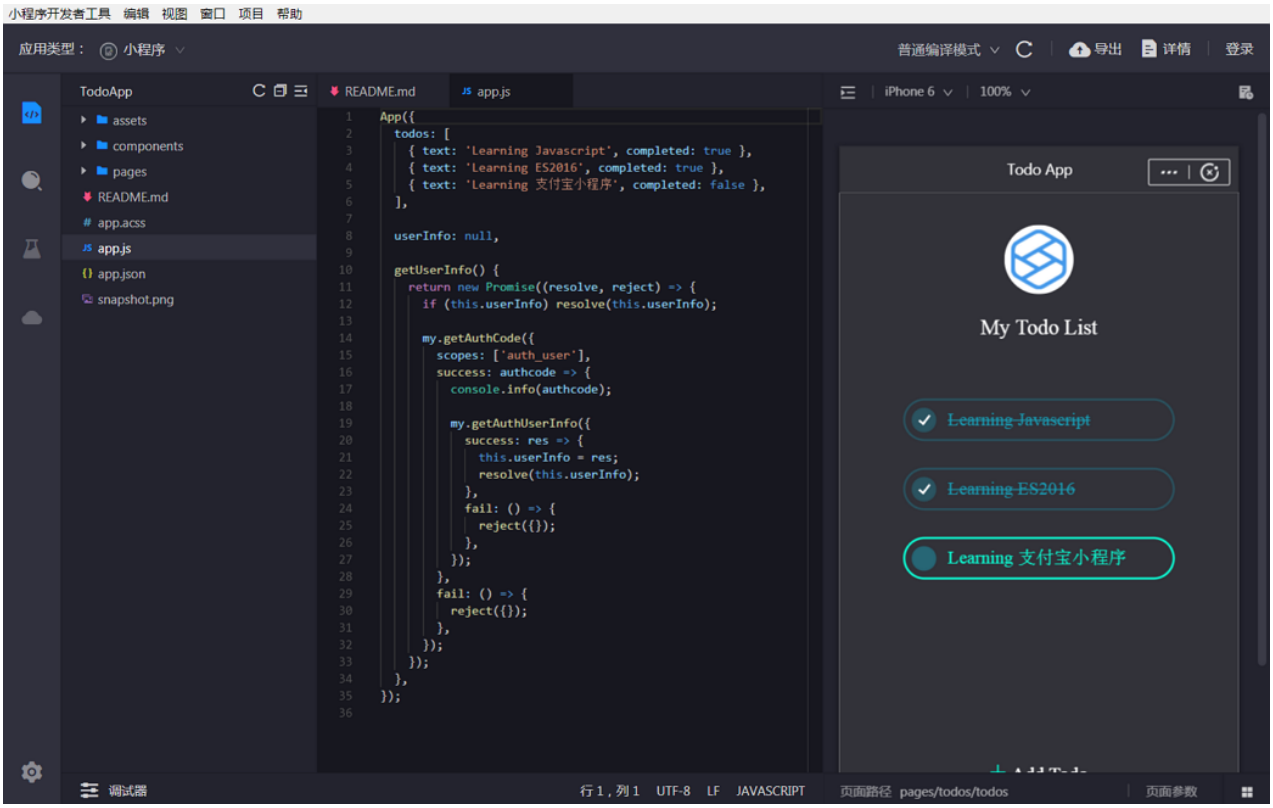
而小程序开发工具旨在为您提供一站式小程序研发服务，包括：

- 项目创建
- 编码
- 模拟器
- 调试工具
- 发布

您可前往小程序 下载中心，下载所需版本的小程序开发工具（IDE）。

界面

您可以选择创建一个示例工程，打开后会看到如下界面：



编码

小程序开发工具为您提供了小程序定制体验：

1. 文件树上支持一键创建小程序页面，包含 axml、acss、js、json 四个文件。
2. 小程序语法和 API 自动补全提示。

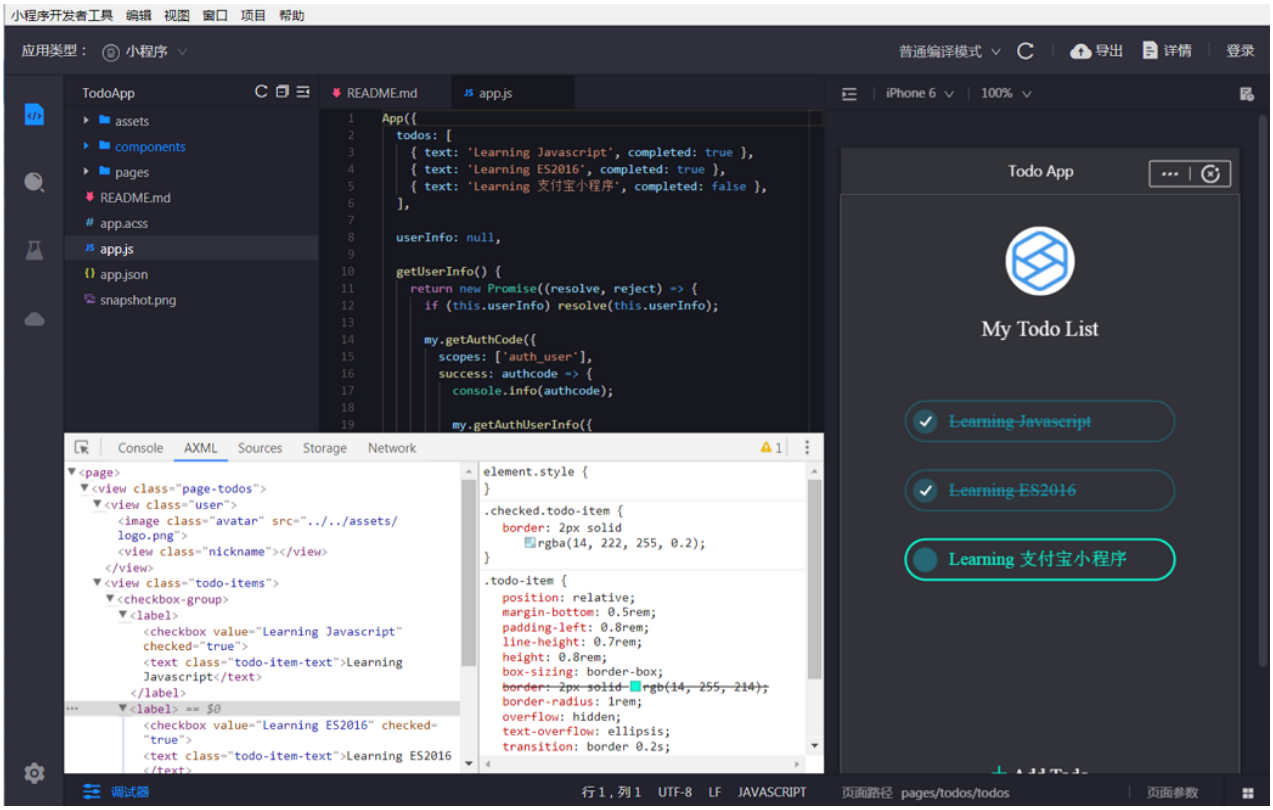
模拟器

模拟器是小程序开发工具的核心，小程序并不是一个普通的 H5 App，您无法在浏览器中预览。

任何代码的修改都会实时反应在模拟器上，并且为了便于调试，模拟器提供了诸如地理位置模拟等功能。

调试工具

小程序开发者工具也提供了定制的 Devtool，用于查看模拟器运行的日志等。调试工具也提供了 AXML 插件用于调试小程序的元素：



发布

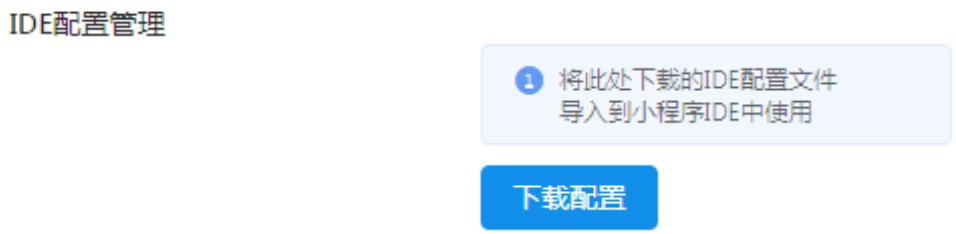
当您完成开发后，可以切换至下载菜单，把小程序包下载到本地，然后通过 mPaaS 发布平台发布小程序包。

7.2 创建/选择环境

小程序 IDE 中支持创建/选择环境，在未创建环境时，默认显示 mPaaS 公有云环境。

下载小程序 IDE 配置文件

您可前往 mPaaS 控制台 > **实时发布** > **小程序包管理** > **配置管理**，进入下载配置文件页面，在 **IDE 配置管理** 中点击 **下载配置**，下载小程序 IDE 配置文件。



说明：该 **IDE 配置文件** 不同于 mPaaS 应用配置文件。

创建环境

1. 点击小程序 IDE 右上方的环境选择下拉菜单。
2. 点击菜单底部的 **+ 新增研发配置**。

在弹出的 **新增环境** 窗口中，输入 **环境名称** 并上传 **小程序 IDE 配置文件**。

点击 **新增**，即可创建新的环境。

选择环境

您可以在小程序 IDE 中切换至已创建的环境，点击小程序 IDE 右上方的环境选择下拉菜单，可看到已有环境，点击环境名即可切换。

编辑、删除环境

点击已有环境右侧的 **编辑**（  ）或 **删除**（  ）按钮，即可对该环境的信息进行编辑或删除。

7.3 登录

小程序 IDE 的登录方式如下：

1. 下载小程序 IDE 配置文件：前往 mPaaS 控制台 > **实时发布** > **小程序包管理** > **配置管理**，进入下载配置文件页面，在 **IDE 配置管理** 中点击 **下载配置**。

   说明：该 **小程序配置文件** 不同于 mPaaS 应用配置文件。

2. 点击下载配置后，会弹出 **添加高级规则** 窗口，您需要在 **动态密码** 中输入一个密码，该密码就是之后登录 IDE 时所使用的登录密码。

3. 在 IDE 中选择 mPaaS 小程序后，点击界面右上方的 **登录**。

4. 在弹出的 **新增登录环境** 窗口中输入环境名，并上传在第 1 步中下载的 **小程序配置文件**，点击 **确定**。

5. 在随后打开的登录窗口中，输入账号密码登录。
    - 账号是 mPaaS 管理后台的用户名
    - 密码是下载配置文件时，在第 2 步输入的 **动态密码**。

6. 登录后，点击界面右上方的 **选择小程序** 来选择需要编辑的小程序项目。
7. 在弹出的 **选择小程序** 窗口中，依次选择 **租户**、**工作空间**、**App**、**小程序** 后，点击 **下一步**。

8. 在最后的 **配置小程序** 窗口中继续点击 **下一步**，就可以开始编辑小程序。

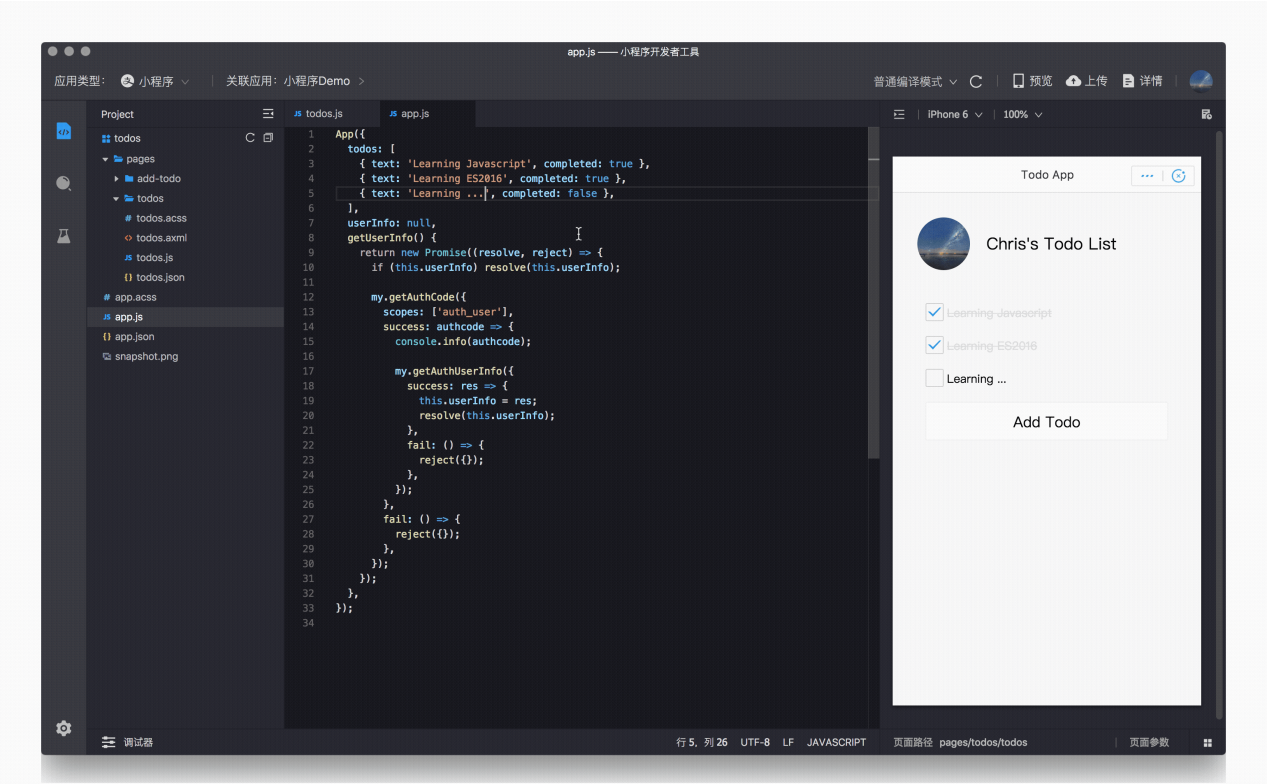
7.4 编码

在基本编辑功能之上，小程序开发者工具还提供了针对小程序定制的功能：

- 实时预览
- 自动补全
- 语法提示

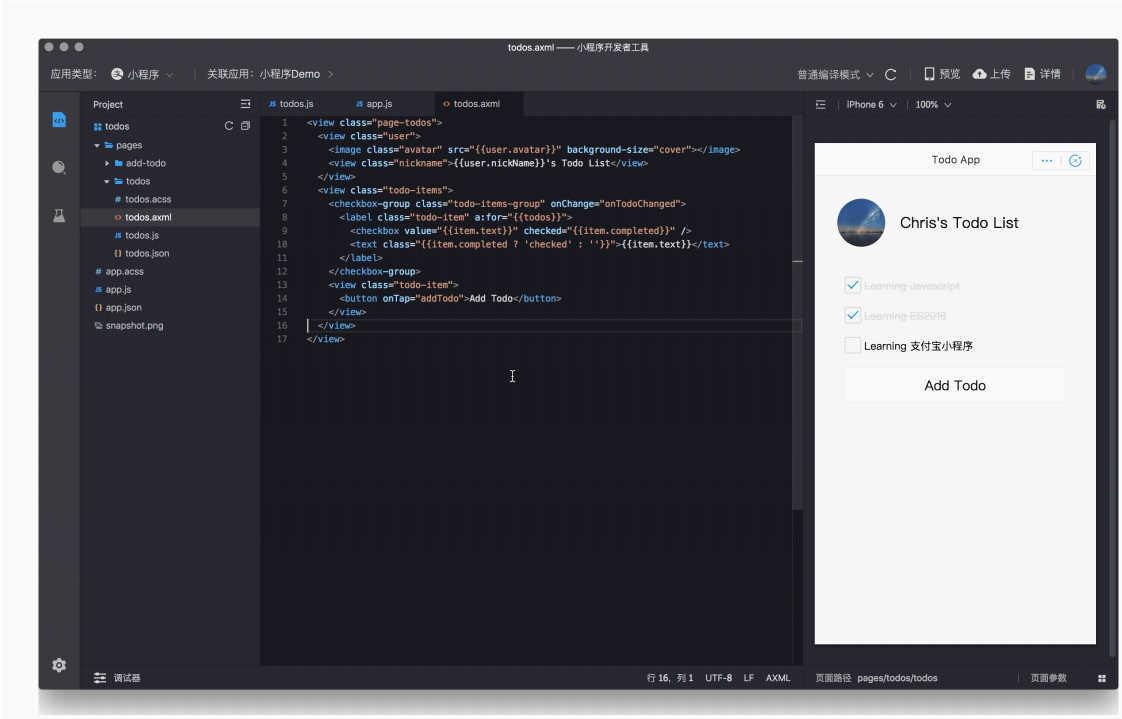
注意：以下功能演示仅供参考，实际界面以您打开的 IDE 界面为准。

实时预览

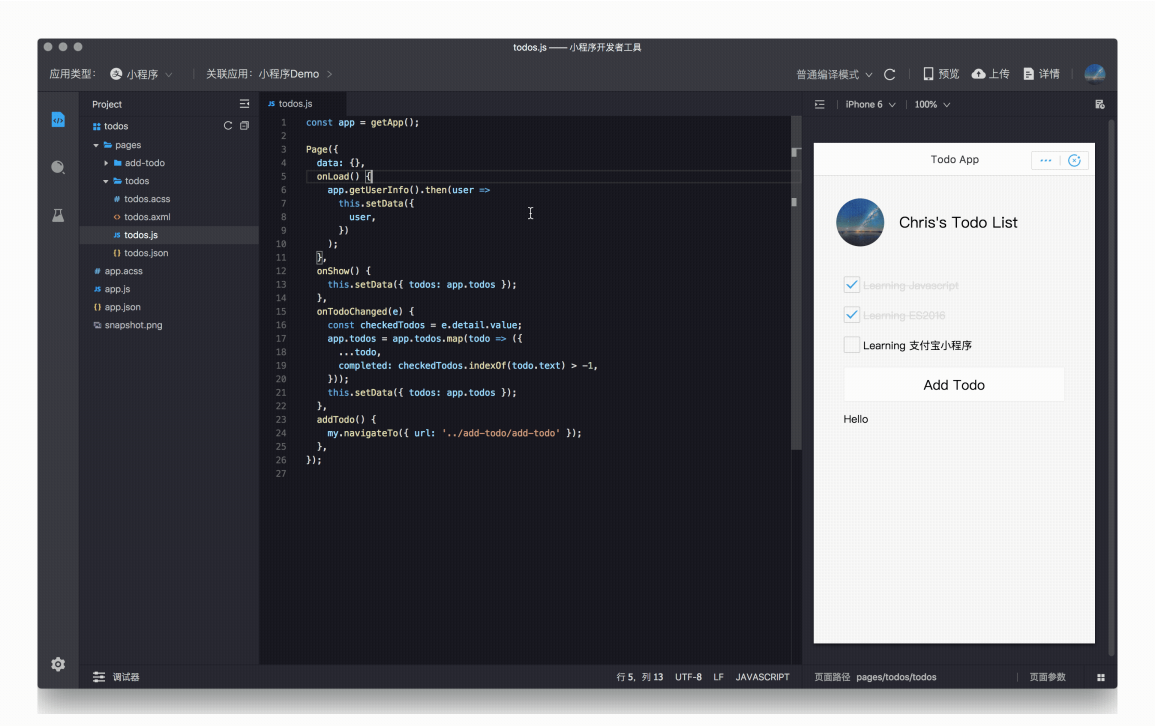


自动补全

AXML 自动补全：



API 自动补全：

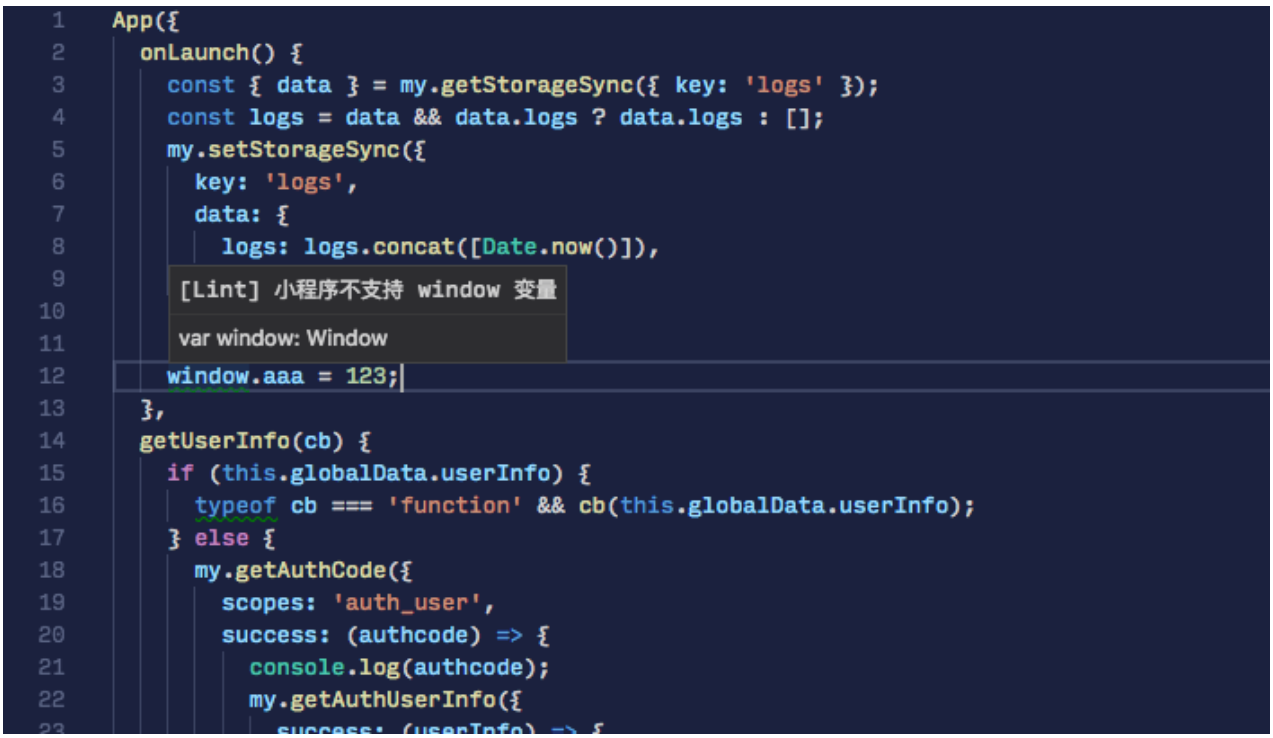


语法提示

小程序开发者工具默认支持 ES5/ES6/ES7，推荐使用 ES6 以上语法。同时，还提供了内置的 ESLint 支持，开发人员可以根据项目需求修改配置。ESLint 作为一个语法风格检查工具，除了可以让您的代码更加一致、可读，还会提供一些最佳实践的提示，帮助规避一些常见的错误。欲了解 ESLint 详情，参见 ESLint 。

小程序特有的语法提示：

在默认的 ESLint 规则之上，还增加了小程序特有的语法提示，比如下图中提示的：小程序不支持 window 变量。因为在小程序中 window 是一个保留字段，操作 window 可能会导致未知错误，所以不建议使用 window。



7.5 模拟器与调试器

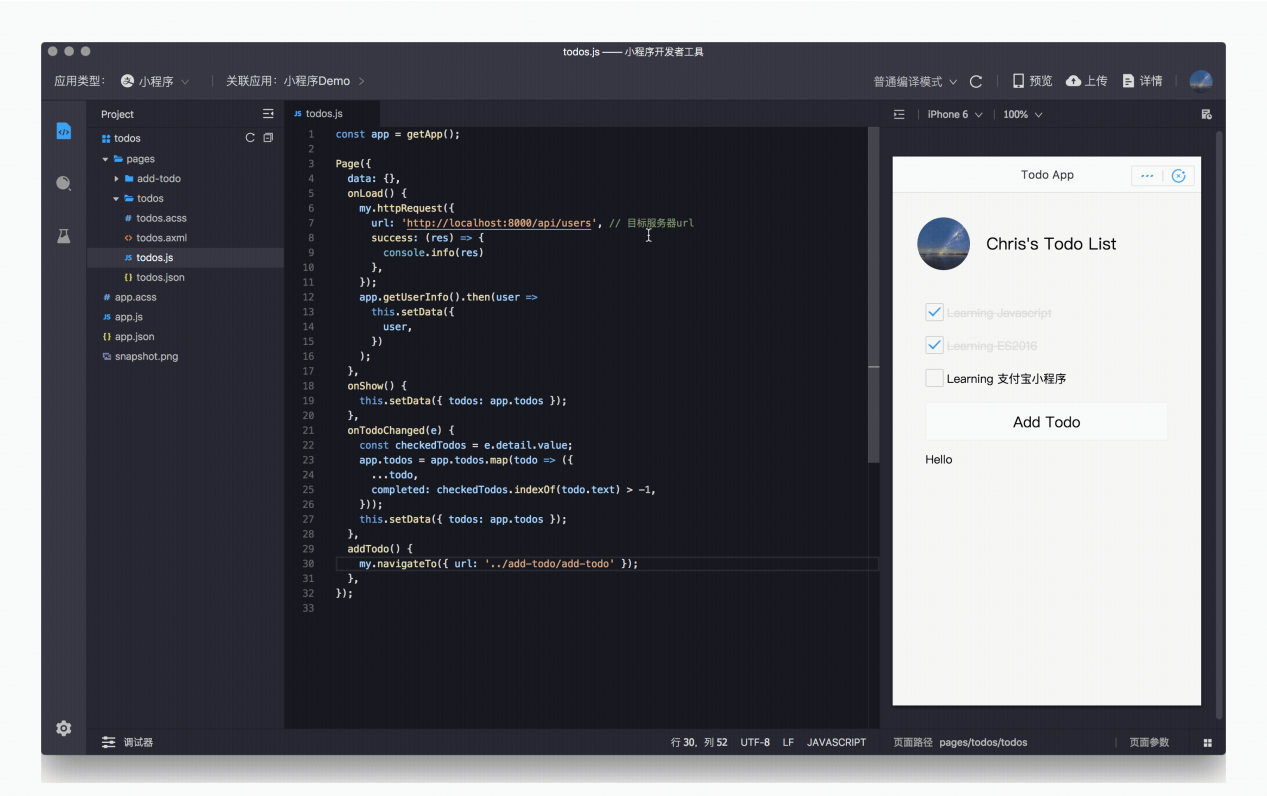
使用模拟器和调试工具进行调试。模拟器中模拟了大部分的真机 API，并且配有调试工具，您可以在模拟器中完成基础功能、样式的调试。

说明：以下功能演示仅供参考，实际界面以您打开的 IDE 界面为准。

模拟器

模拟器提供了如下功能：

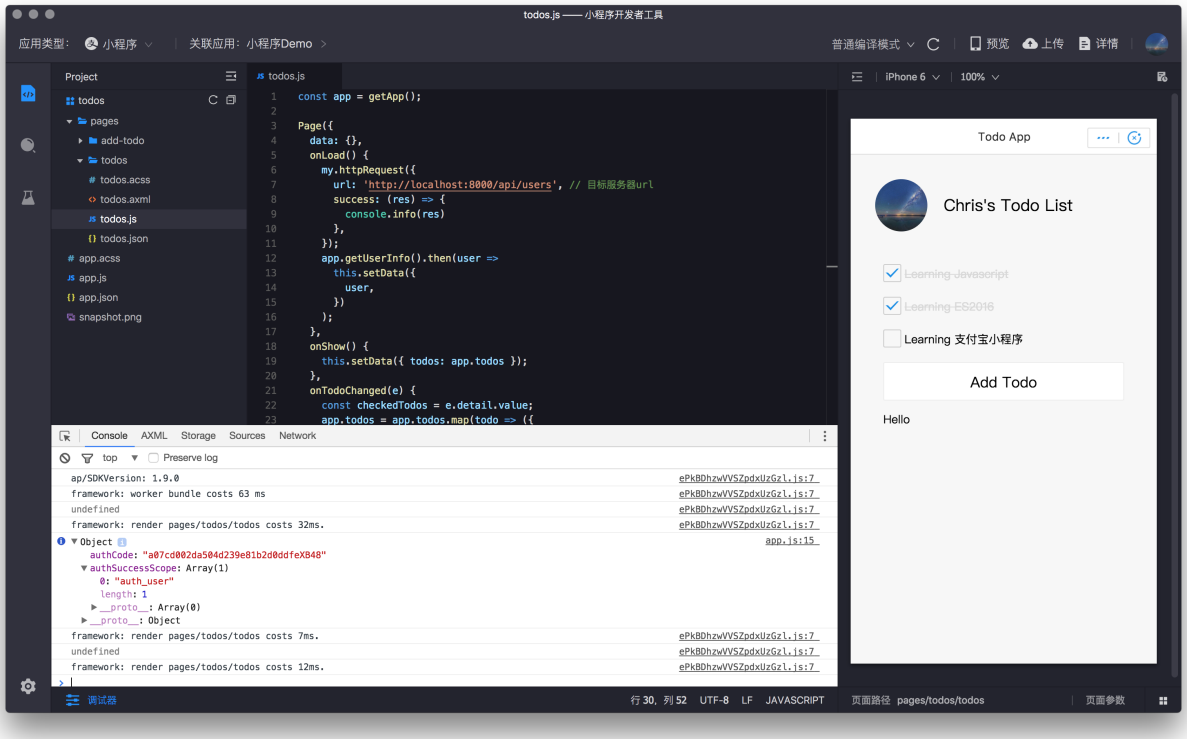
- 设备模拟，例如尺寸、精度等。
- 编译日志、编译错误提示、刷新。
- mPaaS JsAPI 模拟，以及位置、蓝牙、启动参数等模拟接口自定义配置。



调试器

配合模拟器，提供了定制化的 Chrome Devtool，在其基础上提供比如 axml 等扩展。默认展示的有：

- **AXML**：基于小程序元素的 dom、css 调试。
- **Console**：查看运行日志、错误。
- **Storage**：查看、编辑缓存数据。
- **Sources**：查看源码、调试断点。
- **Network**：查看网络资源、请求。



7.6 真机预览与调试

小程序 IDE 支持真机预览与调试，您可在手机客户端上预览当前代码的实际效果或进行调试。

说明：如果您是专有云客户，请确保已部署升级相关服务，详情请咨询支持人员。

操作步骤

1. 点击 IDE 右上方的 **预览** 或 **调试**。
   - IDE 会将当前代码生成 .zip 包并上传至 MDS。
   - MDS 自动创建发布任务，生成并返回二维码至 IDE。
2. 使用手机客户端扫描 IDE 中显示的二维码。

- 扫码之后，会触发 MDS 下发小程序包。
- 二维码有效期为 5 分钟，超时后会显示刷新按钮。
- 在 **当前用户** 中需输入当前扫码手机客户端的登录用户名，并点击 **修改**，否则手机无法接收到小程序包。

3. 待手机客户端收到小程序包后，即可在手机端进入预览或调试界面。

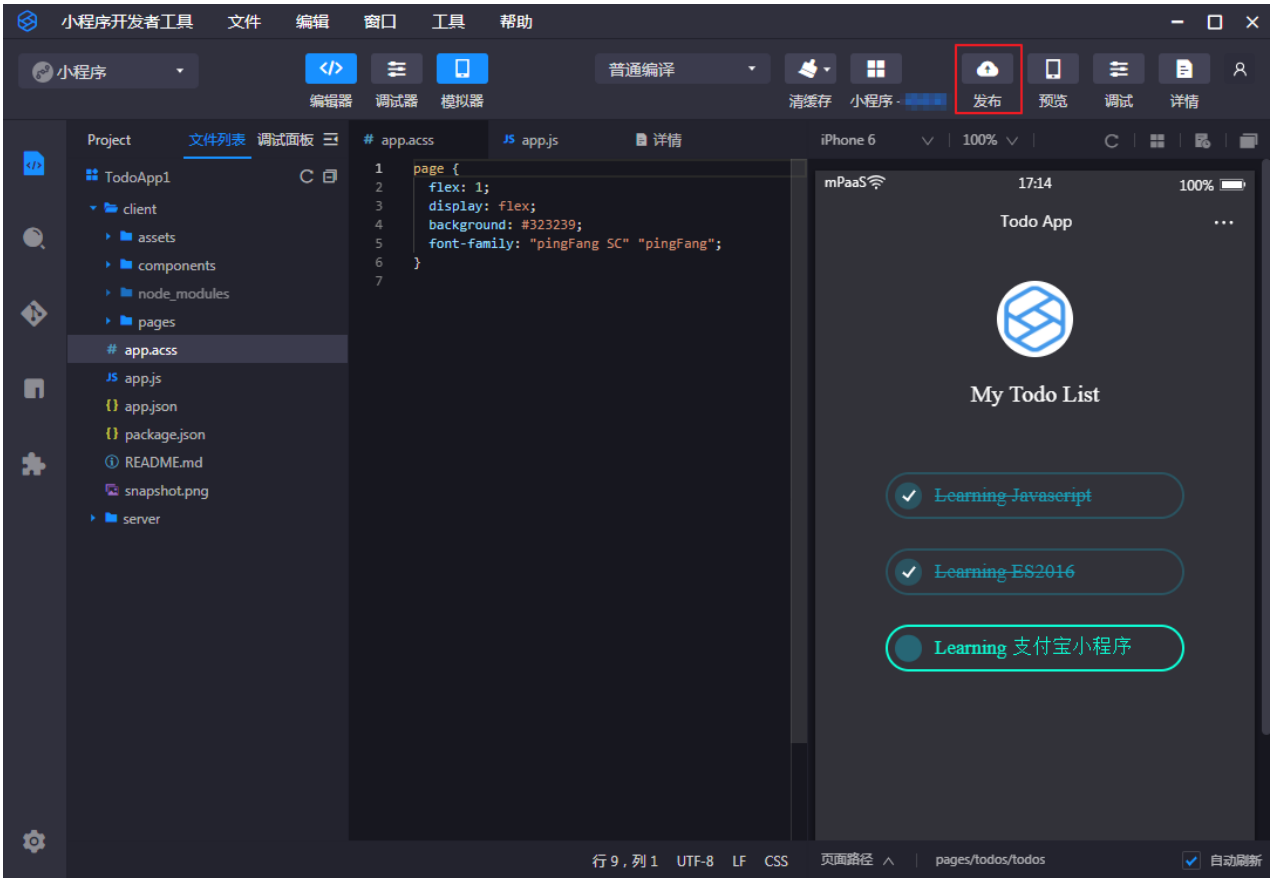
接入真机预览与调试

参见以下文档以获取 Android 与 iOS 小程序接入真机预览与调试的方法：

- Android
- iOS

7.7 上传

在完成本地编码、测试之后，您即可将代码上传至控制台。在小程序开发者工具中，提供了一键上传功能：



前置条件

- 您已 在控制台创建应用 。
- 您已 在控制台创建小程序 App 。

上传步骤

1. 点击 IDE 右上方的 **发布**。
2. 如果当前尚未创建环境或未登录，会弹出提示窗口，详情参见 创建环境 以及 登录 。
3. 登录后，选择对应的环境、应用以及小程序后，即可将小程序代码的 .zip 包上传至 MDS 服务端。
4. 上传完成后，您可前往 mPaaS 控制台 对已上传的小程序包进行发布。关于发布，参见 发布小程序 。

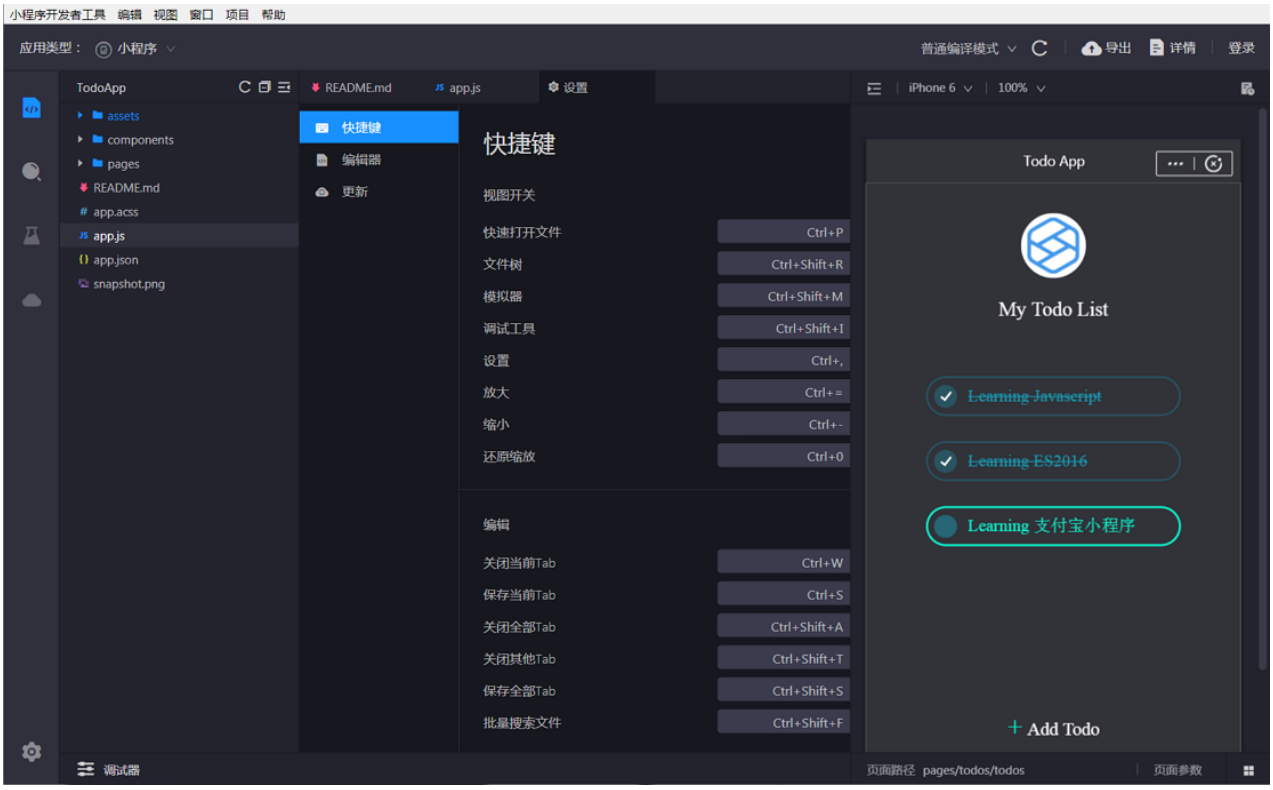
说明：

- 为了便于管理，每次上传都会递增版本号；您也可以自定义版本号。
- 上传时会进行体积检测等，超过小程序限制体积会给出提醒反馈。

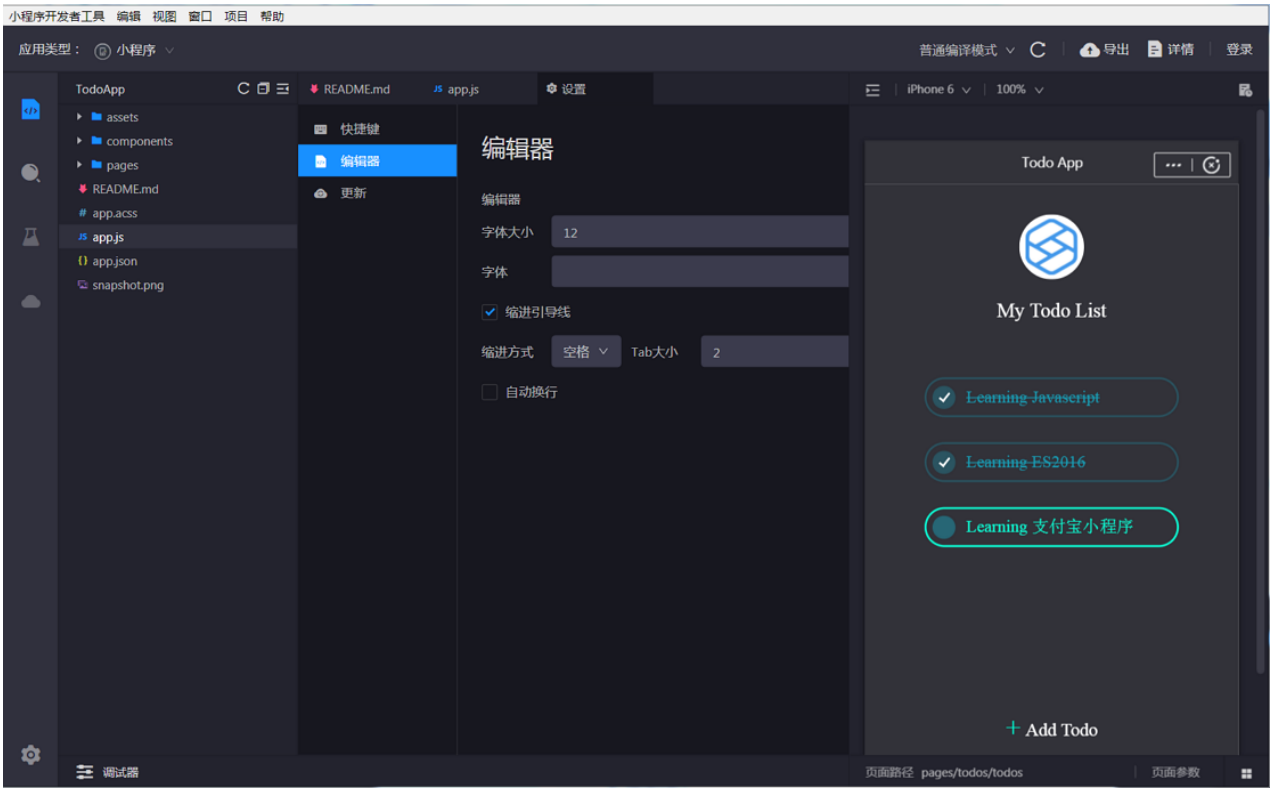
7.8 设置

在小程序开发者工具窗口左下角，点击设置图标（  ）进入设置界面。您可以在该界面设置快捷键和编辑器。

快捷键



编辑器



7.9 ESLint 简介

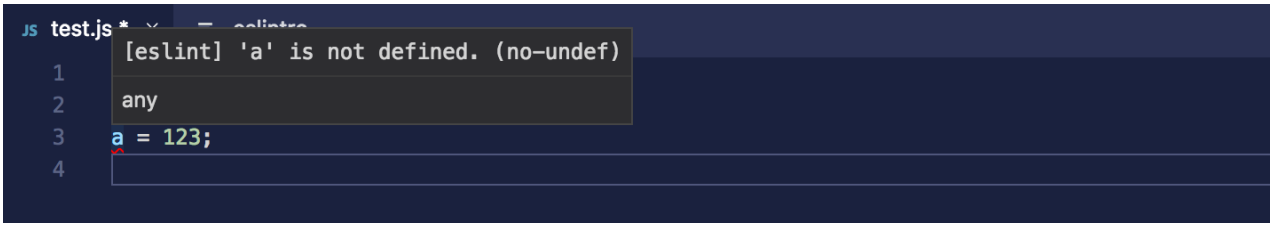
JavaScript 是一门非常灵活的语言，往往实现一个功能会有很多种写法。然而这种灵活性也给我们带来了很多问题。ESLint 作为一个语法风格检查工具，除了可以让您的代码更加一致、可读，还会提供一些最佳实践的提示，帮助规避一些常见的错误。

ESLint 介绍

在 IDE 升级后，如果发现出现很多红色的波浪线提示，那么这些提示其实都来自于 **ESLint**。例如，在 JavaScript 中给未声明的变量赋值时，其实是声明了一个全局变量：

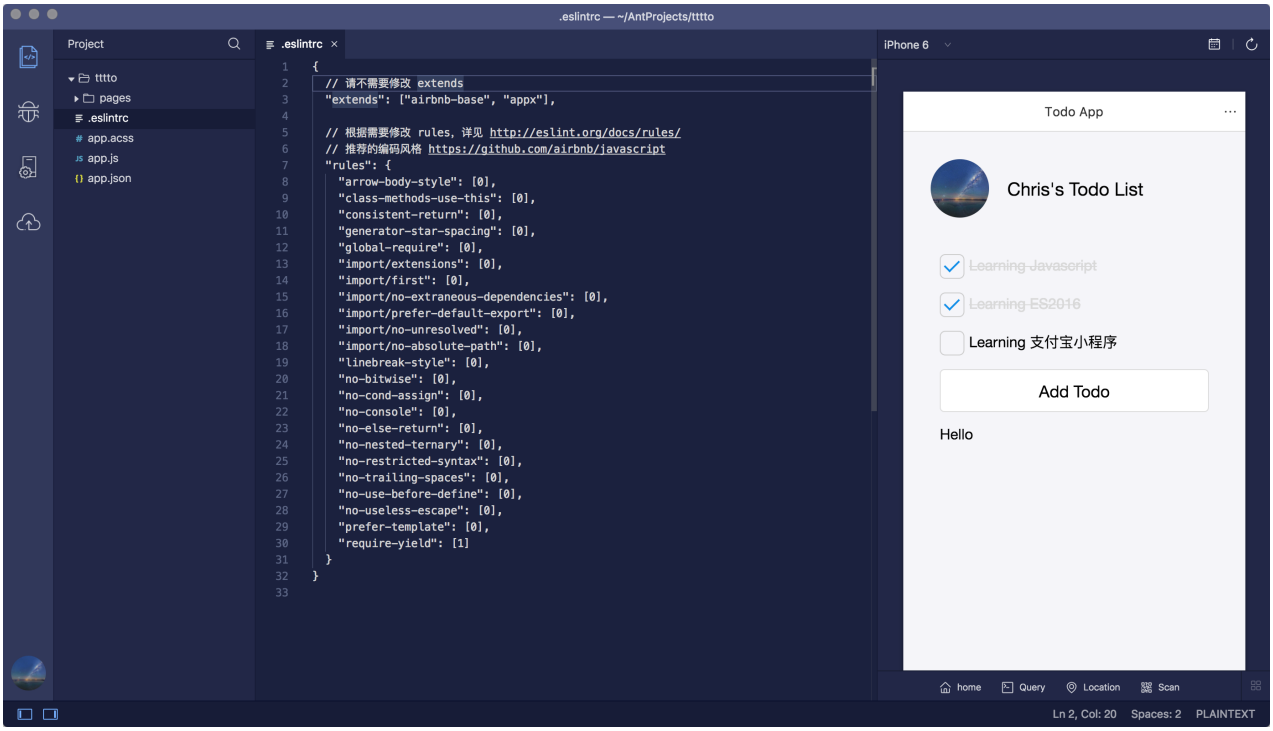
```
a = 123; // 当您忘记使用 var 时，很有可能是创建了一个全局变量 <a>
```

通过 ESLint，您可以第一时间发现这个错误：



因此，建议您遵循内置的 ESLint 规范，这是一套基于 Airbnb Javascript 的规范。如果您觉得这套规范过于严格或者您有特殊的偏好，您也可以进行自定义规范。

小程序项目初始化时，在根目录下，会默认创建 .eslintrc 文件：



您可以修改这份文件并覆盖基于 Airbnb 的默认配置，目前还不支持自定义 extends/plugins。例如，通过设置 "no-console": [0] 来覆盖 Airbnb 默认的配置，把 console warning 去掉。

ESLint 规则

默认配置的 ESLint 为基于 Airbnb 的规则，是业界主流的标准。具体的规则详见 Airbnb JavaScript Style Guide。

您也可以在其基础上自定义规则，在新建的项目中默认提供 .eslintrc 文件，其内容为：

```
{
// 请不需要修改 extends
"extends": ["airbnb-base","appx"],

// 根据需要修改 rules，详见 http://eslint.org/docs/rules/
// 推荐的编码风格 https://github.com/airbnb/javascript
"rules": {
"arrow-body-style": [0],
"class-methods-use-this": [0],
"consistent-return": [0],
"generator-star-spacing": [0],
"global-require": [0],
"import/extensions": [0],
"import/first": [0],
"import/no-extraneous-dependencies": [0],
"import/prefer-default-export": [0],
"import/no-unresolved": [0],
"import/no-absolute-path": [0],
"linebreak-style": [0],
"no-bitwise": [0],
"no-cond-assign": [0],
"no-console": [0],
"no-else-return": [0],
"no-nested-ternary": [0],
"no-restricted-syntax": [0],
"no-trailing-spaces": [0],
"no-use-before-define": [0],
"no-useless-escape": [0],
"prefer-template": [0],
"require-yield": [1]
}
}
```

> **说明**：目前还不支持自定义 extends。

如果您觉得 Airbnb 的规则太过严格，可以把不需要的规则设为 [0]，例如 "no-console": [0]。在 Airbnb 的规则中不推荐在生产环境使用 console，也就是调试时可以写，但是生产环境去掉。由于这条规则过于严格，特此注释，您可以参考调整自己偏好的规则。

另外，针对对齐的问题，您可以直接在编辑器中点击右键并选择 format，可以去掉部分红线。

8 框架

8.1 概述

文件结构

小程序分为 app 和 page 两层。app 用来描述整个应用，page 用来描述各个页面。

app 由三个文件组成，必须放在项目的根目录。

文件	必填	作用
app.js	是	小程序逻辑
app.json	是	小程序全局设置
app.acss	否	小程序全局样式表

page 由四个文件组成，分别是：

文件类型	必填	作用
js	是	页面逻辑
axml	是	页面结构
acss	否	页面样式表
json	否	页面配置

说明：为了方便开发者，我们规定这四个文件必须具有相同的路径与文件名。

开发者写的所有代码最终将会打包成一份 JavaScript 脚本，在小程序启动的时候运行，在小程序结束运行时销毁。

逻辑结构

小程序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: 'taobao',
};

// 注册一个页面
Page({
data: initialData,
changeName(e) {
// 改变数据
this.setData({
name: 'mPaaS',
```

```
});
},
});
```

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 taobao!。

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 taobao 变为 mPaaS ，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 mPaaS!。

注意： 由于框架并非运行在浏览器中，所以 JavaScript 在 web 中的一些能力都无法使用，如 document、window 等对象。

逻辑层 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
```

第三方 NPM 模块

小程序支持引入第三方模块，需先在小程序根目录下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。同时，浏览器相关 web 能力同样无法使用。

8.2 应用

App 代表顶层应用，管理所有页面和全局数据，并提供生命周期方法。它也是一个构造方法，生成 App 实例。一个小程序就是一个 App 实例。

简介

每个小程序的顶层一般包含三个文件：

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

下面是一个简单的 app.json：

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

在上述配置中，指定了小程序包含两个页面，以及应用窗口的默认标题是 Demo。

App 提供四个事件，可以设置钩子方法：

- onLaunch：小程序启动
- onShow：小程序切换到前台
- onHide：小程序切换到后台
- onError: 小程序出错

一个简单的 app.js 代码如下：

```
App({
onLaunch(options) {
// 小程序初始化
},
onShow(options) {
// 小程序显示
},
onHide() {
// 小程序隐藏
},
onError(msg) {
console.log(msg)
},
globalData: {
foo: true,
}
})
```

App()

App() 接受一个 object 作为参数，用来配置小程序的生命周期等。

参数说明：

属性	类型	描述	触发时机
onLaunch	Function	监听小程序初始化	当小程序初始化完成时触发，全局只触发一次
onShow	Function	监听小程序显示	当小程序启动，或从后台进入前台显示时触发
onHide	Function	监听小程序隐藏	当小程序从前台进入后台时触发
onError	Function	监听小程序错误	当小程序发生 js 错误时触发

前台、后台定义： 用户点击左上角关闭，或者按了设备 Home 键离开 mPaaS 客户端时，小程序并不会直接销毁，而是进入了后台，当再次进入 mPaaS 客户端或再次打开小程序时，又会从后台进入前台。

只有当小程序进入后台一定时间后，或占用系统资源过高时，才会被真正销毁。

onLaunch/onShow 方法的参数

属性	类型	描述
query	Object	当前小程序的 query
path	String	当前小程序的页面地址

注意：

Native启动传参方法如下：

```
// 启动小程序demo
Bundle param = new Bundle();
String queryParam = "param1=value1&param2=value2&param3=value3";
param.putString("query", queryParam);
LauncherApplicationAgent.getInstance().getMicroApplicationContext()
        .startApp( s: null,  s1: "2018080616290001", param);
```

Url启动传参方法如下：

query 从启动参数的 query 字段解析而来, path 从启动参数 page 字段解析而来。 例如在如下 URL 中：

```
alipays://platformapi/startapp?appId=1999&query=number%3D1&page=x%2Fy%2Fz
```

- 其中的 query 参数解析如下：

```
number%3D1 === encodeURIComponent('number=1')
```

其中的 path 参数解析如下：

```
x%2Fy%2Fz === encodeURIComponent('x/y/z')
```

page 忽略时默认为首页

那么，当用户第一次启动小程序可以从 onLaunch 方法中获取这个参数，或者小程序在后台时被重新用 schema 打开也可以从 onShow 方法中获取这个参数。

```
App({
onLaunch(options) {
// 第一次打开
// options.query == {number:1}
},
```

```
onShow(options) {
// 从后台被 scheme 重新打开
// options.query == {number:1}
},
})
```

getApp()

我们提供了全局的 getApp() 函数，可以获取小程序实例，一般用在各个子页面之中获取顶层应用。

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

注意：

- App() 必须在 app.js 里调用，且不能调用多次。
- 不要在定义于 App() 内定义的函数中调用 getApp()，使用 this 就可以拿到 app 实例。
- 不要在 onLaunch 里调用 getCurrentPages()，这个时候 page 还没有生成。
- 通过 getApp() 获取实例之后，不要私自调用生命周期函数。

全局的数据可以在 App() 中设置，各个子页面通过全局函数 getApp() 可以获取全局的应用实例。例如：

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


// a.js

// localValue 只在 a.js 有效
var localValue = 'a'

// 生成 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)
```

在上面代码中，a.js 和 b.js 都声明了变量 localValue，它们不会互相影响，因为各个脚本声明的变量和函数只在该文件中有效。

app.json

app.json 用于全局配置，决定页面文件的路径、窗口表现、设置网络超时时间、设置多 tab 等。

以下是一个包含了部分配置选项的简单配置 app.json。

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

app.json 配置项如下。

文件	类型	必填	描述
pages	String 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 tab 的表现

pages

pages属性是一个数组，每一项都是字符串，用来指定小程序的页面。每一项代表对应页面的路径信息，数组的第一项代表小程序的首页。小程序中新增/减少页面，都需要对 pages数组进行修改。

页面路径不需要写 js 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/
pages/index/index.axml
pages/index/index.js
pages/index/index.acss
pages/logs/logs.axml
pages/logs/logs.js
app.js
app.json
app.acss
```

app.json就要写成下面的样子。

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

window

window属性用于设置小程序通用的的状态栏、导航条、标题、窗口背景色。

子属性包括 titleBarColor defaultTitle pullRefresh allowsBounceVertical。

文件	类型	必填	描述
titleBarColor	十进制	否	导航栏背景色
defaultTitle	String	否	页面标题
pullRefresh	Boolean	否	是否允许下拉刷新。默认 false
allowsBounceVertical	String(YES/NO)	否	页面是否支持纵向拽拉超出实际内容。默认 YES

下面是一个例子。

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

tabBar

如果你的小程序是一个多 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。另外，tabBar 的第一个页面必须是首页。

tabBar 配置

文件	类型	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

文件	类型	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 推荐大小为 60*60px 大小，系统会对任意传入的图片非等比拉伸/缩放。

例如

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

启动参数

从 native 代码中打开小程序时可以带上 page 和 query 参数，page 用来指定打开特定页面的路径，query 用来带入参数。

- iOS 示例代码

```
NSDictionary *param = @{@"page":@"pages/card/index", @"query":@"own=1&sign=1&code=2452473"};
[DTContextGet() startApplication:@"1234567891234568"params:param animated:YES];
```

- Android 示例代码

```
Bundle param = new Bundle();
param.putString("page","pages/card/index");
param.putString("query","own=1&sign=1&code=2452473");
LauncherApplicationAgent.getInstance().getMicroApplicationContext().startApp(null,"1234567891234568", param);
```

8.3 页面

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。

页面初始化

页面初始化时，需要提供数据作为页面的第一次渲染：

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

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

定义交互行为时，需要在页面脚本中指定响应函数：

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

上面模板定义用户点击时，调用了 handleTap 方法：

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

页面重新渲染，需要在页面脚本中调用 this.setData 方法。

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

上面代码指定用户触摸按钮时，调用了 changeText 方法。

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

上面代码中，在 changeText 方法中调用了 this.setData 方法，会导致页面的重新渲染。

Page()

Page() 接受一个 object 作为参数，该参数用来指定页面的初始数据、生命周期函数、事件处理函数等。

```
//index.js
Page({
data: {
title:"Alipay"
```

```
},
onLoad(query) {
// 页面加载
},
onReady() {
// 页面加载完成
},
onShow() {
// 页面显示
},
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
},
viewTap() {
// 事件处理
this.setData({
text: 'Set data for updat.'
})
},
go() {
// 带参数的跳转，从 page/index 的 onLoad 函数的 query 中读取 xx
my.navigateTo('/page/index?xx=1')
},
customData: {
hi: 'alipay'
}
})
```

上面的代码中，Page() 方法的参数对象说明如下：

属性	类型	描述
data	Object or Function	初始数据或返回初始化数据的函数
onTitleClick	Function	点击标题触发
onOptionMenuClick	Function	基础库 1.3.0+ 支持，点击格外导航栏图标触发，可通过 my.canIUse('page.onOptionMenuClick') 判断
onPageScroll	Function({scrollTop})	页面滚动时触发
onLoad	Function(query: Object)	页面加载时触发

onReady	Function	页面初次渲染完成时触发
onShow	Function	页面显示时触发
onHide	Function	页面隐藏时触发
onUnload	Function	页面卸载时触发
onPullDownRefresh	Function	页面下拉时触发
onReachBottom	Function	上拉触底时触发
onShareAppMessage	Function	点击右上角分享时触发
其他	Any	开发者可以添加任意的函数或属性到 object 参数中，在页面的函数中可以用 this 来访问

说明：data 为对象时，如果您在页面中修改 data，会影响该页面的不同实例。

生命周期方法

- onLoad：页面加载。一个页面只会调用一次，query 参数为 my.navigateTo 和 my.redirectTo 中传递的 query 对象。
- onShow：页面显示。每次页面显示都会调用一次。
- onReady：页面初次渲染完成。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。当使用 my.navigateTo 跳转到其他页面或在底部使用 tab 切换 tab 时调用。
- onUnload：页面卸载。当使用 my.redirectTo 或 my.navigateBack 跳转到其他页面的时候调用。

事件处理函数

- onPullDownRefresh：下拉刷新。监听用户下拉刷新事件，需要在 app.json 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。
- onShareAppMessage：用户分享，详见 分享 。

Page.prototype.setData()

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

说明：

- 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。
- 请尽量避免一次设置过多的数据。

setData 接受一个对象作为参数。对象的键名 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，并且不需要在 this.data 中预先定义。

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>


Page({
data: {
text: 'test',
array: [{text: 'a'}],
object: {
text: 'blue'
}
},
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'
})
}
})
```

getCurrentPages()

getCurrentPages() 函数用于获取当前页面栈的实例，以数组形式按栈的顺序给出，第一个元素为首页，最后一个元素为当前页面。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('/xx');
} else {
my.navigateTo('/xx');
```

```
}
```

注意：不要尝试修改页面栈，否则会导致路由以及页面状态错误。

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

路由方式	页面栈表现
初始化	新页面入栈
打开新页面	新页面入栈
页面重定向	当前页面出栈，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

page.json

每一个页面也可以使用 [page名].json 文件来对本页面的窗口表现进行配置。

页面的配置比 app.json 全局配置简单得多，只能设置 window 相关的配置项，所以无需写 window 这个键。注意，页面配置会覆盖 app.json 的 window 属性中的配置项。

格外支持 optionMenu 配置导航图标，点击后触发 onOptionMenuClick。

文件	类型	必填	描述
optionMenu	Object	否	基础库 1.3.0+ 支持，设置导航栏格外图标，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

例如：

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

page 样式

每个页面中的根元素为 page，需要设置高度或者背景色时，可以利用这个元素。

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

8.4 视图层

简介

视图文件的后缀名是 axml，定义了页面的标签结构。

下面通过一些例子展示 axml 具有的能力。

数据绑定：

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

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

列表渲染：

```
<view a:for="{{items}}"> {{item}} </view>

// page.js
Page({
data: {
items: [1, 2, 3, 4, 5, 6, 7]
}
})
```

条件渲染：

```
<view a:if="{{view == 'WEBVIEW'}}"> WEBVIEW </view>
<view a:elif="{{view == 'APP'}}"> APP </view>
<view a:else="{{view == 'alipay'}}"> alipay </view>

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

模板：

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

```
</view>
</template>

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


// page.js
// Hats off to the Wechat Mini Program engineers.
Page({
data: {
staffA: {firstName: 'san', lastName: 'zhang'},
staffB: {firstName: 'si', lastName: 'li'},
staffC: {firstName: 'wu', lastName: 'wang'},
},
})
```

事件：

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


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

数据绑定

axml 中的动态数据均来自对应 Page 的 data。

简单绑定

数据绑定使用 Mustache 语法（双大括号）将变量包起来，可以作用于各种场合。

作用于内容，例如：

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


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

作用于组件属性（需要在双引号之内），例如：

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

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

作用于控制属性（需要在双引号之内），例如：

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

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

作用于关键字（需要在双引号之内），例如：

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

- true：boolean 类型的 true，代表真值。

  false： boolean 类型的 false，代表假值。

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

可以在 {{}} 内进行简单的运算，支持的有如下几种方式：

三元运算：

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

算数运算：

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

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

View 中的内容为3 + 3 + d。

逻辑判断：

```
<view 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']
}
})
```

也可以在 Mustache 内直接进行组合，构成新的数组或者对象。

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}。

也可以用扩展运算符...来将一个对象展开。

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


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

最终组合成的对象是 {a: 1, b: 2, c: 3, d: 4, e: 5}。

如果对象的 key 和 value 相同，也可以间接地表达。

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


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

```
}
})
```

最终组合成的对象是 {foo: ‘my-foo’ , bar:’ my-bar’ }。

说明：上面的方式可以随意组合，但是如有存在变量名相同的情况，后边的变量会覆盖前面变量。

```
<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}。

条件渲染

a:if

在框架中，您可以使用 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 来添加一个 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 是一个控制属性，需要将它添加到一个标签上。如果想一次性判断多个组件标签，您可以使用一个 <block/> 标签将多个组件包装起来，并在它的上边使用 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，仅仅是一个包装元素，不会在页面中做任何渲染，只接受控制属性。

列表渲染

a:for

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

默认数组的当前项的下标变量名默认为 index，数组当前项的变量名默认为 item。

```
<view 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 也可以嵌套，下方是九九乘法表的代码示例：

```
<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

类似 block a:if，您也可以将 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

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

a:key 的值以两种形式来提供：

- 字符串，代表在 for 循环的 array 中 item 的某个属性。该属性的值需要是列表中唯一的字符串或数字，并且不能动态的改变。
- 保留关键字 *this，代表在 for 循环中的 item 本身，表示需要 item 本身是唯一的字符串或者数字。比如当数据改变触发渲染层重新执行渲染的时候，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，确保使组件保持自身的状态，并且提高列表渲染时的效率。

如果明确知道列表是静态，或者不关注其顺序，则可以选择忽略。

代码示例如下：

```
<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。

代码示例如下：

```
<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 });
}
}
});
```

同时可以利用 key 来防止组件的复用。例如，如果允许用户输入不同类型的数据：

```
<input a:if="{{name}}"placeholder="Enter your username">
<input a:else placeholder="Enter your email address">
```

那么当你输入 name 然后切换到 email 时，当前输入值会保留，如果不想保留，可以加 key：

```
<input key="name"a:if="{{name}}"placeholder="Enter your username">
<input key="email"a:else placeholder="Enter your email address">
```

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

注意 template 的子节点只能是一个而不是多个，例如：

- 允许

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

- 而不允许

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

include

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

代码示例如下：

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


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


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

模板

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

定义模板

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

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

使用模板

使用 is 属性，声明需要的使用的模板，然后将该模板所需要的 data 传入，比如：

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


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

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

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

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

说明：模板拥有自己的作用域，只能用 data 传入的数据，但可以通过 onXX 绑定页面的逻辑处理函数。

推荐用 template 方式来引入模版片段，因为 template 会指定自己的作用域，只使用 data 传入的数据，因此小程序会对此进行优化。如果该 template 的 data 没有改变，该片段 UI 并不会重新渲染。

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

```
<import src="./a.axml"/> <!-- 相对路径 -->
<import src="/a.axml"/> <!-- 项目绝对路径 -->
<import src="third-party/x.axml"/> <!-- 第三方 npm 包路径 -->
```

8.5 事件

什么是事件？

- 事件是视图层到逻辑层的通讯方式。
- 事件可以将用户的行为反馈到逻辑层进行处理。
- 事件可以绑定在组件上，当达到触发条件，就会执行逻辑层中对应的事件函数。
- 事件对象可以携带额外信息，如 id, dataset, touches。

使用方式

事件分为 **冒泡事件** 和 **非冒泡事件**：

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

事件绑定的写法同组件的属性，为 key、value 的形式。

- key 以 on 或 catch 开头，再加上事件的类型，如 onTap、catchTap。
- value 是一个字符串，需要在对应的 Page 中定义同名的函数。否则当触发事件时会报错。

on 事件绑定不会阻止冒泡事件向上传递，catch 事件绑定可以阻止冒泡事件向上传递。

```
<view id="outter"onTap="handleTap1">
view1
```

```
<view id="middle"catchTap="handleTap2">
view2
<view id="inner"onTap="handleTap3">
view3
</view>
</view>
</view>
```

在上面的代码中，点击 view3 会先后触发 handleTap3 和 handleTap2（因为 tap 事件会传递到 view2，而 view2 阻止了 tap 事件冒泡，不再向父节点传递）。点击 view2 会触发 handleTap2，点击 view1 会触发 handleTap1。

冒泡事件列表：

类型	触发条件
touchStart	触摸动作开始
touchMove	触摸后移动
touchEnd	触摸动作结束
touchcancel	触摸动作被打断，如来电提醒，弹窗
tap	触摸后马上离开
longTap	触摸后，超过 300 ms 再离开

其他事件则不冒泡：

在组件中绑定一个事件处理函数。

如 onTap，当用户点击该组件的时候会在该页面对应的 Page 中找到对应的事件处理函数。

```
<view id="tapTest"data-hi="Alipay"onTap="tapName">
<view id="tapTestInner"data-hi="AlipayInner">
Click me!
</view>
</view>
```

在相应的 Page 定义中写上相应的事件处理函数，参数是 event：

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

可以看到 log 出来的信息大致如下：

```
{
"type":"tap",
```

```
"timeStamp": 13245456,
"target": {
"id":"tapTestInner",
"dataset": {
"hi":"Alipay"
},
"targetDataset": {
"hi":"AlipayInner"
}
},
"currentTarget": {
"id":"tapTest",
"dataset": {
"hi":"Alipay"
}
}
}
```

事件对象

当组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。

BaseEvent：基础事件对象属性列表。

属性	类型	描述
type	String	事件类型
timeStamp	Integer	事件生成时的时间戳
target	Object	触发事件的组件的属性值集合

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

属性	类型	描述
detail	Object	额外的信息

TouchEvent：触摸事件对象属性列表（继承 BaseEvent）。

属性	类型	描述
touches	Array	当前停留在屏幕中的触摸点信息的数组
changedTouches	Array	当前变化的触摸点信息的数组

Type：事件的类型。

timeStamp：页面打开到触发事件所经过的毫秒数。

target：触发事件的源组件。

属性	类型	描述

id	String	事件源组件的 ID
tagName	String	当前组件的类型
dataset	Object	绑定事件的组件上由 data- 开头的自定义属性的集合
targetDataset	Object	实际触发事件的组件上由 data- 开头的自定义属性的集合

dataset

在组件中可以定义数据，这些数据将会通过事件传递给逻辑层。

书写方式： 以 data- 开头，多个单词由连字符（-）链接，不能有大写（大写会自动转成小写），如 data-element-type，最终会在 event.target.dataset 中会将连字符转成驼峰 elementType。

代码示例：

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


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

touches

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

Touch 对象

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

CanvasTouch 对象

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

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

detail：自定义事件所携带的数据，如表单组件的提交事件会携带用户的输入信息，媒体的错误事件会携带错误信息，详细的描述请参考组件定义中各个事件的定义。

8.6 样式

acss（AntFinancial Style Sheet）是一套样式语言，用于描述 axml 页面的组件样式，决定 axml 的组件应该如何显示。

为了适应广大的前端开发者，我们的 acss 具有 CSS 大部分特性。同时为了更适合开发小程序，我们对 CSS 进行了扩充。

与 CSS 相比，acss 的扩展的特性有：

rpx：rpx（responsive pixel）可以根据屏幕宽度进行自适应。规定屏幕宽为 750 rpx。如在 iPhone6 上，屏幕宽度为 375 px，共有 750 个物理像素，则 750 rpx = 375px = 750 物理像素，1 rpx = 0.5 px = 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	1 rpx = 0.42 px	1 px = 2.34 rpx
iPhone6	1 rpx = 0.5 px	1 px = 2 rpx
iPhone6 Plus	1 rpx = 0.552 px	1 px = 1.81 rpx

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的相对路径，用分号（;）表示结束。

代码示例：

```
/** 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/button.acss"; /*第三方 npm 包路径*/
```

内联样式：组件支持使用 style、class 属性来控制样式。

- style 属性：静态的样式统一写到 class 中。style 接收动态的样式，样式在运行时会进行解析

。请尽量避免将静态的样式写进 style 中，以免影响渲染速度。

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

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

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

选择器：与 CSS3 保持一致。

注意：

- 以 .a-、.am- 开头的类选择器为系统组件占用，请不要使用。
- 不支持属性选择器。

全局样式与局部样式：定义在 app.acss 中的样式为全局样式，作用于每一个页面。在 Page 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。

页面容器样式：可以通过 page 元素选择器来设置页面容器的样式，比如页面背景色:

```
page {
background-color: red;
}
```

8.7 小程序全局配置

8.7.1 小程序全局配置介绍

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。

一个小程序就是一个 App 实例。每个小程序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）。

示例代码

一个简单的 app.json 代码如下：

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

这段代码配置指定小程序包含两个页面（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) {
// 小程序启动，或从后台被重新打开
},
onHide() {
// 小程序从前台进入后台
},
onError(msg) {
// 小程序发生脚本错误或 API 调用出现报错
console.log(msg);
},
globalData: {
// 全局数据
name: 'mPaaS',
},
});
```

8.7.2 app.json 全局配置

app.json 用于对小程序进行全局配置，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，用于指定小程序的页面。在小程序中新增或删除页面，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，第一项代表小程序的首页。

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
```

那么 app.json 就要写成：

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

window

window 用于设置小程序的状态栏、导航条、标题、窗口背景色等。

属性	类型	是否必填	描述
defaultTitle	String	否	页面默认标题
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
titlePenetrate	String	否	是否允许导航栏点击穿透。默认 NO，支持 YES / NO
showTitleLoading	String	否	是否进入时显示导航栏的 loading。默认 NO，支持 YES / NO
titleImage	String	否	导航栏图片地址
titleBarColor	HexColor	否	导航栏背景色
backgroundColor	HexColor	否	下拉露出显示的背景颜色
backgroundImageColor	HexColor	否	下拉露出显示的背景图底色
backgroundImageUrl	String	否	下拉露出显示的背景图链接
gestureBack	String	否	iOS 用，是否支持手势返回。默认 NO，支持 YES / NO
enableScrollBar	Boolean	否	Android 用，是否显示 WebView 滚动条。默认 YES，支持 YES / NO

代码示例：

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

tabBar

如果您的小程序是一个多 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。

说明：

- 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 配置如下：

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，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"
}
]
}
}
```

8.7.3 app.acss 全局样式

app.acss 作为全局样式，作用于当前小程序的所有页面。有关 **acss** 的详细内容，参见 ACSS 语法参考 。

8.7.4 app.js 注册小程序

App(object: Object)

- App() 用于注册小程序，接受一个 Object 作为属性，用来配置小程序的生命周期等。
- 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：监听小程序初始化	当小程序初始化完成时触发，全局只触发一次。
onShow	Function	生命周期回调：监听小程序显示	当小程序启动，或从后台进入前台显示时触发。
onHide	Function	生命周期回调：监听小程序隐藏	当小程序从前台进入后台时触发。
onError	Functio	监听小程序错误	当小程序发生 JS 错误时触发。

	n		
onShareAppMessage	Function	全局分享配置	

前台/后台定义：

- 小程序用户点击右上角关闭，或者按下设备 Home 键离开支付宝时，小程序并不会直接销毁，而是进入后台。
- 当用户再次进入支付宝或再次打开小程序时，小程序会从后台进入前台。
- 只有当小程序进入后台一定时间，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	当前小程序的 query，从启动参数的 query 字段解析而来。
path	String	当前小程序的页面地址，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

例如，启动小程序的 schema url 如下:

```
alipays://platformapi/startapp?appId=1999&query=number%3D1&page=x%2Fy%2Fz
```

参数解析如下:

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

- 小程序首次启动时，onLaunch 方法可获取 query、path 属性值。
- 小程序在后台被用 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	来源小程序	
sourceServiceId	String	来源插件，当处于插件运行模式时可见	1.11.0
extraData	Object	来源小程序传过来的数据。	

说明：

- 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() ，因为此时 page 还未生成。

onHide()

小程序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)

小程序发生脚本错误或 API 调用报错时触发。

```
App({
onError(error) {
// 小程序执行出错时
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，具体内容参见分享 。

globalData 全局数据

App() 中可以设置全局数据 globalData。

代码示例：

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

8.7.5 getApp 方法

小程序提供了全局的 getApp() 方法，可获取当前小程序实例，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，您需要注意以下几点：

- App() 函数中不可以调用 getApp()，可使用 this 以获取当前小程序实例。
- 通过 getApp() 获取实例后，请勿私自调用生命周期回调函数。

  需区分全局变量及页面局部变量，例如：

  ```
  // 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。

8.8 小程序页面

8.8.1 小程序页面介绍

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。

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

页面初始化时，提供以下数据：

```
Page({
data: {
title: 'mPaaS',
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，需要在页面脚本中调用 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',
});
```

```
},
});
```

以上代码中，在 changeText 方法中调用 this.setData 方法，会导致页面重新渲染。

8.8.2 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 键。页面配置项会优先于全局配置项。

同时支持以下几点：

支持 optionMenu 配置导航图标，点击后触发 onOptionMenuClick。

说明：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。

完整配置项如下：

文件	类型	必填	描述
optionMenu	Object	否	基础库 1.3.0+ 支持，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	客户端 10.1.52+ 支持，设置导航栏点击穿透。

以下为一个基本示例：

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

8.8.3 页面结构

在 /pages 目录中的 .axml 文件用于定义当前页面的结构。

文件内容遵循 AXML 语法。与 HTML 非常相似，但也有不同之处，详细内容参见 AXML 。

8.8.4 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，可按如下方式进行设置：

```
page {
```

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

有关 **acss** 的更多内容，参见 ACSS 语法参考 。

8.8.5 页面注册

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、事件处理等信息。

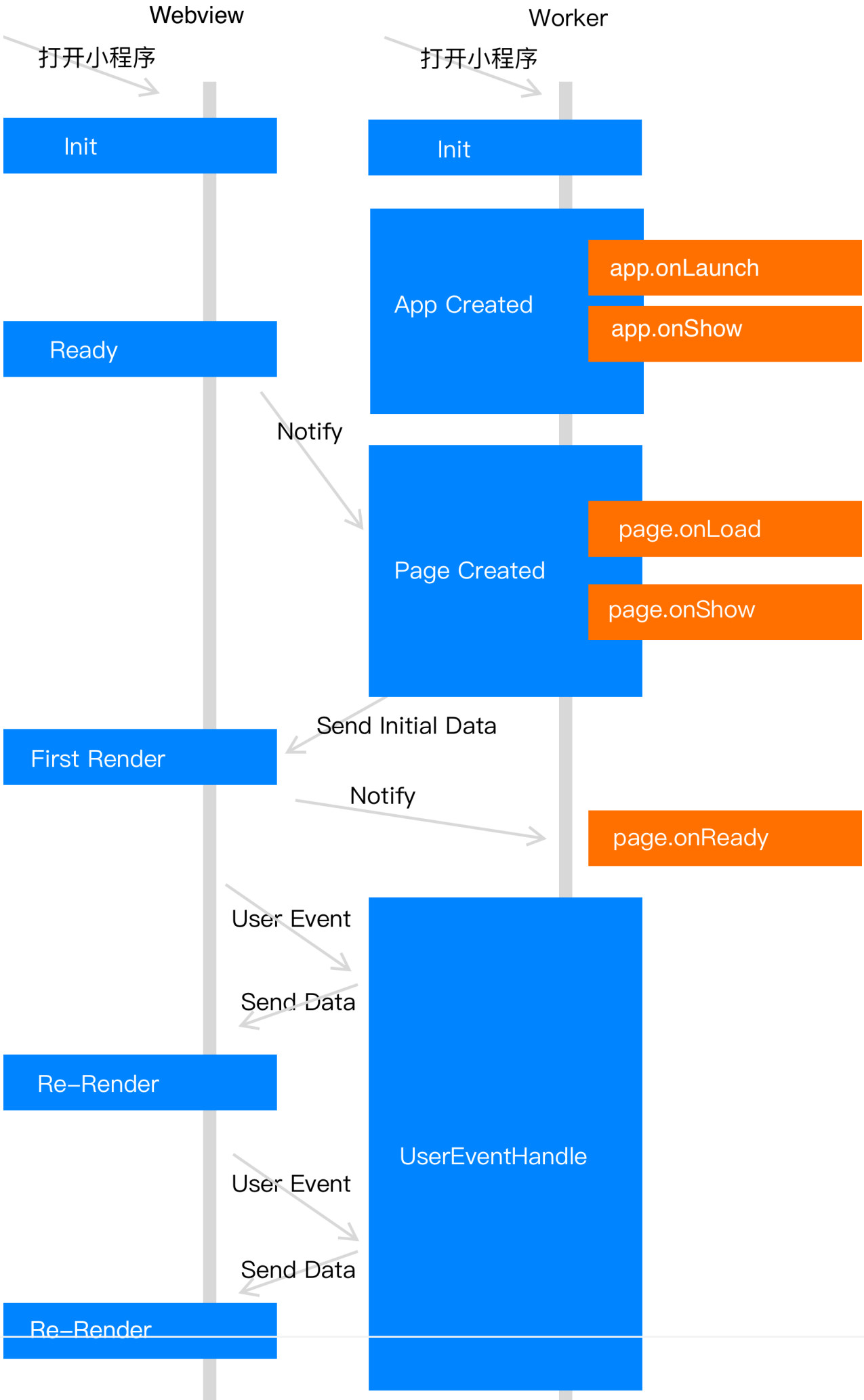
以下为一个基本的页面代码：

```
// pages/index/index.js
Page({
data: {
title:"Alipay",
},
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: 'alipay',
},
});
```

页面生命周期

下图说明了页面 Page 对象的生命周期。


Webview
Worker
打开小程序
打开小程序
Init
Init
App Created
app.onLaunch
app.onShow
Ready
Notify
Page Created
page.onLoad
page.onShow
Send Initial Data
First Render
Notify
page.onReady
User Event
Send Data
Re-Render
UserEventHandle
User Event
Send Data
Re-Render

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。

1. 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。
2. 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。
3. 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。
4. 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	最低版本
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	1.13.7
onLoad	Function(query: Object)	页面加载时触发	
onShow	Function	页面显示时触发	
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	1.3.0
onPopMenuClick	Function		1.3.0
onPullDownRefresh	Function({from: manual|code})	页面下拉时触发	
onPullIntercept	Function	下拉中断时触发	1.11.0
onTabItemTap	Function	点击tabItem时触发	1.11.0
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。对于这种情况，可以通过以下两种方式解决：

设置 data 为不可变数据：

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

说明：不要直接修改 this.data，这样做不仅无法改变页面的状态，还会造成数据不一致。

例如以下错误示例：

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

- 设置 data 为页面独有数据（不推荐）：

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

生命周期函数

onLoad(query: Object)

页面加载时触发。一个页面只会调用一次。query 为 my.navigateTo 和 my.redirectTo 中传递的 query 对象。

属性	类型	描述	最低版本
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，需在 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。详见 分享 。

onTitleClick()

点击标题触发。

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 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	最低版本
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

说明：为了使代码更加简洁，提供了新的事件处理对象 events。原同名事件跟直接在 page 实例上暴露的事件函数等价。events 从 **1.13.7** 开始支持。

事件	类型	描述	最低版本
onBack	Function	页面返回时触发	1.13.7
onKeyboardHeight	Function	键盘高度变化时触发	1.13.7
onOptionMenuClick	Function	点击右上角菜单按钮触发	1.13.7
onPopMenuClick	Function	点击右上角通用菜单按钮触发	1.13.7
onPullIntercept	Function	下拉截断时触发	1.13.7
onPullDownRefresh	Function({from: manual|code})	页面下拉时触发	1.13.7
onTitleClick	Function	点击标题触发	1.13.7
onTabItemTap	Function	点击且切换tabItem后触发	1.13.7
beforeTabItemTap	Function	点击但切换tabItem前触发	1.13.7

示例代码：

```
Page({
data: {
text: 'This is page data.'
},
onLoad(){
// 设置自定义菜单
my.setCustomPopMenu({
menus:[
{name: '菜单1', menuIconUrl: 'https://menu1'},
{name: '菜单2', menuIconUrl: 'https://menu2'},
],
})
},
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” 时表示 startPullDownRefresh 触发的事件；值是 “manual” 时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” 时表示用户点击触发的事件；值是 “api” 时表示 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
},
}
})
```

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。

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

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：

- 直接修改 this.data 无效，不仅无法改变页面的状态，还会造成数据不一致。
- 仅支持设置可 JSON 化的数据。
- 尽量避免一次设置过多的数据。
- 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"> Chane name </button>


Page({
data: {
text: 'test',
array: [{text: 'a'}],
object: {
text: 'blue',
},
name: 'taobao',
},
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: 'alipay',
}, () => { // 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!'});
});
},
});
```

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。详情参见 小程序基础库说明 。

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。

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。	

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（格式为：[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
```

Page.prototype.$batchedUpdates(callback: Function)

批量更新数据。

说明：$spliceData 自 1.14.0 之后才支持，可以使用 **my.canIUse(‘page.\$batchedUpdates’)** 做兼容性处理。详情参见 小程序基础库说明 。

参数说明：

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

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>
```

在上方示例代码中：

- 每次点击按钮，页面的 counter 会加 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。

8.8.6 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('/pages/logs/logs');
} else {
my.navigateTo('/pages/index/index');
}
```

说明：不要尝试修改页面栈，否则会导致路由以及页面状态错误。

8.9 AXML

8.9.1 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,
});
},
});
```

8.9.2 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>
```

说明：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'}。

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}。

8.9.3 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。

8.9.4 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。

说明：

- 如不提供 a:key，会报错。
- 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。

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。

说明：key 不能设置在 block 上。

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 });
}
},
});
```

同时可以利用 key 来防止组件复用，例如允许用户输入不同类型数据：

```
<input a:if="{{name}}"placeholder="Enter your name"/>
<input a:else placeholder="Enter your email address"/>
```

那么当输入 name 然后切换到 email 时，当前输入值会保留，如果不想保留，可以加 key：

```
<input key="name"a:if="{{name}}"placeholder="Enter your name"/>
<input key="email"a:else placeholder="Enter your email address"/>
```

8.9.5 模板

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

说明：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)
},
});
```

8.9.6 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。

例如，在 item.axml 中定义了一个名为 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。

例如，C import B，B import A，在 C 中可以使用 B 定义的 template，在 B 中可以使用 A 定义的 template，但是 C 不能使用 A 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"/>
```

template 的子节点只能是一个，例如：

正确示例：

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

错误示例：

```
<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"/> <!-- 第三方 npm 包路径 -->
```

8.10 SJS 语法参考

8.10.1 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，故二者语法并不一致，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 alipay';
const getMsg = x => x;
export default {
message,
getMsg,
};


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


<!-- 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 alipay
hello taobao
```

说明：

- 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

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。

示例代码：

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


// 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 alipay
3
4
```

说明：

- 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。

8.10.2 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
```

8.10.3 注释

注释方法与 Javascript 一致，您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 = ' alipay';
```

8.10.4 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。

8.10.5 语句

if 语句

在 .js 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。

语法示例：

```
// 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

语法示例：

```
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。

代码示例：

```
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

语法示例：

```
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。

代码示例：

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

输出：

```
0
1
```

while 语句

语法示例：

```
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。

8.10.6 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 alipay';
"hello taobao";
```

ES6 语法

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

属性

- constructor: 返回值 "String"。
- length

说明：除 constructor 外的属性的具体含义参考 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 外的属性的具体含义参考 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 外的属性的具体含义参考 ES5 标准。

方法

- exec
- test
- toString

说明：具体使用参考 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);
```

8.10.7 基础类库

Global

说明：SJS 不支持 JavaScript 的大部分全局属性和方法。

属性

- Infinity
- NaN
- undefined

说明：具体使用参考 ES5 标准。

方法

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

说明：具体使用参考 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

说明：具体使用参考 ES5 标准。

Number

属性

- MAX_VALUE
- MIN_VALUE
- NEGATIVE_INFINITY
- POSITIVE_INFINITY

说明：具体使用参考 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

说明：具体使用参考 ES5 标准。

方法

- abs
- acos
- asin
- atan
- atan2

- ceil
- cos
- exp
- floor
- log
- max
- min
- pow
- random
- round
- sin
- sqrt
- tan

说明：具体使用参考 ES5 标准。

8.10.8 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";
},
};
```

说明：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
```

8.11 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 同系统 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。

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。

```
<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 .，多个以空格分隔。

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

静态样式统一写到 class 中，避免将静态样式写进 style 中，以免影响渲染速度。

选择器

同 CSS3 保持一致。

说明：

- 以 .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');
```

8.12 事件系统

8.12.1 事件介绍

- 事件是视图层到逻辑层的通讯方式。
- 事件可以将用户的行为反馈到逻辑层进行处理。
- 事件可以绑定在组件上，当达到触发条件，就会执行逻辑层中对应的事件函数。
- 事件对象可以携带额外信息，如 id、dataset、touches。

使用方式

若要在组件中绑定一个事件处理函数，如 onTap，则需要在该页面的 .js 文件中的 Page 里定义onTap对应的事件处理函数。

```
<view id="tapTest"data-hi="Alipay"onTap="tapName">
<view id="tapTestInner"data-hi="AlipayInner">
Click me!
</view>
</view>
```

在相应的 Page 中定义相应的事件处理函数 tapName，参数为事件对象 event。

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

控制台输出 event 信息如下所示:

```
{
"type":"tap",
"timeStamp": 1550561469952,
"target": {
"id":"tapTestInner",
"dataset": {
"hi":"Alipay"
},
"targetDataset": {
"hi":"AlipayInner"
}
},
"currentTarget": {
"id":"tapTest",
"dataset": {
```

```
"hi":"Alipay"
}
}
}
```

使用组件（基础组件、扩展组件和自定义组件）时，组件里有哪些可用的事件取决于组件本身是否支持，支持的事件会在具体组件的文档里明确列出。

事件类型

事件分为冒泡事件和非冒泡事件：

- 冒泡事件：以关键字 on 为前缀，当组件上的事件被触发，该事件会向父节点传递。
- 非冒泡事件：以关键字 catch 为前缀，当组件上的事件被触发，该事件不会向父节点传递。

事件绑定的写法同组件的属性，以 key、value 的形式。

- key 以 on 或 catch 开头，然后跟上事件的类型，如 onTap、catchTap。
- value 是一个字符串，对应 Page 中定义的函数名，不存在时触发事件会报错。

```
<view id="outter"onTap="handleTap1">
view1
<view id="middle"catchTap="handleTap2">
view2
<view id="inner"onTap="handleTap3">
view3
</view>
</view>
</view>
```

上述代码中，点击 view3 会先后触发 handleTap3 和 handleTap2（因为 tap 事件会冒泡到 view2，而 view2 阻止了 tap 事件冒泡，不再向父节点传递），点击 view2 会触发 handleTap2，点击 view1 会触发 handleTap1。

所有会发生冒泡的事件：

类型	触发条件
touchStart	触摸动作开始
touchMove	触摸后移动
touchEnd	触摸动作结束
touchCancel	触摸动作被打断，如来电提醒，弹窗
tap	触摸后马上离开
longTap	触摸后，超过500ms再离开

8.12.2 事件对象

组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。

BaseEvent 基础事件

BaseEvent 基础事件对象属性列表：

属性	类型	描述
type	String	事件类型
timeStamp	Integer	事件生成时的时间戳
target	Object	触发事件的组件的属性值集合

type

事件的类型。

timeStamp

事件生成时的时间戳。

target

触发事件的源组件对象，属性列表如下：

属性	类型	描述
id	String	事件源组件的 ID
tagName	String	当前组件的类型
dataset	Object	绑定事件的组件上，由 data- 开头的自定义属性的集合。
targetDataset	Object	实际触发事件的组件上，由 data- 开头的自定义属性的集合。

dataset 在组件中可以定义数据，这些数据将会通过事件传递给逻辑层。以 data- 开头，由连字符 - 连接多个单词，所有字母必须小写（大写字母自动转成小写字母），如 data-element-type，最终会在 event.target.dataset 中会将连字符转成驼峰 elementType。

代码示例：

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


Page({
bindViewTap:function(event) {
event.target.dataset.alphaBeta === 1; // - 会转为驼峰写法
event.target.dataset.alphabeta === 2; // 大写字母会转为小写字母
},
});
```

CustomEvent 自定义事件对象

CustomEvent 自定义事件对象（继承自 BaseEvent），属性列表如下：

属性	类型	描述

detail	Object	额外的信息

detail

自定义事件所携带的数据。表单组件事件会携带用户的输入信息，如 switch 组件 onChange 触发时可通过 event.detail.value 获取用户选择的状态值，媒体的错误事件会携带错误信息，更多信息请参见各组件文档事件说明。

TouchEvent 触摸事件对象

TouchEvent 触摸事件对象（继承自 BaseEvent），属性列表如下：

属性	类型	描述
touches	Array	当前停留在屏幕中的触摸点信息的数组
changedTouches	Array	当前变化的触摸点信息的数组

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

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

Touch 对象

属性	类型	描述
identifier	Number	触摸点的标识符
pageX, pageY	Number	距离文档左上角的距离，左上角为原点，横向为 X 轴，纵向为 Y 轴。
clientX, clientY	Number	距离页面可显示的区域（屏幕除去导航条）的距离，左上角为原点，横向为 X 轴，纵向为 Y 轴。

CanvasTouch 对象

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

8.13 自定义组件

8.13.1 自定义组件介绍

小程序基础库从 1.7.0 版本开始支持自定义组件功能。通过调用 my.canIUse('component') 可判断自定义组件功能是否可在当前版本使用。

自定义组件功能可将需要复用的功能模块抽象成自定义组件，从而在不同页面中复用。

说明：从小程序基础库 **1.14.0** 版本开始，自定义组件有了较大改动：

- 新增 onInit、deriveDataFromProps 生命周期函数 。
- 支持使用 ref 获取自定义组件实例 。

8.13.2 创建自定义组件

与 Page 类似，自定义组件也由 axml、js、json、acss 4 个部分组成。创建自定义组件有以下 2 个步骤：

1. 声明一个组件。
2. 使用 Component 函数，注册自定义组件。

以下为一个基本的组件示例：

```
// app.json
{
"component": true
}


// /components/customer/index.js
Component({
mixins: [], // minxin 方便复用代码
data: { x: 1 }, // 组件内部数据
props: { y: 1 }, // 可给外部传入的属性添加默认值
didMount(){}, // 生命周期函数
didUpdate(){},
didUnmount(){},
methods: { // 自定义方法
handleTap() {
this.setData({ x: this.data.x + 1}); // 可使用 setData 改变内部属性
},
},
})


<!-- /components/customer/index.axml -->
<view>
<view>x: {{x}}</view>
<button onTap="handleTap">plusOne</button>
<slot>
<view>default slot & default value</view>
</slot>
</view>
```

8.13.3 组件配置

在 [component].json 中声明自定义组件。如果该自定义组件还依赖了其它组件，则还需要额外声明依赖哪些自定义组件。

```
{
```

```
"component": true, // 必选，自定义组件的值必须是true
"usingComponents": {
"other":"../other/index"// 依赖的组件
}
}
```

参数详情：

参数	类型	是否必填	说明
component	Boolean	是	声明是自定义组件
usingComponents	Object	否	声明依赖的自定义组件所在路径：项目绝对路径以 / 开头，相对路径以 ./ 或者 ../ 开头

8.13.4 组件模板和样式

与页面类似，自定义组件可以有自己的 axml 模板和 acss 样式。

axml

axml 是自定义组件必选部分。

```
<!-- /components/index/index.axml -->
<view onTap="onMyClick"id="c-{{$id}}"/>


// /components/index/index.js
Component({
methods: {
onMyClick(e) {
console.log(this.is, this.$id);
},
},
});
```

说明：与页面不同，用户自定义事件需要放到 methods 里面。

插槽 slot

通过在组件 js 中支持 props，自定义组件可以和外部调用者交互，接受外部调用者传来的数据，同时可以调用外部调用者传来的函数，通知外部调用者组件内部的变化。

但是这样还不够，自定义组件还不够灵活。除了数据的处理与通知，小程序提供的 slot 使得自定义组件的 axml 结构可以使用外部调用者传来的 axml 组装。外部调用者可以传递 axml 给自定义组件，自定义组件使用其组装出最终的组件 axml 结构。

默认插槽 default slot

代码示例：

```
<!-- /components/index/index.axml -->
<view>
<slot>
<view>default slot & default value</view>
</slot>
<view>other</view>
</view>
```

调用者不传递 axml：

```
// /pages/index/index.json
{
"usingComponents": {
"my-component":"/components/index/index"
}
}

<!-- /pages/index/index.axml -->
<my-component />
```

页面输出：

```
default slot & default value
other
```

调用者传递 axml：

```
<!-- /pages/index/index.axml -->
<my-component>
<view>header</view>
<view>footer</view>
</my-component>
```

页面输出：

```
header
footer
other
```

可以将 slot 理解为插槽，default slot 就是默认插槽，如果调用者在组件标签 <xx> 之间不传递 axml，则渲染的是默认插槽。而如果调用者在组件标签 <xx> 之间传递有 axml，则使用其替代默认插槽，进而组装出最终的 axml 以供渲染。

具名插槽 named slot

default slot 只能传递一份 axml。复杂的组件需要在不同位置渲染不同的 axml，即需要传递多个 axml。此时需要 named slot。使用具名插槽后，外部调用者可以在自定义组件标签的子标签中指定要将哪一部分的 axml 放入到自定义组件的哪个具名插槽中。而自定义组件标签的子标签中没有指定具名插槽的部分则会放入到默认

插槽上。如果仅仅传递了具名插槽，则默认插槽不会被覆盖。

代码示例：

```
<!-- /components/index/index.axml -->
<view>
<slot>
<view>default slot & default value</view>
</slot>
<slot name="header"/>
<view>body</view>
<slot name="footer"/>
</view>
```

只传递具名插槽：

```
<!-- /pages/index/index.axml -->
<my-component>
<view slot="header">header</view>
<view slot="footer">footer</view>
</my-component>
```

页面输出：

```
default slot & default value
header
body
footer
```

传递具名插槽与默认插槽：

```
<!-- /pages/index/index.axml -->
<my-component>
<view>this is to default slot</view>
<view slot="header">header</view>
<view slot="footer">footer</view>
</my-component>
```

页面输出：

```
this is to default slot
header
body
footer
```

作用域插槽 slot-scope

通过使用 named slot ，自定义组件的 axml 要么使用自定义组件的 axml，要么使用外部调用者（比如页面）的 axml。使用自定义组件的 axml，可以访问组件内部的数据，同时通过 props 属性，可以访问外部调用者的数据。

代码示例：

```
// /components/index/index.js
Component({
data: {
x: 1,
},
props: {
y: '',
},
});


<!-- /components/index/index.axml -->
<view>component data: {{x}}</view>
<view>page data: {{y}}</view>


// /pages/index/index.js
Page({
data: { y: 2 },
});


<!-- /pages/index/index.axml -->
<my-component y="{{y}}"/>
```

页面输出：

```
component data: 1
page data: 2
```

自定义组件通过 slot 使用外部调用者（比如页面）的 axml 时，却只能访问外部调用者的数据。

代码示例：

```
<!-- /components/index/index.axml -->
<view>
<slot>
<view>default slot & default value</view>
</slot>
<view>body</view>
</view>


// /pages/index/index.js
Page({
data: { y: 2 },
});
```

```
<!-- /pages/index/index.axml -->
<my-component>
<view>page data: {{y}}</view>
</my-component>
```

页面输出：

```
page data: 2
body
```

slot scope 使得插槽内容可以访问到组件内部的数据。

代码示例：

```
// /components/index/index.js
Component({
data: {
x: 1,
},
});


<!-- /components/index/index.axml -->
<view>
<slot x="{{x}}">
<view>default slot & default value</view>
</slot>
<view>body</view>
</view>


// /pages/index/index.js
Page({
data: { y: 2 },
});


<!-- /pages/index/index.axml -->
<my-component>
<view slot-scope="props">
<view>component data: {{props.x}}</view>
<view>page data: {{y}}</view>
</view>
</my-component>
```

页面输出：

```
component data: 1
page data: 2
body
```

如上所示，自定义组件通过定义 slot 属性的方式暴露组件内部数据，页面使用组件时，通过 slot-scope 申明

为作用域插槽，属性值定义临时变量名 props，即可访问到组件内部数据。

acss

和页面一样，自定义组件也可以定义自己的 acss 样式。acss 会自动被引入使用组件的页面，不需要页面手动引入。详见 acss 语法 。

8.13.5 组件对象

Component 构造器

参数说明

参数	类型	是否必填	说明	最低版本
data	Object	否	组件内部状态	
props	Object	否	为外部传入的数据设置默认值	
onInit	Function	否	组件生命周期函数，组件创建时触发	1.14.0
deriveDataFromProps	Function	否	组件生命周期函数，组件创建时和更新前触发	1.14.0
didMount	Function	否	组件生命周期函数，组件创建完毕时触发	
didUpdate	Function	否	组件生命周期函数，组件更新完毕时触发	
didUnmount	Function	否	组件生命周期函数，组件删除时触发	
mixins	Array	否	组件间代码复用机制	
methods	Object	否	组件的方法，可以是事件响应函数或任意的自定义方法	

代码示例：

```
Component({
mixins:[{ didMount() {}, }],
data: {y:2},
props:{x:1},
didUpdate(prevProps,prevData){},
didUnmount(){},
methods:{
onMyClick(ev){
my.alert({});
this.props.onXX({ ...ev, e2:1});
},
},
})
```

说明：onInit、deriveDataFromProps 仅支持在基础库 1.14.0 版本及以上使用，可调用 my.canIUse('component2')实现兼容。

methods

自定义组件不仅可以渲染静态数据，也可以响应用户点击事件，进而处理并触发自定义组件重新渲染。methods 中可以定义任意自定义方法。

说明：与 Page 不同，自定义组件需要将事件处理函数定义在 methods 中。

```
// /components/counter/index.axml
<view>{{counter}}</view>
<button onTap="plusOne">+1</button>


// /components/counter/index.js
Component({
data: { counter: 0 },
methods: {
plusOne(e) {
console.log(e);
this.setData({ counter: this.data.counter + 1 });
},
},
});
```

页面会渲染一个按钮，每次点击它页面的数字都会加 1。

props

自定义组件可以接受外界的输入，做完处理之后，还可以通知外界说：我做完了。这些都可以通过 props 实现。

说明：

- props 为外部传过来的属性，可指定默认属性，不能在自定义组件内部代码中修改。
- 自定义组件的 axml 中可以直接引用 props 属性。
- 自定义组件的 axml 中的事件只能由自定义组件的 js 的 methods 中的方法来响应， 如果需要调用父组件传递过来的函数，可以在 methods 中通过 this.props 调用.

```
// /components/counter/index.js
Component({
data: { counter: 0 },
// 设置默认属性
props: {
onCounterPlusOne: (data) => console.log(data),
extra: 'default extra',
},
methods: {
plusOne(e) {
console.log(e);
const counter = this.data.counter + 1;
this.setData({ counter });
this.props.onCounterPlusOne(counter); // axml中的事件只能由methods中的方法响应
},
},
});
```

以上代码中 props 设置默认属性，然后在事件处理函数中通过 this.props 获取这些属性。

```
// /components/counter/index.axml
<view>{{counter}}</view>
<view>extra: {{extra}}</view>
<button onTap="plusOne">+1</button>


// /pages/index/index.json
{
"usingComponents": {
"my-component":"/components/counter/index"
}
}
```

外部使用不传递 props

```
// /pages/index/index.axml
<my-component />
```

页面输出：

```
0
extra: default extra
+1
```

此时并未传递参数，所以页面会显示组件 js 中 props 设定的默认值。

外部使用传递 props

说明：外部使用自定义组件时，如果传递参数是函数，一定要以 on 为前缀，否则会将其处理为字符串。

```
// /pages/index/index.js
Page({
onCounterPlusOne(data) {
console.log(data);
},
});


// /pages/index/index.axml
<my-component extra="external extra"onCounterPlusOne="onCounterPlusOne"/>
```

页面输出：

```
0
extra: external extra
+1
```

此时传递了参数，所以页面会显示外部传递的 extra 值 external extra 。

组件实例属性

属性名	类型	说明
data	Object	组件内部数据
props	Object	传入组件的属性
is	String	组件路径
$page	Object	组件所属页面实例
$id	Number	组件 ID，可直接在组件 axml 中渲染值

代码示例：

```
// /components/index/index.js
Component({
didMount(){
this.$page.xxCom = this; // 通过此操作可以将组件实例挂载到所属页面实例上
console.log(this.is);
console.log(this.$page);
console.log(this.$id);
}
});


<!-- /components/index/index.axml 组件 id 可直接在组件 axml 中渲染值 -->
<view>{{$id}}</view>


// /pages/index/index.json
{
"usingComponents": {
"my-component":"/components/index/index"
}
}


// /pages/index/index.js
Page({
onReady() {
console.log(this.xxCom); // 可以访问当前页面所挂载的组件
},
})
```

当组件在页面上渲染后，执行 didMount 回调，控制台输出如下：

```
/components/index/index
{$viewId: 51, route:"pages/index/index"}
1
```

组件实例方法

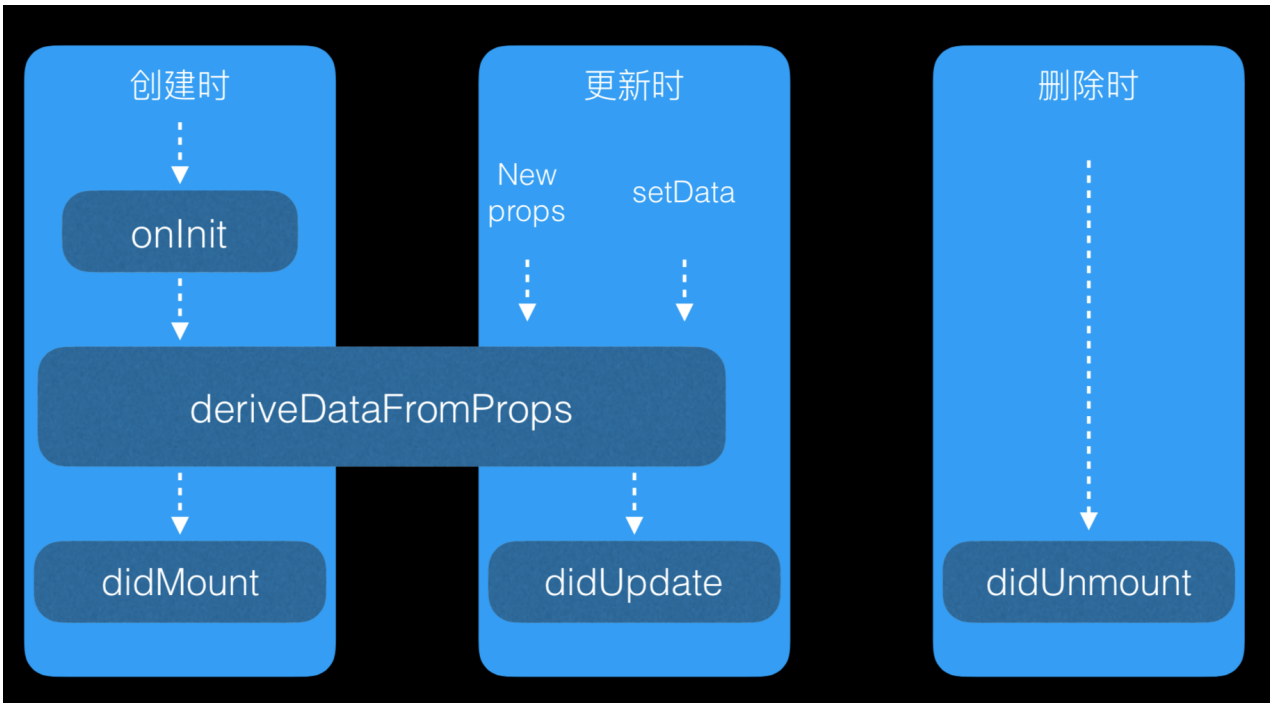
方法名	参数	说明	最低版本
setData	Object	设置data触发视图渲染	-

$spliceData	Object	设置data触发视图渲染	-

具体使用方式同 页面 。

8.13.6 生命周期

组件的生命周期函数在特殊的时间点由框架触发。组件生命周期示意图如下：



生命周期函数具体信息见下表：

生命周期	参数	说明	最低版本
onInit	无	组件创建时触发	1.14.0
deriveDataFromProps	nextProps	组件创建时和更新前触发	1.14.0
didMount	无	组件创建完毕时触发	-
didUpdate	(prevProps,prevData)	组件更新完毕时触发	-
didUnmount	无	组件删除时触发	-

说明：onInit、deriveDataFromProps 自基础库 1.14.0 才支持，可以使用 my.canIUse('component2') 做兼容。

onInit

onInit在组件创建时触发。在onInit中，可以：

- 访问 this.is、this.$id、this.$page 等属性。
- 访问 this.data、this.props 等属性。
- 访问组件 methods 中的自定义属性。
- 调用 this.setData、this.$spliceData 修改数据。

代码示例一

```
// /components/counter/index.js
Component({
data: {
counter: 0,
},
onInit() {
this.setData({
counter: 1,
is: this.is,
});
},
})


<!-- /components/counter/index.axml -->
<view>{{counter}}</view>
<view>{{is}}</view>
```

当组件在页面上渲染后，页面输出如下：

```
1
/components/counter/index
```

代码示例二

```
// /components/counter/index.js
Component({
onInit() {
this.xxx = 2;
this.data = { counter: 0 };
},
})


<!-- /components/counter/index.axml -->
<view>{{counter}}</view>
```

当组件在页面上渲染后，页面输出如下：

```
0
```

deriveDataFromProps

deriveDataFromProps 在组件创建和更新时都会触发。在deriveDataFromProps 中可以:

- 访问 this.is、this.$id、this.$page 等属性。
- 访问this.data、this.props 等属性。

- 访问组件 methods 中的自定义属性。
- 调用this.setData、 this.$spliceData 修改数据。
- 使用 nextProps 参数获取将要更新的 props 参数。

代码示例

```
// /components/counter/index.js
Component({
data: {
counter: 5,
},
deriveDataFromProps(nextProps) {
if (this.data.counter < nextProps.pCounter) {
this.setData({
counter: nextProps.pCounter,
});
}
},
})


<!-- /components/counter/index.axml -->
<view>{{counter}}</view>


// /pages/index/index.js
Page({
data: {
counter: 1,
},
plus() {
this.setData({ counter: this.data.counter + 1 })
},
})


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

说明：在本示例中，点击 + 按钮，页面上的 counter 会一直保持不变，直到 pCounter 的值大于 5。

didMount

didMount 为自定义组件首次渲染完毕后的回调，此时页面已经渲染，通常在这时请求服务端数据。

代码示例

```
Component({
data: {},
didMount() {
```

```
let that = this;
my.httpRequest({
url: 'http://httpbin.org/post',
success: function(res) {
console.log(res);
that.setData({name: 'xiaoming'});
}
});
},
});
```

didUpdate

didUpdate 为自定义组件数据更新后的回调，每次组件数据变更的时候都会调用。

代码示例

```
Component({
data: {},
didUpdate(prevProps, prevData) {
console.log(prevProps, this.props, prevData, this.data);
},
});
```

说明：

- 组件内部调用 this.setData 会触发 didUpdate。
- 外部调用者调用 this.setData 也会触发 didUpdate。

didUnmount

didUnmount 为自定义组件被卸载后的回调，每当组件实例从页面卸载的时候都会触发此回调。

代码示例

```
Component({
data: {},
didUnmount() {
console.log(this);
},
});
```

8.13.7 mixins

开发者有时可能会实现多个自定义组件，而这些自定义组件可能会有些公共逻辑要处理，小程序提供 mixins 用于解决这种情况。

以下为示例：

```
// /minxins/lifecylce.js
```

```
export default {
onInit(){},
deriveDataFromProps(nextProps){},
didMount(){},
didUpdate(prevProps,prevData){},
didUnmount(){},
};
```

说明：onInit 与 deriveDataFromProps 自基础库 1.14.0 开始支持，可以使用 my.canIUse('component2') 做兼容判断。

```
// /pages/index/index.js
import lifecylce from './minxins/lifecylce';

const initialState = {
data: {
isLogin: false,
},
};

const defaultProps = {
props: {
age: 30,
},
};

const methods = {
methods: {
onTapHandler() {},
},
}

Component({
mixins: [
lifecylce,
initialState,
defaultProps,
methods
],
data: {
name: 'alipay',
},
});
```

8.13.8 ref 获取组件实例

从 1.14.0 版本开始，自定义组件支持使用 ref 获取自定义组件实例，可以使用 my.canIUse('component2')做兼容.

```
// /pages/index/index.js
Page({
plus() {
this.counter.plus();
},
// saveRef 方法的参数 ref 为自定义组件实例，运行时由框架传递给 saveRef
```

```
saveRef(ref) {
// 存储自定义组件实例，方便以后调用
this.counter = ref;
},
})



<!-- /pages/index/index.axml -->
<counter ref="saveRef"/>
<button onTap="plus">+</button>
```

说明：

- 使用ref 绑定 saveRef 之后，会在组件初始化时触发 saveRef 方法。
- saveRef 方法的参数 ref 为自定义组件实例，由框架传递给saveRef方法。
- ref 同样可以用于父组件获取子组件的实例。

```
// /components/counter/index.js
Component({
data: {
counter: 0,
},
methods: {
plus() {
this.setData({ counter: this.data.counter + 1 })
},
},
})



<!-- /components/counter/index.axml -->
<view>{{counter}}</view>
```

8.13.9 使用自定义组件

说明：自定义组件的事件（如 onTap 等），并不是每个自定义组件默认支持的，需要自定义组件本身明确支持才能使用。自定义组件支持事件具体方法请参见 component 构造器 。

使用方法

自定义组件的使用和基础组件类似。

在页面 JSON 文件中指定使用的自定义组件。

```
// /pages/index/index.json
{
"usingComponents": {
"customer":"/components/customer/index"
}
}
```

在页面的 AXML 文件中使用自定义组件，与使用基础组件类似。

```
<!-- /pages/index/index.axml -->
<view>
<!-- 给自定义组件传递 属性name与属性age -->
<customer name="tom"age="{{23}}"/>
</view>
```

说明：

- 使用自定义组件时，给自定义组件传递的属性可以在自定义组件内通过 this.props 获取，参见 props 。
- 自定义组件只能在 page 自身的 AXML 文件和组件自身的 AXML 文件中使用，不能通过 import 或 include 使用。

正确示例：

```
<!-- /pages/index/index.axml -->
<my-com />
```

错误示例：

```
<!-- /pages/index/index.axml -->
<include src="./template.axml"/>

<!-- /pages/index/template.axml -->
<view>
<my-com />
</view>
```

引用自定义组件

```
// 在 /pages/index/index.json 中配置（不是 app.json ）
{
"usingComponents":{
"your-custom-component":"mini-antui/es/list/index",
"your-custom-component2":"/components/card/index",
"your-custom-component3":"./result/index",
"your-custom-component4":"../result/index"
}
}

// 项目绝对路径以 / 开头，相对路径以 ./ 或者 ../ 开头
```

8.13.10 发布自定义组件

mPaaS 小程序原生支持引入第三方 npm 模块。因此，自定义组件也支持发布到 npm，方便开发者复用和分享。

推荐用于发布的自定义组件目录

以下目录结构，仅供参考。

文件结构

```
├── src // 用于单个自定义组件
│ ├── index.js
│ ├── index.json
│ ├── index.axml
│ ├── index.acss
│ └── demo // 用于自定义组件的 demo 演示
│ ├── index.js
│ ├── index.json
│ ├── index.axml
│ └── index.acss
├── app.js // 用于上方的自定义组件小程序 demo
├── app.json
└── app.acss
```

JSON 示例

```
// package.json
{
"name":"your-custom-compnent",
"version":"1.0.0",
"description":"your-custom-compnent",
"repository": {
"type":"git",
"url":"your-custom-compnent-repository-url"
},
"files": [
"es"
],
"keywords": [
"custom-component",
"mini-program"
],
"devDependencies": {
"rc-tools":"6.x"
},
"scripts": {
"build":"rc-tools run compile && node scripts/cp.js && node scripts/rm.js",
"pub":"git push origin && npm run build && npm publish"
}
}
```

js 文件示例

```
// scripts/cp.js
const fs = require('fs-extra');
const path = require('path');
// copy file
fs.copySync(path.join(__dirname, '../src'), path.join(__dirname, '../es'), {
```

```
filter(src, des){
return !src.endsWith('.js');
}
});



// scripts/rm.js
const fs = require('fs-extra');
const path = require('path');

// remove unnecessary file
const dirs = fs.readdirSync(path.join(__dirname, '../es'));

dirs.forEach((item) => {
if (item.includes('app.') || item.includes('DS_Store') || item.includes('demo')) {
fs.removeSync(path.join(__dirname, '../es/', item));
} else {
const moduleDirs = fs.readdirSync(path.join(__dirname, '../es/', item));
moduleDirs.forEach((item2) => {
if (item2.includes('demo')) {
fs.removeSync(path.join(__dirname, '../es/', item, item2));
}
});
}
});

fs.removeSync(path.join(__dirname, '../lib/'));
```

8.14 性能优化建议

运行原理

与传统的 H5 应用不同，小程序运行架构分为 webview 和 worker 两个部分。webview 负责渲染，worker 则负责存储数据和执行业务逻辑。

1. webview 和 worker 之间的通信是异步的。这意味着当我们调用 setData 时，我们的数据并不会立即渲染，而是需要从 worker 异步传输到 webview。
2. 数据传输时需要序列化为字符串，然后通过 evaluateJavascript 方式传输，数据大小会影响性能。

优化首屏

首屏有多种定义，这里的首屏是指业务角度第一次有意义的渲染。比如: 对于列表页，首屏就是列表第一次渲染出的内容。

控制小程序资源包大小

当用户访问一个小程序时，支付宝客户端会首先从 CDN 下载小程序资源包，所以资源包的大小会影响小程序启动性能。

优化建议：

- 及时删除无用图片资源，因为所有图片资源都会默认打包进去。

- 控制图片大小，避免使用大图，大图建议从 CDN 渠道上传。
- 及时清理无用代码。

将数据请求提前至 onLoad

- 小程序运行时，先触发页面的 onLoad 生命周期函数，再将页面初始数据（Page data）从 worker 传递到 webview 进行一次初始渲染。
- 页面初始渲染完成，从 webview 发出通知到 worker，触发 onReady 生命周期函数。

部分小程序会在 onReady 中发出请求，导致首屏渲染延缓。

优化建议：将数据请求提前到 onLoad 中。

控制首屏一次性渲染节点数量

业务请求返回后，通常会调用 setData 触发页面重新渲染。执行过程如下:

1. 数据从 worker 传递到 webview
2. webview 上根据传过来的数据构造虚拟 DOM，并与之前做差异比较（从根节点开始），然后渲染。

由于 worker 与 webview 通信时，数据需要序列化，然后到了 webview 需要执行 evaluateJavascript，因此如果一次性传输数据太大，会影响首屏渲染性能。

另外，如果 webview 上构造节点过多，层级嵌套太深（例如有的小程序列表页面一次性渲染超过 100 个列表项，每个列表项又有嵌套内容，而实际上整个屏幕可能只是显示不到 10 个），会导致差异比较时间较长，同时由于是首屏渲染，会一次性构造很多 DOM，影响首屏渲染性能。

优化建议：

- setData 数据量不宜过大，避免一次性传递过长的列表。
- 首屏请勿一次性构造太多节点，服务端可能一次请求传递大量数据，请勿一次性 setData，可先 setData 一部分数据，然后等待一段时间（比如 400 ms，具体需要业务调节）再调用 $spliceData 将其他数据传输过去。

优化 setData 逻辑

任何页面变化都会触发 setData，同一时间可能会有多个 setData 触发页面进行重新渲染。如下四个接口都会触发 webview 页面重新渲染。

- Page.prototype.setData: 触发整个页面做差异比较
- Page.prototype.$spliceData: 针对长列表做优化，避免每次传递整个列表，触发整个页面做差异比较
- Component.prototype.setData: 只会从对应组件节点开始做差异比较
- Component.prototype.$spliceData: 针对长列表做优化，避免每次传递整个列表，只会从对应组件节点开始做差异比较

优化建议：

- 避免频繁触发 setData 或者 $spliceData，不管是页面级别还是组件级别。在我们分析的案例中，有些页面有倒计时逻辑，但是有的倒计时过于频繁触发（ms 级别的触发）。
- 需要频繁触发重新渲染时，避免使用页面级别的 setData 和 $spliceData， 将这一块封装成自定义组件，然后使用组件级别的 setData 或 $spliceData 触发组件重新渲染。
- 长列表数据触发渲染时，使用 $spliceData 多次追加数据，而不用传递整个列表。
- 复杂页面建议封装成自定义组件，减少页面级别的 setData。

优化案例：

推荐指定路径设置数据：

```
this.setData({
'array[0]': 1,
'obj.x':2,
});
```

不推荐 如下用法（虽然拷贝了 this.data, 仍然直接更改了其属性）：

```
const array = this.data.array.concat();
array[0] = 1;
const obj={...this.data.obj};
obj.x=2;
this.setData({array,obj});
```

更不推荐 直接更改 this.data（违反不可变数据原则）：

```
this.data.array[0]=1;
this.data.obj.x=2;
this.setData(this.data)
```

长列表使用 $spliceData：

```
this.$spliceData({ 'a.b': [1, 0, 5, 6] })
```

说明：有时业务逻辑封装到了组件中，当组件 UI 需要重新渲染时，只需在组件内部调用 setData。但有时需要从页面触发组件重新渲染，比如在页面上监听了 onPageScroll 事件，当事件触发时，需要通知对应组件重新渲染，此时的处理措施如下所示：

```
// /pages/index/index.js
Page({
onPageScroll(e) {
if (this.xxcomponent) {
this.xxcomponent.setData({
scrollTop: e.scrollTop
})
}
}
```

```
})


// /components/index/index.js
Component({
didMount(){
this.$page.xxcomponent = this;
}
})
```

可在组件的 didMount 中将组件挂载到对应的页面上，即可在页面中调用组件级别的 setData 只触发组件重新渲染。

使用 key 参数

在 for 中使用 key 来提高性能。

重要：key 不能设置在 block 上。
示例代码：

```
<view a:for="{{array}}"key="{{item.id}}"></view>
<block a:for="{{array}}"><view key="{{item.id}}"></view></block>
```

9 组件

9.1 组件概述

基础组件

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。

组件共有属性

所有的组件包含以下属性：

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 。

组件属性类型

每个组件提供了一系列的属性配置，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	-

组件数据绑定

通过 {{}} 才能传入指定的属性类型数据，参见 数据绑定 。

基础组件总览

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
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	单选项目
checkbox	多项选择器组
switch	单选项目
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

画布

名称	功能说明
canvas	画布

地图

名称	功能说明
map	地图组件

开放组件

名称	功能说明
web-view	承载H5网页的组件
关注生活号	关注生活号组件
智能客服	智能客服组件
小程序收藏	小程序收藏组件

9.2 组件常见问题

键盘与组件交互异常

对于需要启动 **键盘** 的组件（如 input、textarea 等），目前默认使用的是原生键盘。如果键盘和组件的交互存在异常，可在组件中添加 enableNative="{{false}}" 属性，如：

```
<textarea value="{{inputValue}}"enableNative="{{false}}"maxlength="500"onInput="onInput"/>
```

即可恢复到使用 WKWebview 的键盘。

9.3 视图容器

9.3.1 view

视图容器，相当于 Web 的 div 标签或者 React Native 的 View 组件。

属性名	类型	默认值	描述	最低版本
disable-scroll	Boolean	false	是否阻止区域内滚动页面	
hover-class	String		点击时添加的样式类	
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			用于动画，详见 my.createAnimation	
hover-stop-propagation	Boolean	false	是否阻止当前元素的祖先元素出现点击态	1.10.0
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		过渡结束时触发	1.8.0
onAnimationIteration	EventHandle		每开启一次新的动画过程时触发。（第一次不触发）	1.8.0
onAnimationEnd	EventHandle		动画结束时触发	1.8.0
onAppear	EventHandle		当前元素可见时触发。	1.9.0
onDisappear	EventHan		当前元素从可见变为不可见时触发。	1.9.0

	dle			
onFirstAppear	EventHandle		当前元素首次可见时触发。	1.9.4

说明：使用 my.createAnimation 生成的动画是通过 过渡 实现的，只会触发 onTransitionEnd；不会触发 onAnimationStart、onAnimationIteration、onAnimationEnd。

代码示例

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

9.3.2 swiper

滑块视图容器。

属性名	类型	默认值	描述	最低版本
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	
active-class	String		swiper-item 可见时的 class	1.13.7
changing-class	String		acceleration 设置为 {{true}} 时且处于滑动过程中，中间若干屏处于可见时的 class	1.13.7
autoplay	Boolean	false	是否自动切换	
current	Number	0	当前页面的 index	
duration	Number	500(ms)	滑动动画时长	
interval	Number	5000(ms)	自动切换时间间隔	
circular	Boolean	false	是否启用无限滑动	
vertical	Boolean	false	滑动方向是否为纵向	
previous-margin	String	‘0px	前边距，单位 px，1.9.0 暂时只支持水平方向	1.9.0

		，		
next-margin	String	，‘0px	后边距，单位 px，1.9.0 暂时只支持水平方向	1.9.0
acceleration	Boolean	false	当开启时，会根据滑动速度，连续滑动多屏	1.13.7
disable-programmatic-animation	Boolean	false	是否禁用代码变动触发 siwper 切换时使用动画。	1.13.7
onChange	EventHandle		current 改变时会触发，event.detail = {current, isChanging}，其中 isChanging 需 acceleration 设置为 {{true}} 时才有值，当连续滑动多屏时，中间若干屏触发 onChange 事件时 isChanging 为 true，最后一屏返回 false。	
onTransition	EventHandle		swiper 中 swiper-item 的位置发生改变时会触发 transition 事件。	1.15.0
onAnimationEnd	EventHandle		动画结束时会触发 animationEnd 事件，event.detail = {current, source}，其中source的值有 autoplay 和 touch	1.15.0
disable-touch	Boolean	false	是否禁用用户 touch 操作	1.15.0

swiper-item

仅可放置在组件中，宽高自动设置为 100%。

图示

代码示例

```
<swiper
indicator-dots="{{indicatorDots}}"
autoplay="{{autoplay}}"
interval="{{interval}}"
>
```

```
<block a:for="{{background}}">
<swiper-item>
<view class="swiper-item bc_{{item}}"></view>
</swiper-item>
</block>
</swiper>
<view class="btn-area">
<button class="btn-area-button"type="default"onTap="changeIndicatorDots">indicator-dots</button>
<button class="btn-area-button"type="default"onTap="changeAutoplay">autoplay</button>
</view>
<slider onChange="intervalChange"value="{{interval}}"show-value min="2000"max="10000"/>
<view class="section__title">interval</view>


Page({
data: {
background: ['green', 'red', 'yellow'],
indicatorDots: true,
autoplay: false,
interval: 3000,
},
changeIndicatorDots(e) {
this.setData({
indicatorDots: !this.data.indicatorDots
})
},
changeAutoplay(e) {
this.setData({
autoplay: !this.data.autoplay
})
},
intervalChange(e) {
this.setData({
interval: e.detail.value
})
},
})
```

9.3.3 scroll-view

可滚动视图区域。

属性名	类型	默认值	描述	最低版本
class	String		外部样式名	
style	String		内联样式名	
scroll-x	Boolean	false	允许横向滚动	
scroll-y	Boolean	false	允许纵向滚动	
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。	

scroll-top	Number		设置竖向滚动条位置	
scroll-left	Number		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	
scroll-animation-duration	Number		当 scroll-with-animation 设置为 true 时，可以设置 scroll-animation-duration 来控制动画的执行时间，单位 ms。	1.9.0
enable-back-to-top	Boolean	false	当点击 iOS 顶部状态栏或者双击安卓标题栏时，滚动条返回顶部，只支持竖向。	1.11.0
trap-scroll	Boolean	false	纵向滚动时，当滚动到顶部或底部时，强制禁止触发页面滚动，仍然只触发 scroll-view 自身的滚动。	1.11.2
onScrollToUpper	EventHandle		滚动到顶部/左边，会触发 scrolltoupper 事件。	
onScrollToLower	EventHandle		滚动到底部/右边，会触发 scrolltolower 事件。	
onScroll	EventHandle		滚动时触发，event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth}	
onTouchStart	EventHandle		触摸动作开始	1.15.0
onTouchMove	EventHandle		触摸后移动	1.15.0
onTouchEnd	EventHandle		触摸动作结束	1.15.0
onTouchCancel	EventHandle		触摸动作被打断，如来电提醒、弹窗	1.15.0

说明：使用竖向滚动时，需要给出固定高度，通过 acss 设置 height。

代码示例

```
<view class="page">
<view class="page-description">可滚动视图区域</view>
<view class="page-section">
<view class="page-section-title">vertical scroll</view>
<view class="page-section-demo">
<scroll-view scroll-y="{{true}}"style="height:
200px;"onScrollToUpper="upper"onScrollToLower="lower"onScroll="scroll"scroll-into-view="{{toView}}"scroll-
top="{{scrollTop}}">
<view id="blue"class="scroll-view-item bc_blue"></view>
<view id="red"class="scroll-view-item bc_red"></view>
<view id="yellow"class="scroll-view-item bc_yellow"></view>
<view id="green"class="scroll-view-item bc_green"></view>
</scroll-view>
</view>
<view class="page-section-btns">
<view onTap="tap">next</view>
<view onTap="tapMove">move</view>
<view onTap="scrollToTop">scrollToTop</view>
</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 bc_blue"></view>
<view id="red2"class="scroll-view-item_H bc_red"></view>
<view id="yellow2"class="scroll-view-item_H bc_yellow"></view>
<view id="green2"class="scroll-view-item_H bc_green"></view>
</scroll-view>
</view>
</view>
</view>


const order = ['blue', 'red', 'green', 'yellow'];

Page({
data: {
toView: 'red',
scrollTop: 100,
},
upper(e) {
console.log(e);
},
lower(e) {
console.log(e);
},
scroll(e) {
console.log(e.detail.scrollTop);
},
scrollToTop(e) {
console.log(e);
this.setData({
scrollTop: 0,
});
},
});
```

提示

- scroll-into-view 的优先级高于 scroll-top。
- 在滚动 scroll-view 时会阻止页面回弹，所以在 scroll-view 中滚动时，无法触发 onPullDownRefresh。

9.3.4 CoverView

cover-view

覆盖在原生组件之上的文本视图。可覆盖 map 原生组件，小程序基础库 1.10.0 及以上版本开始支持嵌套。

属性名	类型	默认值	描述	最低版本
onTap	EventHandle	-	点击事件回调	1.9.0

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，小程序基础库 1.10.0 及以上版本开始支持嵌套。

属性名	类型	默认值	描述	最低版本
src	String	-	图片地址，支持的地址格式同 image 一致	1.9.0
onTap	EventHandle	-	点击事件回调	1.9.0

代码示例

```
<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>
```

9.3.5 movable-view

基础库 1.11.0 开始支持，低版本需做兼容处理，操作参见 小程序基础库说明 。

movable-area

movable-view 的可移动区域。

> 注意：movable-area 必须设置width和height属性，不设置默认为10px

movable-view

可移动的视图容器，在页面中可以拖拽滑动

属性名	类型	默认值	描述	最低版本
direction	String	none	movable-view的移动方向，属性值有 all、vertical、horizontal、none。	-
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。	-
disabled	Boolean	false	是否禁用	-
onTouchStart	EventHandle		触摸动作开始	1.11.5
onTouchMove	EventHandle		触摸后移动	1.11.5
onTouchEnd	EventHandle		触摸动作结束	1.11.5
onTouchCancel	EventHandle		触摸动作被打断，如来电提醒、弹窗	1.11.5
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}	-

示例代码

```
<movable-area style="width: 100px;height: 100px;background-color: red;margin-left: 100px;">
<movable-view
onChange="onMovableViewChange"
onChangeEnd="onMovableViewChangeEnd"
direction="vertical"
x="{{10}}"
y="{{10}}"
style="width: 40px;height: 40px;background-color: rgba(0, 0, 0, 0.5);"
>
<view onTap="onTapMovableView">movable-view</view>
</movable-view>
</movable-area>
```

提示

- movable-view 必须设置 width 和 height 属性，不设置默认为 10 px。
- movable-view 默认为绝对定位（请勿修改），top 和 left 属性为 0 px。
- 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）。
- movable-view 必须在 <movable-area/> 组件中，并且必须是直接子节点，否则不能移动。

9.4 基础内容

9.4.1 text

文本，组件内只支持嵌套。

属性名	类型	默认值	描述	最低版本
selectable	Boolean	false	是否可选	-
space	String		显示连续空格	-
decode	Boolean	false	是否解码	-
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

代码示例

```
<view class="page">
<view class="text-view">
<text>{{text}}</text>
</view>
</view>


Page({
data: {
text: `移动开发平台（Mobile PaaS，简称 mPaaS）是源于支付宝 App 的移动开发平台`,
},
})
```

9.4.2 icon

图标。

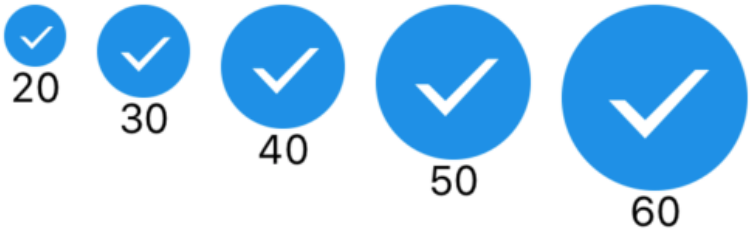
属性名	类型	默认值	描述	最低版本
type	String		icon 类型，有效值：info, warn、waiting、cancel、download、search、clear、success、success_no_circle、loading。	loading (1.7.2)
size	Number	23	icon 大小，单位 px	
color	Color		icon 颜色，同 CSS 的 color	

图示

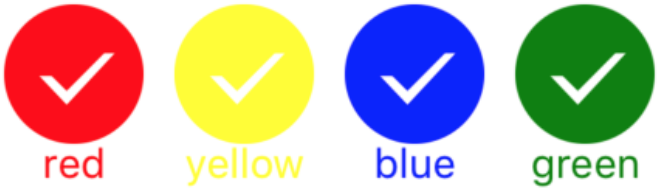
图标

Type

Size



Color



代码示例

```
<block a:for="{{iconType}}">
<view class="item">
<icon type="{{item}}"aria-label="{{item}}"size="45"/>
<text>{{item}}</text>
</view>
</block>

<block a:for="{{iconSize}}">
<view class="item">
<icon type="success"size="{{item}}"/>
<text>{{item}}</text>
</view>
</block>

<block a:for="{{iconColor}}">
<view class="item">
<icon type="success"size="45"color="{{item}}"/>
<text style="color:{{item}}">{{item}}</text>
</view>
</block>


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',
]
}
})
```

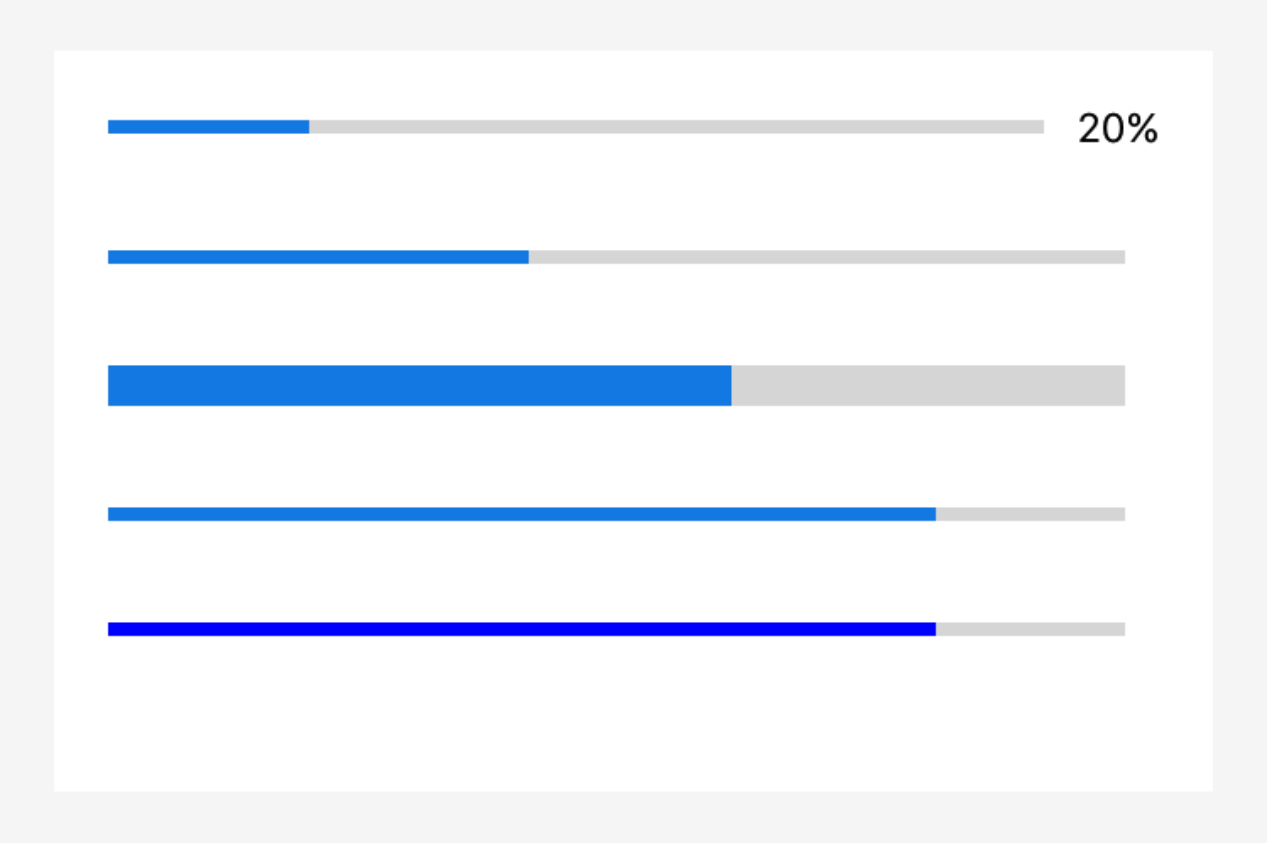
9.4.3 progress

进度条。

属性名	类型	默认值	描述	最低版本
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	从左往右是否进行加载动画	

图示



代码示例

```
<progress percent="20"show-info/>
<progress percent="40"active/>
<progress percent="60"stroke-width="10"/>
<progress percent="80"active/>
<progress percent="80"color="#10AEFF"/>
```

9.4.4 rich-text

rich-text 是一个富文本组件。基础库 1.11.0 开始支持，低版本需做兼容处理，操作参见 小程序基础库说明 。

属性名	类型	默认值	描述	最低版本
nodes	Array	[]	只支持 **节点列表**	-

默认支持如下事件：

- tap
- touchstart

- touchmove
- touchcancel
- touchend
- longtap

说明：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。

nodes

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

元素节点

属性名	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

文本节点

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

支持的HTML节点及属性

支持 class 和 style 属性，不支持 ID 属性。

节点	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	-

代码示例

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

```
// page.js
Page({
data: {
nodes: [{
name: 'div',
attrs: {
class: 'test_div_class',
style: 'color: green;'
},
children: [{
type: 'text',
text: 'Hello World! This is a text node.'
}]
}]
},
tap() {
console.log('tap')
}
})
```

注：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	<
>	大于号	>
&	和号	&
“	引号	"
‘	撇号	'

9.5 表单组件

9.5.1 button

本文介绍按钮（button）。

属性名	类型	默认值	描述	最低版本
size	String	default	有效值为 default、mini	-
type	String	default	按钮的样式类型，有效值为 primary、default、warn	-
plain	Boolean	false	是否镂空	-
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。hover-class="none" 时表示没有点击态效果	-
hover-start-time	Number	20	按住后多少事件后出现点击状态，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点击态	1.10.0
form-type	String	-	有效值为 submit、reset，用于组件，点击分别会触发 submit/reset 事件	-
open-type	String	-	开放能力	1.1.0
scope	String	-	当 open-type 为 getAuthorize 时有效	1.11.0
onTap	EventHandle	-	点击	-
app-parameter	String	-	打开 APP 时，向 APP 传递的参数，open-type="launchApp" 时有效	-

说明：button-hover 默认为 {background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;}。

open-type 有效值

值	说明	最低版本
share	触发自定义分享，可使用 canIUse('button.open-type.share') 判断	1.1.0
getAuthorize	支持小程序授权，可使用 canIUse('button.open-type.getAuthorize') 判断	1.11.0
contactShare	分享到通讯录好友，可使用 canIUse('button.open-type.contactShare') 判断	1.11.0

scope 有效值

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

值	说明	最低版本
phoneNumber	唤起授权界面，用户可以授权小程序获取用户手机号	1.11.0

图示

按钮

type-primary/ghost

type-default

代码示例

```
<view class="page">
<view class="section">
<view class="title">Type</view>
<button type="default">default</button>
<button type="primary">primary</button>
<button type="warn">warn</button>
</view>
<view class="section"style="background:#ddd;">
<view class="title">Misc</view>
<button type="default"plain>plain</button>
<button type="default"disabled>disabled</button>
<button type="default"loading={{true}}>loading</button>
<button type="default"hover-class="red">hover-red</button>
</view>
<view class="section">
<view class="title">Size</view>
<button type="default"size="mini">mini</button>
</view>
<view class="section">
<view class="title">Type</view>
<form onSubmit="onSubmit"onReset="onReset">
<button form-type="submit">submit</button>
<button form-type="reset">reset</button>
</form>
</view>
</view>
```

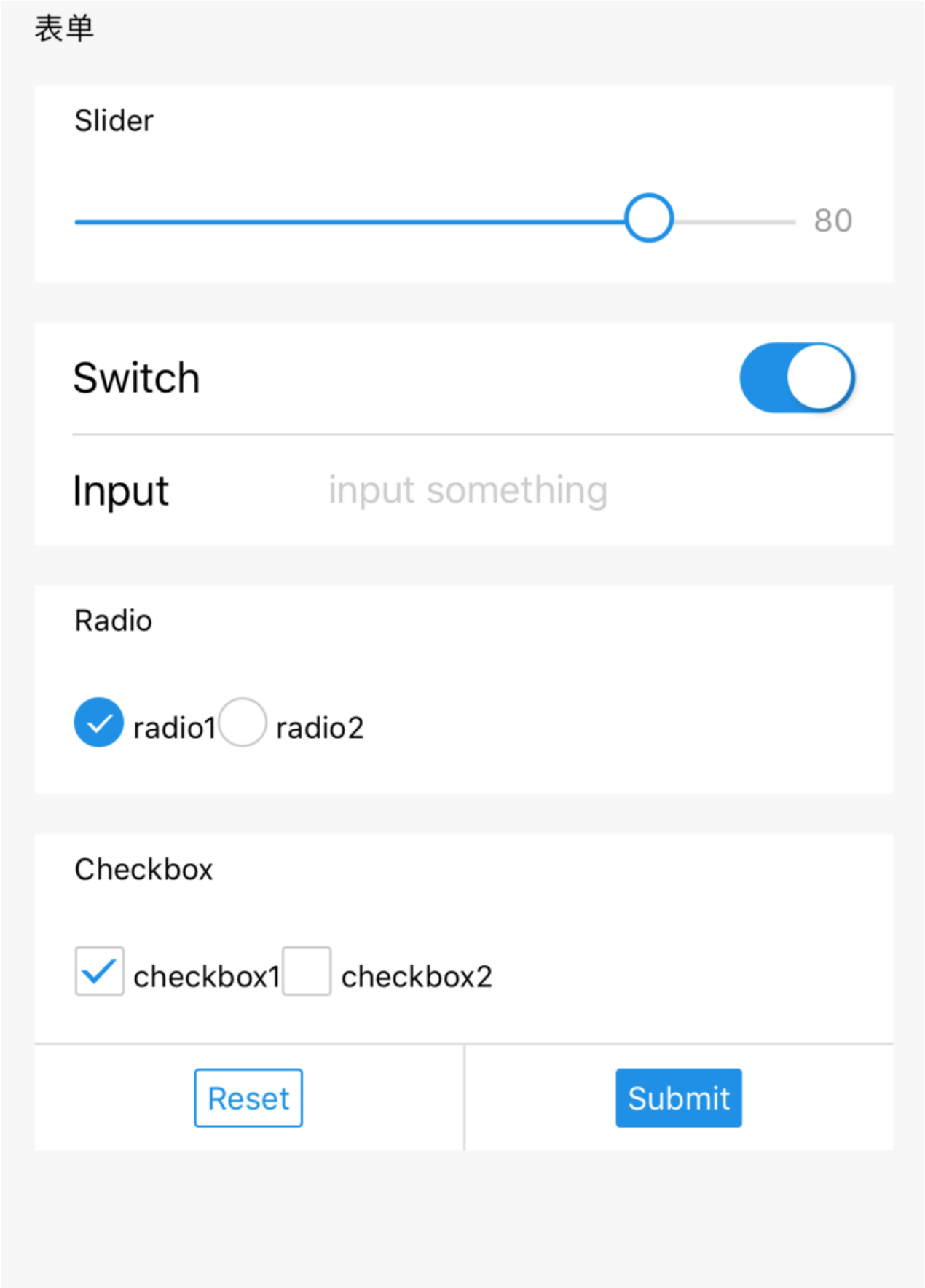
9.5.2 form

表单（form），用于将组件内的用户输入的 <textarea>、<switch/>、<input/>、<checkbox-group/>、<slider/>、<radio-group/>、<picker/> 等组件提交。

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

属性名	类型	默认值	描述	最低版本
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	1.3.0
onSubmit	EventHandle		携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'dao14'}, buttonTarget: {'dataset': 'buttonDataset'} }	buttonTarget 1.7.0 开始支持
onReset	EventHandle		表单重置时会触发 reset 事件	-

图示


表单
Slider
80
Switch
Input
input something
Radio
radio1
radio2
Checkbox
checkbox1
checkbox2
Reset
Submit

代码示例

```
<form onSubmit="formSubmit"onReset="formReset"report-submit="true">
<view class="section section_gap">
<view class="section__title">switch</view>
<switch name="switch"/>
</view>
<view class="section section_gap">
<view class="section__title">slider</view>
<slider name="slider"show-value ></slider>
</view>

<view class="section">
<view class="section__title">input</view>
<input name="input"placeholder="please input here"/>
</view>
<view class="section section_gap">
<view class="section__title">radio</view>
<radio-group name="radio-group">
<label><radio value="radio1"/>radio1</label>
<label><radio value="radio2"/>radio2</label>
</radio-group>
</view>
<view class="section section_gap">
<view class="section__title">checkbox</view>
<checkbox-group name="checkbox">
<label><checkbox value="checkbox1"/>checkbox1</label>
<label><checkbox value="checkbox2"/>checkbox2</label>
</checkbox-group>
</view>
<view class="btn-area">
<button formType="submit">Submit</button>
<button formType="reset">Reset</button>
</view>
</form>



Page({
formSubmit: function(e) {
console.log('form发生了submit事件，携带数据为：', e.detail.value)
},
formReset: function() {
console.log('form发生了reset事件')
}
})
```

9.5.3 label

标签（label）可以用来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。

目前可以绑定的控件有：<checkbox/>、<radio/>、<input/>、<textarea/>。

属性名	类型	描述	最低版本
for	String	绑定组件的 ID	

图示

代码示例

```
<view class="section">
<view class="title">Checkbox，label 套 checkbox</view>
<checkbox-group>
<view>
<label>
<checkbox value="aaa"/>
```

```
<text>aaa</text>
</label>
</view>
<view>
<label>
<checkbox value="bbb"/>
<text>bbb</text>
</label>
</view>
</checkbox-group>
</view>
</view>

<view class="section">
<view class="title">Radio，通过 for 属性关联</view>
<radio-group>
<view>
<radio id="aaa"value="aaa"/>
<label for="aaa">aaa</label>
</view>
<view>
<radio id="bbb"value="bbb"/>
<label for="bbb">bbb</label>
</view>
</radio-group>
</view>
</view>

<view class="section">
<view class="title">多个 Checkbox 点击后只选中一个</view>
<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>
```

9.5.4 input

本文介绍输入框（input）。

属性名	类型		描述	最低版本	
value	String		初始内容	-	
name	String		组件名字，用于表单提交获取数据	-	

type	String		input 的类型，有效值：text、 number、 idcard、 digit、numberpad、digitpad、idcardpad。	numberpad、digitpad、idcardpad 1.13.0，类型客户端 10.1.50 开始支持，可通过 my.canIUse('input.type.numberpad') 来检测。
password	Boolean		是否是密码类型	-
placeholder	String		占位符	-
placeholder-style	String		指定 placeholder 的样式	1.6.0
placeholder-class	String		指定 placeholder 的样式类	1.6.0
disabled	Boolean		是否禁用	-
maxlength	Number		最大长度	-
focus	Boolean		获取焦点	-
confirm-type	String		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	1.7.0
confirm-hold	Boolea		点击键盘右下角按钮时是否保持键盘不收起状态	1.7.0

	n			
cursor	Number		指定focus时的光标位置	-
selection-start	Number		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	1.7.0
selection-end	Number		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	1.7.0
randomNumber	Boolean		当type为number、digit、idcard数字键盘是否随机排列	1.9.0
controlled	Boolean		是否为受控组件。为 true时，value内容会完全受setData控制	1.8.0
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	EventHandl		失去焦点时触发，event.detail = {value: value}	-

	e				

图示

代码示例

```
<input placeholder="此处只有在点击下方按钮时才聚焦"focus="{{focus}}"/>
<input maxlength="10"placeholder="最大输入长度10"/>
<input onInput="bindKeyInput"placeholder="输入同步到view中"/>
<input type="number"placeholder="这是一个数字输入框"/>
<input password type="text"placeholder="这是一个密码输入框"/>
<input type="digit"placeholder="带小数点的数字键盘"/>
<input type="idcard"placeholder="身份证输入键盘"/>


Page({
data: {
focus: false,
inputValue: '',
},
bindButtonTap() {
this.setData({
focus: true,
});
},
bindKeyInput(e) {
this.setData({
inputValue: e.detail.value,
});
},
});
```

9.5.5 textarea

本文介绍多行输入框（textarea）。

属性名	类型	默认值	描述	最低版本
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	1.6.0
placeholder-class	String		指定 placeholder 的样式类	1.6.0
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	
auto-height	Boolean	false	是否自动增高	
show-count	Boolean	true	是否渲染字数统计功能	1.8.0
controlled	Boolean	false	是否为受控组件。为true时，value内容会完全受setData控制	1.8.0
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}	

图示

自动聚焦

0/140

受控聚焦

0/140

聚焦

自适应高度

代码示例

```
<view class="section">
<textarea onBlur="bindTextAreaBlur"auto-height placeholder="自动变高"/>
</view>
<view class="section">
<textarea placeholder="这个只有在按钮点击的时候才聚焦"focus="{{focus}}"/>
<view class="btn-area">
<button onTap="bindButtonTap">使得输入框获取焦点</button>
</view>
</view>
<view class="section">
<form onSubmit="bindFormSubmit">
<textarea placeholder="form 中的 textarea"name="textarea"/>
<button form-type="submit"> 提交 </button>
</form>
</view>


Page({
data: {
focus: false,
inputValue: ''
},
bindButtonTap() {
this.setData({
focus: true
})
},
bindTextAreaBlur: function(e) {
console.log(e.detail.value)
},
bindFormSubmit: function(e) {
console.log(e.detail.value.textarea)
}
})
```

9.5.6 radio

本文介绍单项选择器（radio）。

radio-group

单项选择器组。

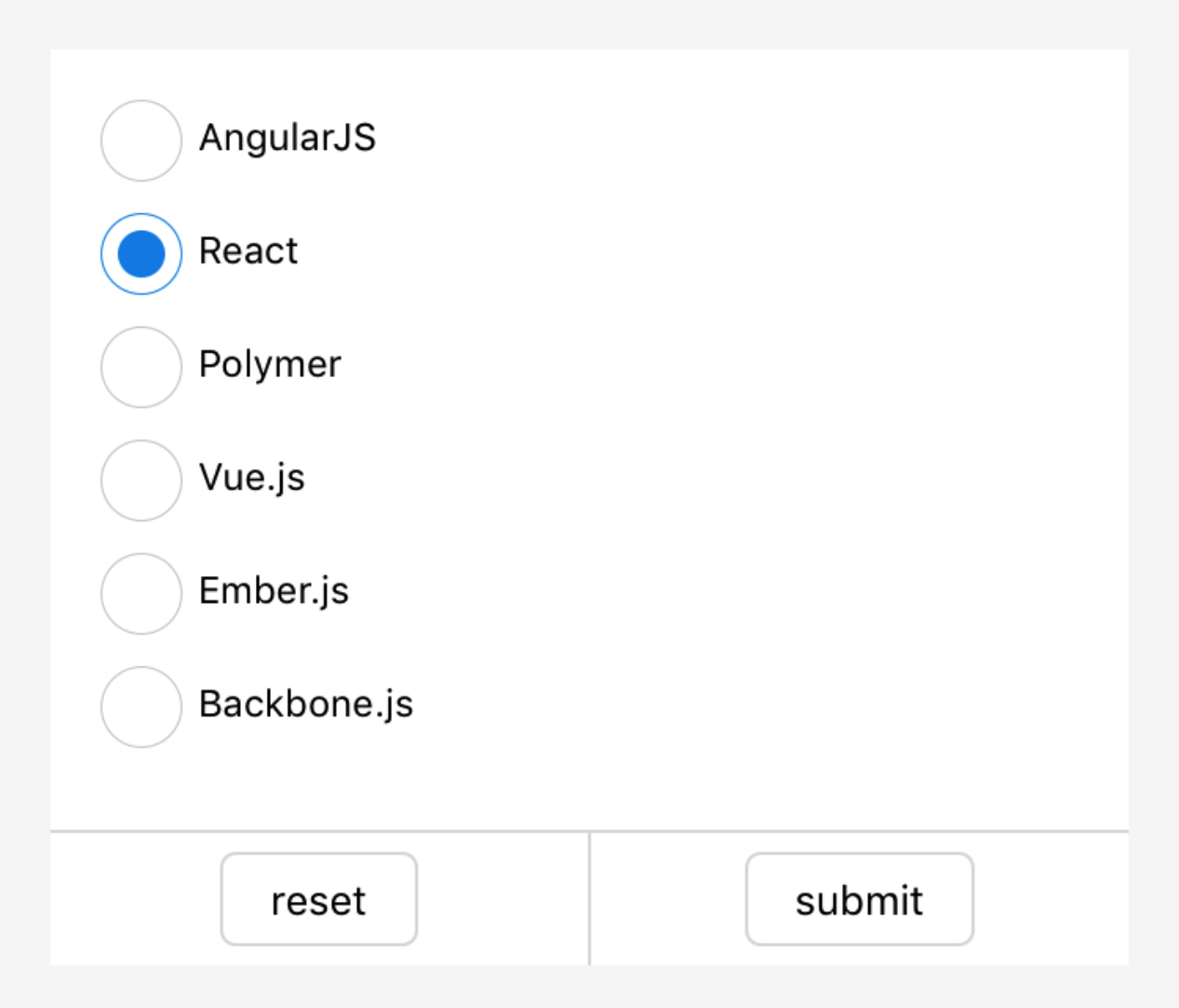
属性名	类型	默认值	描述	最低版本
onChange	EventHandle		选中项发生变化时触发，event.detail = {value: 选中项 radio 的 value}	-
name	String		组件名字，用于表单提交获取数据	-

radio

单选项目。

属性名	类型	默认值	描述	最低版本
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	Color		radio 的颜色	1.10.0

图示



示例代码

```
<radio-group class="radio-group"onChange="radioChange">
<label class="radio"a:for="{{items}}">
<radio value="{{item.name}}"checked="{{item.checked}}"/>{{item.value}}
</label>
</radio-group>
```

```
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'},
]
},
radioChange: function(e) {
console.log('你选择的框架是：', e.detail.value)
}
})
```

9.5.7 checkbox

本文介绍多项选择器（checkbox）。

checkbox-group

多项选择器组。

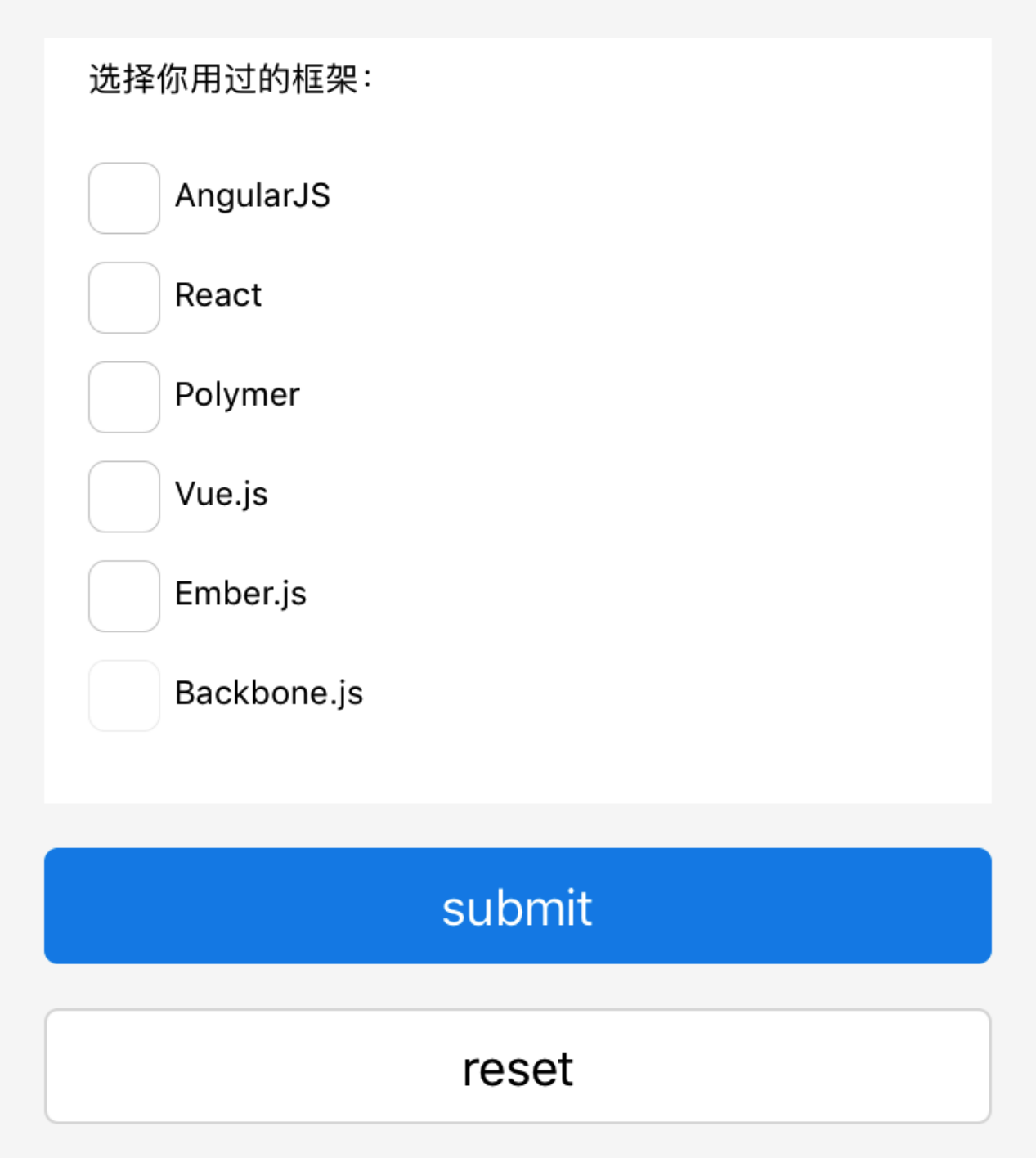
属性名	类型	默认值	描述	最低版本
name	String		组件名字，用于表单提交获取数据	-
onChange	EventHandle		中选中项发生改变时触发，detail = {value: 选中的checkbox项value的值}	-

checkbox

多选项目。

属性名	类型	默认值	描述	最低版本
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	Color	-	checkbox的颜色	1.10.0

图示



代码示例

```
// acss
.checkbox {
display: block;
margin-bottom: 20rpx;
}

.checkbox-text {
font-size:34rpx;
```

```
line-height: 1.2;
}


<checkbox-group onChange="onChange">
<label class="checkbox"a:for="{{items}}">
<checkbox value="{{item.name}}"checked="{{item.checked}}"disabled="{{item.disabled}}"/>
<text class="checkbox-text">{{item.value}}</text>
</label>
</checkbox-group>


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},
],
},
onChange(e) {
my.alert({
title: `你选择的框架是 ${e.detail.value}`,
});
},
});
```

9.5.8 switch

本文介绍单选项目（switch）。

属性名	类型	默认值	描述	最低版本
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	
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	1.8.0
color	Color		switch 的颜色	1.10.0

图示

代码示例

```
<view class="page">
<view class="switch-list">
<view class="switch-item">
<switch checked onChange="switchChange"/>
</view>
</view>
</view>


Page({
switchChange (e){
console.log('switchChange 事件，值:', e.detail.value)
},
})
```

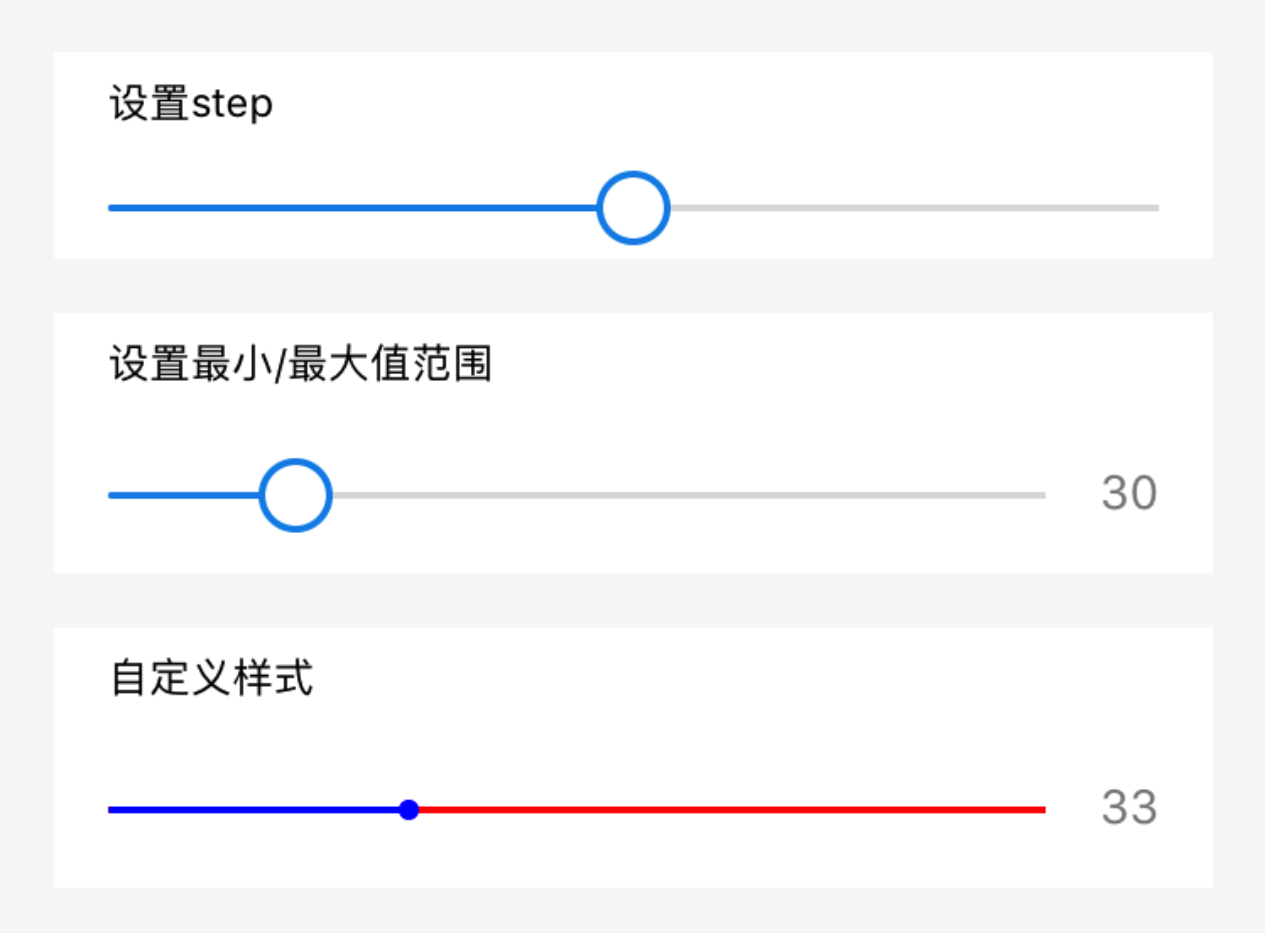
9.5.9 slider

本文介绍滑动选择器（slider）。

属性名	类型	默认值	描述	最低版本
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	-
background-color	String	#ddd	背景条的颜色	-
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	-
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}	1.5.0

图示



代码示例

```
<view class="section section-gap">
<text class="section-title">设置step</text>
<view class="body-view">
<slider value="60"onChange="sliderChange"step="5"/>
</view>
</view>
```

```
<view class="section section-gap">
<text class="section-title">显示当前value</text>
<view class="body-view">
<slider value="50"show-value/>
</view>
</view>

<view class="section section-gap">
<text class="section-title">设置最小/最大值</text>
<view class="body-view">
<slider value="100"min="50"max="200"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>


Page({
sliderChange(e)
console.log('slider 改变后的值:', e.detail.value)
})
```

9.5.10 picker-view

本文介绍嵌入页面的滚动选择器（picker-view）。

属性名	类型	默认值	描述	最低版本
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	1.10
mask-style	String	-	蒙层的样式	1.10
mask-class	String	-	蒙层的类名	1.10
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	-

说明：其中只可放置组件，其他节点不会显示。该组件请勿放入 hidden 或 display none 的节点内部，需要隐藏请用 a:if 切换。

推荐用法：

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

错误用法：

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

图示

代码示例

```
<!-- 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>
```

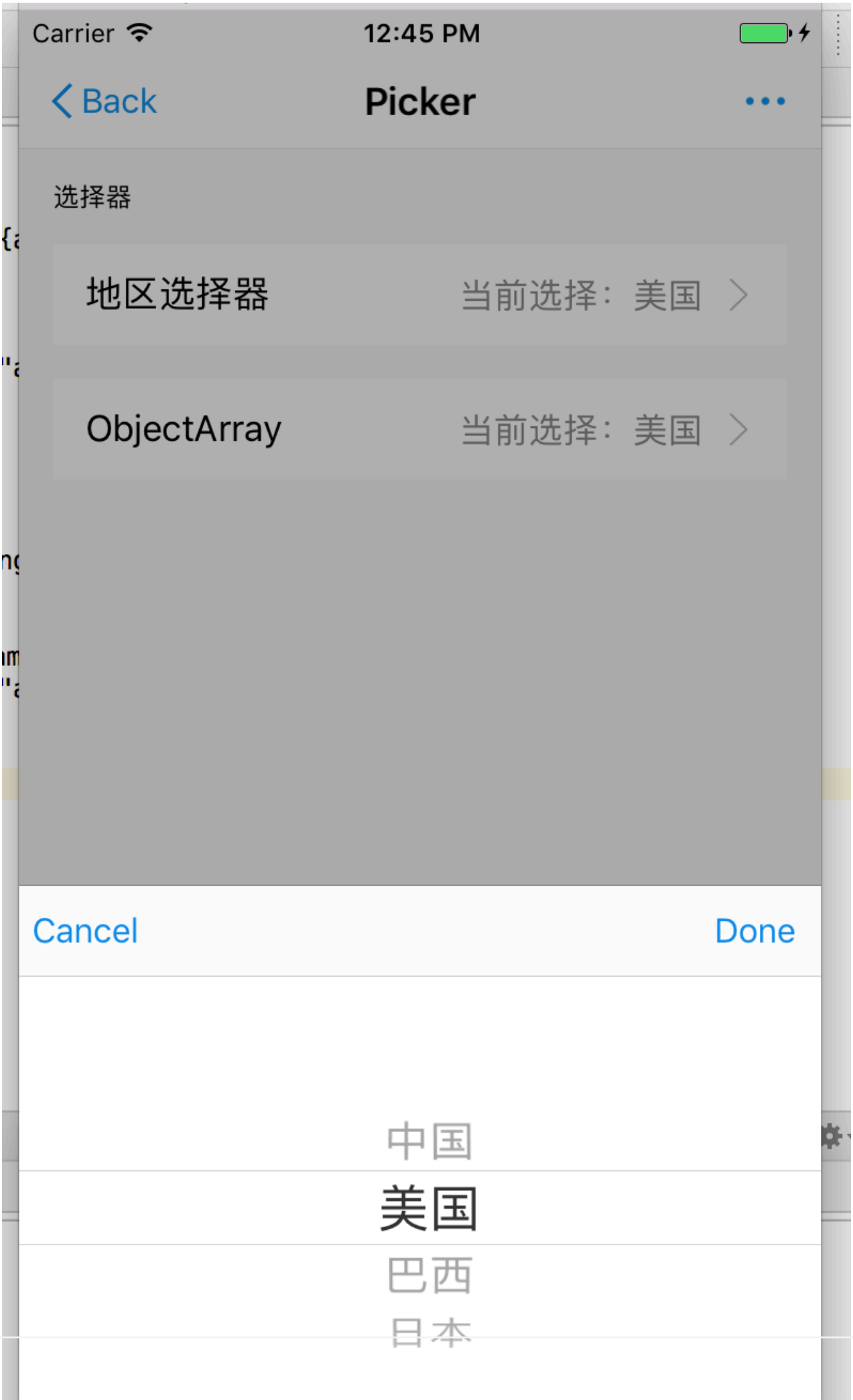
```
<view>夏</view>
<view>秋</view>
<view>冬</view>
</picker-view-column>
</picker-view>
</view>
</view>
</view>


// 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;
}
```

9.5.11 picker

本文介绍从底部弹起的滚动选择器（picker）。

属性名	类型	默认值	描述	最低版本
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	-

图示


Carrier
12:45 PM
Back
Picker
选择器
地区选择器
当前选择：美国
ObjectArray
当前选择：美国
Cancel
Done
中国
美国
巴西
日本

代码示例

```
<view class="section">
<view class="section-title">地区选择器</view>
<picker onChange="bindPickerChange"value="{{index}}"range="{{array}}">
<view class="picker">
当前选择：{{array[index]}}
</view>
</picker>

<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>


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,
```

```
});
},
});
```

9.6 navigator

本文介绍页面链接（navigator）。

属性名	类型	默认值	描述	最低版本
open-type	String	navigate	跳转方式	-
hover-class	String	none	点击时附加的类	-
hover-start-time	Number	-	按住后多事件后出现点击状态，单位毫秒	-
hover-stay-time	Number	-	手指松开后点击状态保留时间，单位毫秒	-
url	String	-	应用内的跳转链接	-

open-type 有效值

属性名	描述	最低版本
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	-

代码示例

```
<!-- sample.axml -->
<view class="btn-area">
<navigator url="/page/navigate/navigate?title=navigate"hover-class="navigator-hover">跳转到新页面</navigator>
<navigator url="../../redirect/redirect/redirect?title=redirect"open-type="redirect"hover-class="other-navigator-hover">在当前页打开</navigator>
<navigator url="/page/index/index"open-type="switchTab"hover-class="other-navigator-hover">切换Tab</navigator>
</view>
```

9.7 媒体组件

9.7.1 图片

本文介绍图片（image）。

属性名	类型	默认值	描述	最低版本
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。	1.9.0
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，并阻止事件冒泡	-

说明：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

属性名	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

属性名	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.

图示

原图

scaleToFill

不保持纵横比缩放，使图片完全适应。

aspectFit

保持纵横比缩放，使图片的长边能完全显示出来。

aspectFill

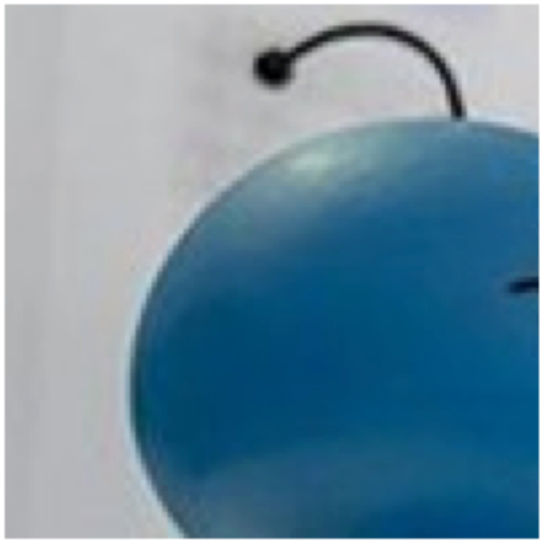
保持纵横比缩放，只保证图片的短边能完全显示出来。

widthFix

宽度不变，高度自动变化，保持原图宽高比不变。

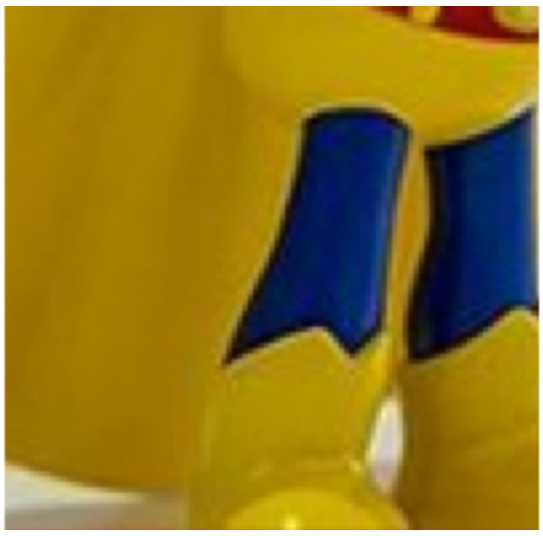
top

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。

代码示例

```
<view class="section"a:for="{{array}}"a:for-item="item">
<view class="title">{{item.text}}</view>
<image style="background-color: #eeeeee; width: 300px;
height:300px;"mode="{{item.mode}}"src="{{src}}"onError="imageError"onLoad="imageLoad"/>
</view>


Page({
data: {
array: [{
mode: 'scaleToFill',
text: 'scaleToFill：不保持纵横比缩放，使图片完全适应'
}, {
mode: 'aspectFit',
text: 'aspectFit：保持纵横比缩放，使图片的长边能完全显示出来'
}, {
mode: 'aspectFill',
text: 'aspectFill：保持纵横比缩放，只保证图片的短边能完全显示出来'
}, {
```

```
mode: 'top',
text: 'top：不缩放图片，只显示顶部区域'
}, {
mode: 'bottom',
text: 'bottom：不缩放图片，只显示底部区域'
}, {
mode: 'center',
text: 'center：不缩放图片，只显示中间区域'
}, {
mode: 'left',
text: 'left：不缩放图片，只显示左边区域'
}, {
mode: 'right',
text: 'right：不缩放图片，只显示右边边区域'
}, {
mode: 'top left',
text: 'top left：不缩放图片，只显示左上边区域'
}, {
mode: 'top right',
text: 'top right：不缩放图片，只显示右上边区域'
}, {
mode: 'bottom left',
text: 'bottom left：不缩放图片，只显示左下边区域'
}, {
mode: 'bottom right',
text: 'bottom right：不缩放图片，只显示右下边区域'
}],
src: './2.png'
},
imageError: function (e) {
console.log('image3 发生错误', e.detail.errMsg)
},
imageLoad: function (e) {
console.log('image 加载成功', e);
}
})
```

9.7.2 视频

基础库版本 1.10.0 开始支持，低版本需要做 兼容处理。

用户可通过 video 组件上传并播放视频。相关 API：my.createVideoContext

代码示例

请在 .amxl 文件中写入以下代码：

```
<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}}"
mobilenetHintType="{{video.mobilenetHintType}}"
onPlay="onPlay"
onPause="onPause"
onEnded="onEnded"
onError="onPlayError"
onTimeUpdate="onTimeUpdate"
/>
</view>
```

请在 .js 文件中写入以下代码：

```
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,
mobilenetHintType: 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
src	String		要播放视频的资源地址。 src 支持的协议如下： **apFilePath**: https://resource/xxx.video
poster	String		视频封面图的 URL，支持 jpg、png 等格式的图片，如 https://***.jpg。如果不传的话，默认取视频的首帧图作为封面图。
objectFit	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。
initial-time	Number		指定视频初始播放位置，单位为秒。
duration	Number		指定视频时长，单位为秒，默认读取视频本身市场信息。
controls	Boolean	true	是否显示默认播放控件（底部工具条，包括播放/暂停按钮、播放进度、时间）。
autoplay	Boolean	false	是否自动播放。
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 度）。
loop	Boolean	false	是否循环播放。
muted	Boolean	false	是否静音播放。
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。
enableProgressGesture	Boolean	true	全屏模式下是否开启控制进度的手势。

mobilenetHintType	Number	1	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）。
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’}
onLoading	EventHandle		视频出现缓冲时触发。
onError	EventHandle		视频播放出错时触发（errorCode 见下方 错误码 列表）。
onFullScreenChange	EventHandle		视频进入和退出全屏时触发，event.detail = {fullScreen, direction}，direction 取为 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		播放器内部错误
1005		网络连接失败
1006	loading 异常	数据源错误
1007		播放器准备失败
1008		网络错误
1009		搜索视频出错（源出错的一种）
1010		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		读 ups 信息超时

3001		audio 渲染出错
3002		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		播放中内部错误（ffmpeg 内错误）

支持的视频封装格式

格式	iOS	Android
mp4	支持	支持
mov	支持	支持
m4v	支持	支持
3gp	支持	支持
m3u8	支持	支持
flv	支持	支持

支持的编码格式

编码格式	iOS	Android
H.264	支持	支持
AAC	支持	支持

9.8 canvas

本文介绍画布（canvas）。

属性名	类型	默认值	描述	最低版本
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	-
onTouchEn	EventHa	-	触摸动作结束	-

d	ndle			
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	-

说明：

- 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;"/>
```

图示

代码示例

```
<canvas
id="canvas"
class="canvas"
onTouchStart="log"
onTouchMove="log"
```

```
onTouchEnd="log"
/>


Page({
onReady() {
this.point = {
x: Math.random() * 295,
y: Math.random() * 295,
dx: Math.random() * 5,
dy: Math.random() * 5,
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);
},

draw() {
var ctx = my.createCanvasContext('canvas');
ctx.setFillStyle('#FFF');
ctx.fillRect(0, 0, 305, 305);

ctx.beginPath();
ctx.arc(this.point.x, this.point.y, 10, 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 <= 5 || this.point.x >= 295) {
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 <= 5 || this.point.y >= 295) {
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)
}
})
```

9.9 map

本文介绍地图组件（map）。

相关API： my.createMapContext(mapId)

使用说明

- map 组件是由客户端创建的原生组件，原生组件的层级是最高的，所以页面中的其他组件无论设置 z-index 为多少，都无法在原生组件之上。
- 请勿在 scroll-view 中使用 map 组件。
- css 动画对 map 组件无效。
- 缩小或者放大了地图比例尺之后，请在 onRegionChange 函数中重新设置 data 的 scale 值，否则会出现拖动地图区域后，重新加载导致地图比例尺又变回缩放前的大小，具体请参照示例代码 regionchange 函数部分。基础库1.14.0以上，可以使用default-scale属性替代scale来优化代码。

Map

同一个页面需要展示多个 map 组件的话，需要使用不同的 ID 。

属性	类型	默认值	说明	支持版本
style	String		内联样式	
class	String		样式名	
latitude	Number		中心纬度	10.1.32
longitude	Number		中心经度	10.1.32
scale	Number	16	缩放级别，取值范围为 5-18	10.1.32
default-scale	Number	16	默认缩放级别，取值范围为 5-18，如果只需指定初始 scale，可以设置 default-scale 来替代 scale。当用户缩放后，也不需要再监听 onRegionChange 重新设置 scale	基础库 1.14.0
markers	Array		覆盖物，在地图上的一个点绘制图标	10.1.32
polyline	Array		覆盖物，多个连贯点的集合（路线）	10.1.32
circles	Array		覆盖物，圆	10.1.32
controls	Array		在地图View之上的一	10.1.32

			个控件	
polygon	Array		覆盖物，多边形	10.1.32
show-location	Boolean		是否显示带有方向的当前定位点	10.1.32
include-points	Array		视野将进行小范围延伸包含传入的坐标 [{ latitude: 30.279383, longitude: 120.131441,}]	10.1.32
include-padding	Object		视野在地图 padding 范围内展示 { left:0, right:0, top:0, bottom:0}	10.1.35
ground-overlays	Array		覆盖物，自定义贴图 [{ // 右上，左下 'include-points':[{ latitude: 39.935029, longitude: 116.384377, },{ latitude: 39.939577, longitude: 116.388331, }], image:'/image/overlay.png', alpha:0.25, zIndex:1}]	10.1.35
tile-overlay	Object		覆盖物，网格贴图 { url:' http://xxx', type:0, // url类型 tileWidth:256, tileHeight:256, zIndex:1,}	10.1.35
setting	Object		设置 { // 手势 gestureEnable: 1, // 比例尺 showScale: 1, // 指南针 showCompass: 1, //双手下滑 tiltGesturesEnabled: 1, // 交通路况展示 trafficEnabled: 0, // 地图 POI 信息 showMapText: 0, // 高德地图 logo 位置 logoPosition:	

			{ centerX: 150, centerY: 90 }}	
onMarkerTap	EventHandle		点击Marker时触发 { markerId, latitude, longitude,}	10.1.32
onCalloutTap	EventHandle		点击Marker对应的callout时触发 { markerId, latitude, longitude,}	10.1.32
onControlTap	EventHandle		点击control时触发 { controlId}	10.1.32
onRegionChange	EventHandle		视野发生变化时触发 { type:"begin/end", latitude, longitude, scale}	10.1.32
onTap	EventHandle		点击地图时触发 { latitude, longitude,}	10.1.32

markers

标记点，用于在地图上显示标记的位置。

属性名	说明	类型	必填	备注	最低版本
id	标记点id	Number	否	标记点 id，点击事件回调会返回此 id	
latitude	纬度	Float	是	范围 -90 ~ 90	
longitude	经度	Float	是	范围 -180 ~ 180	
title	标注点名	String	否		
iconPath	显示的图标	String	是	项目目录下的图片路径，可以用相对路径写法，以'/'开头则表示相对小程序根目录	
rotate	旋转角度	Number	否	顺时针旋转的角度，范围 0 ~ 360，默认为 0	
alpha	标注的透明度	Number	否	是否透明，默认	

				为 1	
width	标注图标宽度	Number	否	默认为图片的实际宽度	
height	标注图标高度	Number	否	默认为图片的实际高度	
callout	自定义标记点上方的气泡窗口	Object	否	marker 上的气泡，地图上最多同时展示一个，绑定 onCalloutTap `{ content:"xxx" }`	
anchorX	经纬度在标注图标的锚点-横向值	Double	否	这两个值需要成对出现，anchorX表示横向(0-1)，y表示竖向(0-1)，anchorX:0.5, anchorY:1 表示底边中点	
anchorY	经纬度在标注图标的锚点-竖向值	Double	否		
customCallout	callout背景自定义 目前只支持高德地图style	Object	否	`{"type": 2,"descList": [{"desc":"预计","descColor":"#333333"}, {"desc":"5分钟","descColor":"#108EE9"}, {"desc":"到达","descColor":"#333333"}], "isShow": 1}`	
iconAppendStr	marker图片可以来源于View	String	否	和iconPath一起使用，会将iconPath对应的图片及该字符串共同生成一个图片，当成marker的图标	
iconAppendStrColor	marker图片可以来源于View,底部描述文本颜色	String	否	默认是：#33B276	
fixedPoint	基于屏幕位置扎点	Object	否	基于屏幕位置扎点 `{ //距离地图左上角的像素数,Number originX:100, originY:100}`	
markerLevel	marker在地图上的绘制层级	Number	否	与地图上其他覆盖物统一的 Z 坐标系	10.1.32
label	marker 上的气	Object	否	marker 上的气	10.1.38

	泡			泡，地图上可同时展示多个，绑定 onMarkerTap { content:"Hello Label", color:"#000000", fontSize:12, borderRadius:3, bgColor:"#ffffff", padding:5,}	
style	自定义marker样式	Object	否	自定义marker的样式和内容	10.1.35
					

polygon

用于构造多边形对象。

属性名	说明	类型	必填	备注	支持版本
points	经纬度数组	Array	是	[{ latitude: 0, longitude: 0}]	10.1.32
color	线的颜色	String	否	用 8 位十六进制表示，后两位表示 alpha 值，如：#eeeeeeAA	10.1.32
fillColor	填充色	String	否	用 8 位十六进制表示，后两位表示 alpha 值，如：#eeeeeeAA	10.1.32
width	线的宽度	Number	否		10.1.32

polyline

用于指定一系列坐标点，从数组第一项连线至最后一项。

属性名	说明	类型	必填	备注	最低版本
points	经纬度数组	Array	是		

				[{ latitude: 0, longitude: 0}]	
color	线的颜色	String	否	用 8 位十六进制表示，后两位表示 alpha 值，如：#eeeeeeAA	
width	线的宽度	Number	否		
dottedLine	是否虚线	Boolean	否	默认 false	
iconPath	线的纹理地址	String	否	项目目录下的图片路径，可以用相对路径写法，以'/'开头则表示相对小程序根目录	10.1.35
iconWidth	使用纹理时的宽度	Number	否		10.1.35
zIndex	覆盖物的 Z 轴坐标	Number			10.1.35
iconPath	纹理	String		项目目录下的图片路径，可以用相对路径写法，以'/'开头则表示相对小程序根目录， 如果有iconPath，会忽略color。但是iconPath可以和colorList联合使用，这样纹理会浮在彩虹线上方，为避免覆盖彩虹线，纹理图片背景色可以设置透明	10.1.35
colorList	彩虹线	Array		彩虹线，分段依据points。例如points有5个点，那么colorList就应该传4个颜色值，依此类推。如果colorList数量小于4，那么剩下的线路颜色和最后一个颜色一样 ["#AAAAAA", "#BBBBBB"]	10.1.38

circles

用于在地图上显示圆。

属性名	说明	类型	必填	备注	支持版本
latitude	纬度	Float	是	范围 -90 ~ 90	10.1.32

longitude	经度	Float	是	范围 -180 ~ 180	10.1.32
color	描边的颜色	String	否	用 8 位十六进制表示，后两位表示 alpha 值，如：#eeeeeeAA	10.1.32
fillColor	填充颜色	String	否	用 8 位十六进制表示，后两位表示 alpha 值，如：#eeeeeeAA	10.1.32
radius	半径	Number	是		10.1.32
strokeWidth	描边的宽度	Number	否		10.1.32

controls

用于在地图上显示控件，控件不随着地图移动。

属性名	说明	类型	必填	备注	支持版本
id	控件id	Number	否	控件 id,点击事件回调会返回此 id	10.1.32
position	控件在地图的位置	Object	是	相对地图位置	10.1.32
iconPath	显示的图标	String	是	项目目录下的图片路径，可以用相对路径写法，以’/‘开头则表示相对小程序根目录	10.1.32
clickable	是否可点击	Boolean	否	默认为false	10.1.32

position

控件在地图的位置，以及控件的大小。

属性名	说明	类型	必填	备注
left	距离地图的左边界多远	Number	否	默认为0
top	距离地图的上边界多远	Number	否	默认为0
width	控件宽度	Number	否	默认为图片宽度
height	控件高度	Number	否	默认为图片高度

callout

自定义标记点上方的气泡窗口。

属性名	说明	类型	必填	备注
content	内容	String	否	默认为空（null）

customCallout

自定义 callout 背景。目前只支持高德地图 style 。

属性名	说明	类型	必填	备注
type	样式类型	Number	是	0为黑色style,1为白

				色style,2为背景+文本，见下图
time	时间	String	是	时间值
descList	描述数组	Array	是	描述数组 {"type": 0,"time":"3","descList": [{"desc":"点击立即打车","descColor":"#ffffff"}],"isShow": 1}

fixedPoint

基于屏幕位置的扎点。| 属性名 | 说明 | 类型 | 必填 | 备注 || :——— | :————- | :——- | :—- | :———————————— || originX | 横向像素点 | Number | 是 | 距离地图左上角的像素数值

，从0开始 || originY | 纵向像素点 | Number | 是 | 距离地图左上角的像素数值，从0开始 |

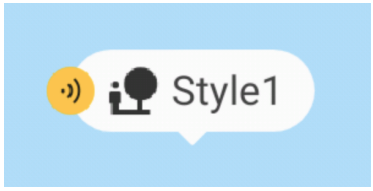
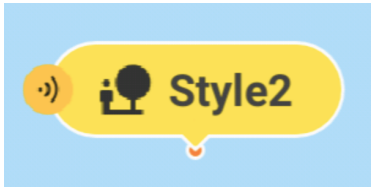
地图组件的经纬度是必需设置的, 若未设置经纬度，则默认是北京的经纬度。

Marker 图鉴

Marker 样式优先级说明

- customCallout，callout 与 label 互斥，优先级排序为：label > customCallout > callout 。
- style 与 icon 互斥，优先级排序为：style > iconAppendStr ; style > icon 。

style

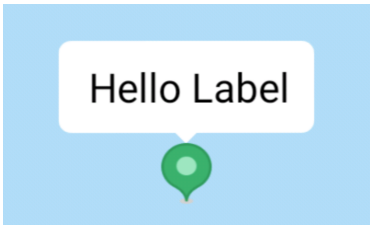
结构	图片示例	支持版本
{ type:1, text1:"Style1", icon1:'xxx', icon2:'xxx'}		10.1.35
{ type:2, text1:"Style2", icon1:'xxx', icon2:'xxx'}		10.1.38
{ type:3, icon:xxx, //选填 text:xxx, //必填 color:xxx, //默认#33B276 bgColor:xxx, //默认#FFFFFF gravity:"left/center/right", //默认 center fontType:"small/standard/large"//默认 standard}		10.1.50

customCallout

结构	图片示例	支持版本
{"type": 0,"time":"3","descList": [{"desc":"点击立即打车 ","descColor":"#ffffff"}],"isShow": 1}		10.1.32
{"type": 1,"time":"3","descList": [{"desc":"点击立即打车 ","descColor":"#333333"}],"isShow": 1}		10.1.32

{"type": 2,"descList": [{"desc":"预计","descColor":"#333333"}, {"desc":"5分钟","descColor":"#108EE9"}, {"desc":"到达","descColor":"#333333"}],"isShow": 1}		10.1.32

label- content：必填- color：选填，默认”#000000”- fontsize：选填，默认14- borderRadius：选填，默认20- bgColor：选填，默认”#FFFFFF”- padding：选填，默认10

结构	图鉴	支持版本
{ content:"Hello Label", color:"#000000", fontSize:16, borderRadius:5, bgColor:"#ffffff", padding:12,}		10.1.38

图示

改 scale | getCenterLocation | moveToLocation

改 center | 改 markers

示例代码

```
<view>
<map id="map"longitude="120.131441"latitude="30.279383"scale="{{scale}}"controls="{{controls}}"
onControlTap="controltap"markers="{{markers}}"
onMarkerTap="markertap"
```

```
polyline="{{polyline}}"circles="{{circles}}"
onRegionChange="regionchange"
onTap="tap"
show-location style="width: 100%; height: 300px;"
include-points="{{includePoints}}"></map>
<button onTap="changeScale">改scale</button>
<button onTap="getCenterLocation">getCenterLocation</button>
<button onTap="moveToLocation">moveToLocation</button>
<button onTap="changeCenter">改center</button>
<button onTap="changeMarkers">改markers</button>
</view>


Page({
data: {
scale: 14,
longitude: 120.131441,
latitude: 30.279383,
markers: [{
iconPath:"/image/green_tri.png",
id: 10,
latitude: 30.279383,
longitude: 120.131441,
width: 50,
height: 50
},{
iconPath:"/image/green_tri.png",
id: 10,
latitude: 30.279383,
longitude: 120.131441,
width: 50,
height: 50,
customCallout: {
type: 1,
time: '1',
},
fixedPoint:{
originX: 400,
originY: 400,
},
iconAppendStr: '黄龙时代广场黄龙时代广场黄龙时代广场黄龙时代广场test'
}],
includePoints: [{
latitude: 30.279383,
longitude: 120.131441,
}],
polyline: [{
points: [{
longitude: 120.131441,
latitude: 30.279383
}, {
longitude: 120.128821,
latitude: 30.278200
}, {
longitude: 120.131618,
latitude: 30.277600
```

```
}, {
longitude: 120.132520,
latitude: 30.279393
}, {
longitude: 120.137517,
latitude: 30.279383
}],
color:"#FF0000DD",
width: 5,
dottedLine: false
}],
circles: [{
latitude: 30.279383,
longitude: 120.131441,
color:"#000000AA",
fillColor:"#000000AA",
radius: 80,
strokeWidth: 5,
}],
controls: [{
id: 5,
iconPath: '../../resources/pic/2.jpg',
position: {
left: 0,
top: 300 - 50,
width: 50,
height: 50
},
clickable: true
}]
},

onReady(e) {
// 使用 my.createMapContext 获取 map 上下文
this.mapCtx = my.createMapContext('map')
},

getCenterLocation() {
this.mapCtx.getCenterLocation(function (res) {
console.log(res.longitude)
console.log(res.latitude)
})
},

moveToLocation() {
this.mapCtx.moveToLocation()
},

regionchange(e) {
console.log('regionchange', e);
// 注意：如果缩小或者放大了地图比例尺以后，请在 onRegionChange 函数中重新设置 data 的
// scale 值，否则会出现拖动地图区域后，重新加载导致地图比例尺又变回缩放前的大小。
if (e.type === 'end') {
this.setData({
scale: e.scale
});
```

```
}
},

markertap(e) {
console.log('marker tap', e);
},

controltap(e) {
console.log('control tap', e);
},

tap() {
console.log('tap:');
},

changeScale() {
this.setData({
scale: 8,
});
},

changeCenter() {
this.setData({
longitude: 113.324520,
latitude: 23.199994,
includePoints: [{
latitude: 23.199994,
longitude: 113.324520,
}],
});
},
//支持地图不接受手势事件, isGestureEnable为1 表示支持，为0表示不支持
gestureEnable() {
this.mapCtx.gestureEnable({isGestureEnable:1});
},
//地图是否显示比例尺, showsScale 为1表示显示，为0表示不显示
showsScale() {
this.mapCtx.showsScale({isShowsScale:1});
},
//地图是否显示指南针, showsCompass 为1表示显示，为0表示不显示
showsCompass() {
this.mapCtx.showsCompass({isShowsCompass:1});
},
changeMarkers() {
this.setData({
markers: [{
iconPath:"/image/green_tri.png",
id: 10,
latitude: 21.21229,
longitude: 113.324520,
width: 50,
height: 50
}],
includePoints: [{
latitude: 21.21229,
longitude: 113.324520,
```

```
}],
});
},
})
```

9.10 开放组件

9.10.1 web-view

<web-view /> 组件用于承载 H5 网页，自动铺满整个小程序页面。

属性名	类型	默认值	描述
src	String	无	web-view 要渲染的 H5 网页 URL
onMessage	EventHandle	无	网页向小程序 postMessage 消息。e.detail = { data }

说明：基础库 1.6.0 开始支持，低版本需做兼容处理，操作参见 小程序基础库说明 。

每个页面只能有一个<web-view />，请不要渲染多个<web-view />，会自动铺满整个页面，并覆盖其它组件。

代码示例

```
<!-- axml -->
<!-- 指向支付宝首页的web-view -->
<web-view src="https://ds.alipay.com/"onMessage="test"></web-view>
```

相关 API

<web-view /> H5页面可以使用手动引入 https://appx/web-view.min.js（此链接仅支持在 mPaaS 客户端内访问），提供了相关的接口返回小程序页面。支持的接口有：

接口类别	接口名	说明
导航栏	my.navigateTo	保留当前页面，跳转到应用内的某个指定页面
导航栏	my.navigateBack	关闭当前页面，返回上一级或多级页面
导航栏	my.switchTab	跳转到指定 tabBar 页面，并关闭其他所有非 tabBar 页面
导航栏	my.reLaunch	关闭当前所有页面，跳转到应用内的某个指定页面
导航栏	my.redirectTo	关闭当前页面，跳转到应用内的某个指定页面
图片	my.chooseImage	拍照或从手机相册中选择图片
图片	my.previewImage	预览图片
位置	my.getLocation	获取用户当前的地理位置信息
位置	my.openLocation	使用支付宝内置地图查看位置

交互反馈	my.alert	alert 警告框
交互反馈	my.showLoading	显示加载提示
交互反馈	my.hideLoading	隐藏加载提示
缓存	my.setStorage	将数据存储在本地缓存中指定的 key 中，会覆盖掉原来该 key 对应的数据
缓存	my.getStorage	获取缓存数据
缓存	my.removeStorage	删除缓存数据
缓存	my.clearStorage	清除本地数据缓存
缓存	my.getStorageInfo	异步获取当前storage的相关信息
网络状态	my.getNetworkType	获取当前网络状态
分享	my.startShare	分享当前页面,当执行 my.startShare() 时会唤起当前小程序页面的分享功能
唤起支付	my.tradePay	唤起支付
向小程序发送消息	my.postMessage	向小程序发送消息，自定义一组或多组key、value数据，格式为JSON，如：my.postMessage({name:”测试web-view”})
监听小程序发过来的消息	my.onMessage	监听小程序发过来的消息， webview 组件控制
获取当前环境	my.getEnv	获取当前环境

代码示例

<web-view />H5页面：

```
<!-- html -->
<script type="text/javascript"src="https://appx/web-view.min.js"></script>
// 如该 H5 页面需要同时在非 mPaaS 客户端内使用，为避免该请求404，可参考以下写法
// 请尽量在 html 头部执行以下脚本
<script>
if (navigator.userAgent.indexOf('AlipayClient') > -1 || navigator.userAgent.indexOf('mPaaSClient') > -1) {
document.writeln('<script src="https://appx/web-view.min.js"' + '>' + '<' + '/' + 'script>');
}

// javascript
my.navigateTo({url: '../get-user-info/get-user-info'});

// 网页向小程序 postMessage 消息
my.postMessage({name:"测试web-view"});

// 接收来自小程序的消息。
my.onMessage = function(e) {
console.log(e); //{'sendToWebView': '1'}
}
```

```
// 判断是否运行在小程序环境里
my.getEnv(function(res) {
console.log(res.miniprogram) // true
});

my.startShare();

</script>
```

my.postMessage 信息发送后，小程序页面接收信息时，会执行 onMessage 配置的方法：

```
// 小程序页面对应的 page.js 声明 test 方法，
// 由于 page.axml 里的 web-view 组件设置了 onMessage="test",
// 当网页里执行完 my.postMessage 后，test 方法会被执行
Page({
onLoad(e){
this.webViewContext = my.createWebViewContext('web-view-1');
},
test(e){
my.alert({
content:JSON.stringify(e.detail),
});
this.webViewContext.postMessage({'sendToWebView': '1'});
},
)};
```

my.getEnv 示例代码：

```
// 判断是否运行在小程序环境里
my.getEnv(function(res){
console.log(res.miniprogram); //true
});
```

用户分享时可获取当前<web-view />的 URL，即在 onShareAppMessage 回调中返回 webViewUrl 参数。

```
Page({
onShareAppMessage(options) {
console.log(options.webViewUrl)
}
});
```

常见问题

H5 怎么传递信息给小程序？

请使用 my.postMessage 接口来传递数据，代码示例如下：

```
my.postMessage({key1:"value1",key2:"value2"});
```

小程序如何传递信息给 H5？

<web-view /> 目前已支持了双向通信能力，更多细节参见 webview 组件控制 一节。

webview 里如何返回小程序？

<web-view /> H5 页面可以使用手动引入 https://appx/web-view.min.js(此链接仅支持在 mPaaS 客户端内访问），再调用 my.navigateTo 接口即可。

使用了小程序的 chooseImage 接口，如何在 H5 里进行图片上传？

可将获取到的图片路径通过 my.postMessage() 将相关数据传递给小程序后进行上传。

10 扩展组件

10.1 概述

小程序扩展组件库是 基础组件库 的重要补充，是基于 小程序自定义组件规范 开发的一套开源UI组件库，供小程序开发者快速复用。

安装

```
$ npm install mini-antui --save
```

使用

在页面json中文件中进行注册，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"
/>
```

组件更新日志

更新日志请查看 changelog。

10.2 布局导航

10.2.1 列表（list）

本文介绍列表（list）。

list

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

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list-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

代码示例

```
{
"defaultTitle":"小程序AntUI组件库",
```

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

```
<view>
<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}}"
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>
</view>
`


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',
},
],
},
onItemClick(ev) {
my.alert({
content: `点击了第${ev.index}行`,
});
},
});
```

10.2.2 选项卡（tabs）

选项卡（tabs）可让用户在不同的视图中进行切换。

tabs

属性名	描述	类型	默认值	必选
className	自定义 class	String	-	false
activeCls	自定义激活 tabbar 的 class	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>	-	true
activeTab	当前激活Tab索引	Number	-	true
showPlus	是否显示‘+’icon	Boolean	false	false
onPlusClick	‘+’icon被点击时的回调	() => {}	-	false
onTabClick	tab 被点击的回调	(index:	-	fal

		Number) => void		se
onChange	tab变化时触发	(index: Number) => void	-	fal se
swipeable	是否可以滑动内容切换	Boolean	true	fal se
duration	当swipeable为true时滑动动画时长，单位ms	Number	500(ms)	fal se
tabBarBackgroundColor	tabBar背景色	String	-	fal se
tabBarActiveTextColor	tabBar激活Tab文字颜色	String	-	fal se
tabBarInactiveTextColor	tabBar非激活Tab文字颜色	String	-	fal se
tabBarUnderlineColor	tabBar下划线颜色	String	-	fal se
tabBarCls	tabBar自定义样式class	String	-	fal se

tab-content

视图内容

属性名	描述	类型	默认值	必选	
index	列表项的唯一索引	String	-	-	-

代码示例

```
{
"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}}"
>
<block a:for="{{tabs}}">
<tab-content key="{{index}}">
<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;
height: 300px;
}
```

10.2.3 纵向选项卡（vtabs）

纵向选项卡（vtabs）用于让用户在不同的视图中进行切换。

vtabs

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

代码示例

```
{
"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,
});
},
});
```

10.2.4 卡片（card）

本文介绍卡片（card）。

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
onCardClick	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

代码示例

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

```
<card
thumb="{{thumb}}"
title="卡片标题"
subTitle="副标题非必填"
onClick="onCardClick"
footer="描述文字"
footerImg="{{footerImg}}"
info="dadadadadada"
/>
```

```
Page({
data: {
tagData: [
{ date: '2018-05-14', tag: '还房贷', tagColor: 5 },
{ date: '2018-05-28', tag: '公积金', tagColor: 2 },
],
},
handleSelect() {},
onMonthChange() {},
});
```

10.2.5 宫格（grid）

本文介绍宫格（（grid））。

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

代码示例

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

```
<grid onGridItemClick="onItemClick"columnNum="{{3}}"list="{{list3}}"/>
```

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

10.2.6 步骤条（steps）

本文介绍根据步骤显示的进度条（steps）。

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

items 属性详细描述：

属性名	描述	类型	默认值	必须
items.title	步骤详情标题	String		true
items.description	步骤详情描述	String		true
items.icon	尚未到达步骤的 icon（只在 vertical 模式下生效）	String		true
items.activeIcon	已到达步骤的 icon（只在 vertical 模式下生效）	String		true
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		true

代码示例

```
{
"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',
}]
}
});
```

10.2.7 页脚（footer）

本文介绍显示页面页脚（footer）。

属性名	描述	类型	默认值	必选
copyright	版权信息	String		false
links	页脚链接	Array<text, url>		false

代码示例

```
{
"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: '© 2004-2019 Alipay.com. All rights reserved.',
links: [
{ text: '底部链接', url: '../../list/demo/index' },
{ text: '底部链接', url: '../../card/demo/index' },
],
},
});
```

10.2.8 布局（flex）

本文介绍 CSS flex 布局的封装（flex）。

flex

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、column-reverse	String	row	false
wrap	子元素的换行方式，可选 nowrap、wrap、wrap-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。

代码示例

```
{
"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({});
```

10.2.9 分页（pagination）

本文介绍分页（pagination）。

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	无

prevText和nextText当且仅当mode为text时生效。

代码示例

```
{
"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({})
```

10.2.10 折叠面板（collapse）

本文介绍折叠面板（collapse）。

collapse

属性名	描述	类型	默认值	必选
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	false
onChange	切换面板的回调	(activeKeys: Array): void		false
accordion	手风琴模式	Boolean	false	false
collapseKey	唯一标示 collapse 和对应的 collapse-item	String	false	false

collapse-item

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

代码示例

```
{
"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({});
```

10.3 操作浮层

10.3.1 气泡（Popover）

本文介绍气泡（Popover）。

popover

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

popover-item

属性名	描述	类型	默认值	必选
className	单项样式	String	-	false
onItemClick	单项点击事件	() => void	-	false

代码示例

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

```
<popover
position="{{position}}"
show="{{show}}"
showMask="{{showMask}}"
onMaskClick="onMaskClick"
>
<view onTap="onShowPopoverTap">点击显示</view>
<view slot="items">
<popover-item onItemClick="itemTap1">
<text>line1</text>
</popover-item>
<popover-item>
<text>line2</text>
</popover-item>
</view>
</popover>
```

```
Page({
data: {
position: 'bottomRight',
```

```
show: false,
showMask: true,
},
onMaskClick() {
this.setData({
show: false,
});
},
onShowPopoverTap() {
this.setData({
show: true,
});
},
itemTap1() {
my.alert({
content: '点击1',
});
},
});
```

10.3.2 筛选（Filter）

Filter 用作标签卡筛选。

filter

属性名	描述	类型	默认值	必选
show	是否显示 可选值 show hide	String	hide	false
max	可选数量最大值，1 为单选	Number	10000	false
onChange	多选时提交选中回调	(e: Object) => void	-	false

filter-item

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

代码示例

```
{
"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">
<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,
});
}
});
```

10.3.3 对话框（Modal）

本文介绍对话框（Modal）。

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	关闭图表类型 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、md、sm	String	md

advice	是否是运营类弹窗	Boolean	false

slots

slotName	说明
header	可选，modal 头部
footer	可选，modal 尾部

代码示例

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

```
<view>
<button onTap="openModal">打开modal</button>
<modal
show="{{modalOpened}}"
onModalClick="onModalClick"
onModalClose="onModalClose"
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>
</view>
```

```
Page({
data: {
modalOpened: false,
},
openModal() {
this.setData({
modalOpened: true,
});
},
onModalClick() {
this.setData({
modalOpened: false,
});
},
onModalClose() {
this.setData({
modalOpened: false,
});
```

```
}
});
```

10.3.4 弹出菜单（Popup）

本文介绍弹出菜单（Popup）。

属性名	描述	类型	默认值	必选
className	自定义class	String	-	false
show	是否显示菜单	Boolean	false	false
animation	是否开启动画	Boolean	true	false
mask	是否显示mask，不显示时点击外部不会触发 onClose	Boolean	true	true
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	false
disableScroll	展示mask时是否禁止页面滚动	Boolean	true	false
zIndex	定义 popup 的层级	Number	0	false

slots

可以在 popup 组件中定义要展示部分，具体可参看下面示例。

代码示例

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

```
<view>
<view class="btn-container">
<button onTap="onTopBtnTap">弹出popup</button>
</view>
<popup show="{{showTop}}"position="top"onClose="onPopupClose">
<view style="height: 200px; background: #fff; display: flex; justify-content: center; align-items: center;">hello world</view>
</popup>
</view>
```

```
Page({
```

```
data: {
showTop: false,
},
onTopBtnTap() {
this.setData({
showTop: true,
});
},
onPopupClose() {
this.setData({
showTop: false,
});
},
});
```

10.4 结果类

10.4.1 异常页（PageResult）

本文介绍异常页面（PageResult）。

属性名	描述	类型	默认值	必选
type	异常页面类型，可选，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	false
local	是否是局部异常内容	Boolean	false	false
title	错误提示标题	String		false
brief	错误提示简要	String		false

代码示例

```
{
"defaultTitle":"异常反馈",
"usingComponents": {
"page-result":"mini-antui/es/page-result/index"
}
}
```

```
<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>
```

10.4.2 结果页（Message）

本文介绍结果页（Message）。

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

代码示例

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

```
<view>
<message
title="{{title}}"
subTitle="{{subTitle}}"
type="success"
mainButton="{{messageButton.mainButton}}"
subButton="{{messageButton.subButton}}"
onTapMain="goBack">
</message>
</view>
```

```
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();
}
});
```

10.5 提示引导

10.5.1 提示（Tips）

本文介绍小提示（Tips）。分 tips-dialog 和 tips-plain 两种类型。

tips-dialog

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

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	必选
className	自定义class	String		false
time	自动关闭时间，单位毫秒	Number	5000(ms)	false
onClose	回调并关闭提示框	() => void		false

代码示例

```
{
"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"
}
}
```

tips-dialog

```
<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'
});
}
});
```

10.5.2 通告栏（Notice）

本文介绍通告栏（Notice）。

属性名	描述	类型	默认值
mode	提示可选类型：link、closable	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 }

代码示例

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

```
<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 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>


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
});
}
})
```

10.5.3 徽标（Badge）

徽标（Badge）为红点、数字或文字，用于告诉用户待处理的事物或者更新数。

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

slots

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

代码示例

```
{
"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"
}
}
```

```
<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>
```

```
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',
},
],
},
});
```

10.6 表单类

10.6.1 文本输入（InputItem）

本文介绍文本输入（InputItem）。

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	''
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	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	

slots

slotname	说明
extra	可选，用于渲染 input-item 项右边说明

代码示例

```
{
"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"
}
}
```

```
<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>


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],
});
},
});
},
});
```

```
.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;
}
```

10.6.2 选择输入（PickerItem）

本文介绍选择输入（PickerItem）。

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
extra	可选，用于渲染 picker-item 项右边说明

代码示例

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

```
<view>
<list>
<input-item
data-field="password"
placeholder="密码"
password
>
密码
</input-item>
<picker-item
data-field="bank"
placeholder="选择发卡银行"
value="{{bank}}"
onPickerTap="onSelect"
>
发卡银行
</picker-item>
</list>
</view>
```

```
const banks = ['网商银行', '建设银行', '工商银行', '浦发银行'];

Page({
data: {
bank: '',
},
onSelect() {
my.showActionSheet({
title: '选择发卡银行',
items: banks,
success: (res) => {
this.setData({
bank: banks[res.index],
```

```
});
},
});
},
});
```

10.6.3 金额输入（AmountInput）

本文介绍金额输入框（AmountInput）。

属性名	描述	类型	默认值	必选
type	input 的类型，有效值：digit、number	String	number	false
title	左上角标题	String	-	false
extra	左下角说明	String	-	false
value	输入框当前值	String	-	false
btnText	右下角按钮文案	String	-	false
placeholder	placeholder	String	-	false
focus	自动获取光标	Boolean	false	false
onInput	键盘输入时触发	(e: Object) => void	-	false
onFocus	获取焦点时触发	(e: Object) => void	-	false
onBlur	失去焦点时触发	(e: Object) => void	-	false
onConfirm	点击键盘完成时触发	(e: Object) => void	-	false
onClear	点击 clear 图标触发	() => void	-	false
onButtonClick	点击右下角按钮时触发	() => void	-	false
maxLength	最多允许输入的字符个数	Number	-	false
controlled	是否为受控组件。为 true 时，value 内容会完全受 setData 控制。	Boolean	false	false

代码示例

```
{
"defaultTitle":"小程序AntUI组件库",
"usingComponents": {
```

```
"amount-input":"mini-antui/es/amount-input/index"
}
}
```

```
<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>
```

```
Page({
data: {
value: 200,
},
onInputClear() {
this.setData({
value: '',
});
},
onInputConfirm() {
my.alert({
content: 'confirmed',
});
},
onInput(e) {
const { value } = e.detail;
this.setData({
value,
});
},
onButtonClick() {
my.alert({
content: 'button clicked',
});
},
onInputFocus() {},
onInputBlur() {},
});
```

10.6.4 搜索框（SearchBar）

本文介绍搜索框（SearchBar）。

属性名	描述	类型	默认值	必选

value	搜索框的当前值	String	-	false
placeholder	placeholder	String	-	false
focus	自动获取光标	Boolean	false	false
onInput	键盘输入时触发	(value: String) => void	-	false
onClear	点击 clear 图标触发	(val: String) => void	-	false
onFocus	获取焦点时触发	() => void	-	false
onBlur	失去焦点时触发	() => void	-	false
onCancel	点击取消时触发	() => void	-	false
onSubmit	点击键盘的 enter 时触发	(val: String) => void	-	false
disabled	设置禁用	Boolean	-	false
maxLength	最多允许输入的字符个数	Number	-	false
showCancelButton	是否一直显示取消按钮	Boolean	-	false

代码示例

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

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

```
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,
});
},
});
```

10.6.5 复选框（AMCheckBox）

本文介绍复选框（AMCheckBox）。

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

代码示例

```
{
"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"
}
}
```

```
<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.checked}}"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>


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); },
});
```

10.7 手势类

10.7.1 可滑动单元格

本文介绍可滑动单元格（SwipeAction）。

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}]	[]
onRightItemClick	点击滑动选项	({index, detail, extra, done}) => void	调用done从而使swipe-action合上
restore	还原组件到初始状态，当有多个 swipe-action 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	-

代码示例

```
{
"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"
}
}
```

```
<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>


Page({
data: {
swipeIndex: null,
list: [
{ 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,
});
},
});
```

10.8 其他

10.8.1 日历（Calendar）

本文介绍日历（Calendar）。

属性名	描述	类型	默认值	必选
type	选择类型 single: 单日 range: 日期区间	String	single	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 5 个可选值：#f5a911、#e8541e、#07a89b、#108ee9、#b5b5b5.	Array<date, tag, tagColor>	-	false
onSelect	选择区间回调	([startDate, endDate]) => void	-	false
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	-	false
onSelectHasDisableDate	选择区间包含不可用日期	(currentMonth, prevMonth) => void	-	false

代码示例

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

```
<view>
<calendar
type="single"
tagData="{{tagData}}"
```

```
onSelect="handleSelect"/>
</view>


Page({
data: {
tagData: [
{ date: '2018-05-14', tag: '还房贷', tagColor: 5 },
{ date: '2018-05-28', tag: '公积金', tagColor: 2 },
],
},
handleSelect() {},
onMonthChange() {},
});
```

10.8.2 步进器（Stepper）

步进器（Stepper）用作增加或者减少当前数值。

属性名	描述	类型	默认值	必选
min	最小值	Number	0	true
max	最大值	Number	10000	true
value	初始值	Number	10	true
step	每次改变步数，可以为小数	Number	1	false
onChange	变化时回调函数	(value: Number) => void	-	false
disabled	禁用	Boolean	false	false
readOnly	input 只读	Boolean	false	false
showNumber	是否显示数值，默认不显示	Boolean	false	false

代码示例

```
{
"defaultTitle":"Stepper",
"usingComponents":{
"stepper":"mini-antui/es/stepper/index",
"list":"mini-antui/es/list/index",
"list-item":"mini-antui/es/list/list-item/index"
}
}


<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>


Page({
data: {
value: 8,
},
callBackFn(value){
console.log(value);
},
modifyValue() {
this.setData({
value: this.data.value + 1,
});
}
});
```

10.8.3 图标（AMIcon）

本文介绍图标（AMIcon）。

属性名	描述	类型	默认值	必选
type	icon 类型，具体效果扫上面二维码预览	String	-	true
size	icon 大小，单位 px	String	-	false
color	icon 颜色，同 CSS 的 color	String	-	false

图标风格	type 有效值
基础类型	arrow-left、 arrow-up、 arrow-right、 arrow-down、cross、plus
描边	close-o、dislike-o、heart-o、help-o、like-o、location-o、info-o、success-o、wait-o、warning-o、star-o、download、

风格	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

代码示例

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

```
<view>
<am-icon type="like"size="{{24}}"color="#333"/>
</view>
```

11 API

11.1 概述

mPaaS 框架提供给开发者更多的 JSAPI 和 OpenAPI 能力，通过小程序可以为用户提供多样化便捷服务。

说明

以 my.on 开头的 API 用来监听系统事件，接收一个 callback 函数作为参数。当该事件触发时，会调用 callback 函数，该 callback 函数可以传给对应的以 my.off 开头的 API 来解除监听关系。如果直接调用 my.off 开头的 API，则为解除所有监听关系。例如：

```
Page({
onLoad() {
this.callback = this.callback.bind(this);
my.onBLECharacteristicValueChange(this.callback);
},
onUnload() {
// 页面卸载时解除监听
my.offBLECharacteristicValueChange(this.callback);
},
callback(res) {
console.log(res);
},
});
```

其他 API 都接收一个 object 作为参数。可以指定 success（调用成功）、fail（调用失败）或 complete（调用成功或失败）来接收接口调用结果。回调结果如无特殊说明，一般为一个对象，其中如果有 error/errorMessage 则表示调用失败。调用后返回值为一个 promise 对象。 例如：

```
my.httpRequest({
url: '/x.htm',
success:(res1) => {
},
}).then((res2) => {
// res1 === res2
},(res2) => {
console.log(res.error, res.errorMessage);
})
```

11.2 界面

11.2.1 导航栏

my.getTitleColor

该接口用于获取导航栏背景色。

版本要求：基础库 1.13.0 或更高版本；支付宝客户端 10.1.50 或更高版本，若版本较低，建议做 兼容处理。

代码示例

```
// API-DEMO page/API/get-title-color/get-title-color.json
{
"defaultTitle":"获取导航栏背景颜色"
}
```

```
<!-- API-DEMO page/API/get-title-color/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>
```

```
// API-DEMO page/API/get-title-color/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。

my.hideBackHome

该接口用于隐藏标题栏上的返回首页图标（如下图所示）和右上角通用菜单中的返回首页功能。

版本要求：基础库 1.16.4 或更高版本；支付宝客户端 10.1.52 或更高版本，若版本较低，建议做 兼容处理。

说明：

- 若您在启动小程序时，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。
- 若您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。


收藏
输入框
受控聚焦
input something
聚焦
显示输入
show input content
你输入的是:
最大长度
maxlength 10
收起键盘
输入 123 自动收起键
输入密码
密码输入框
输入数字
数字输入框
小数点键盘
带小数点的数字键盘
身份证键盘
身份证输入键盘
搜索框
搜索

代码示例

```
//.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)
}
```

my.hideNavigationBarLoading

该接口用于在当前页面隐藏导航条的加载动画。

代码示例

```
// 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

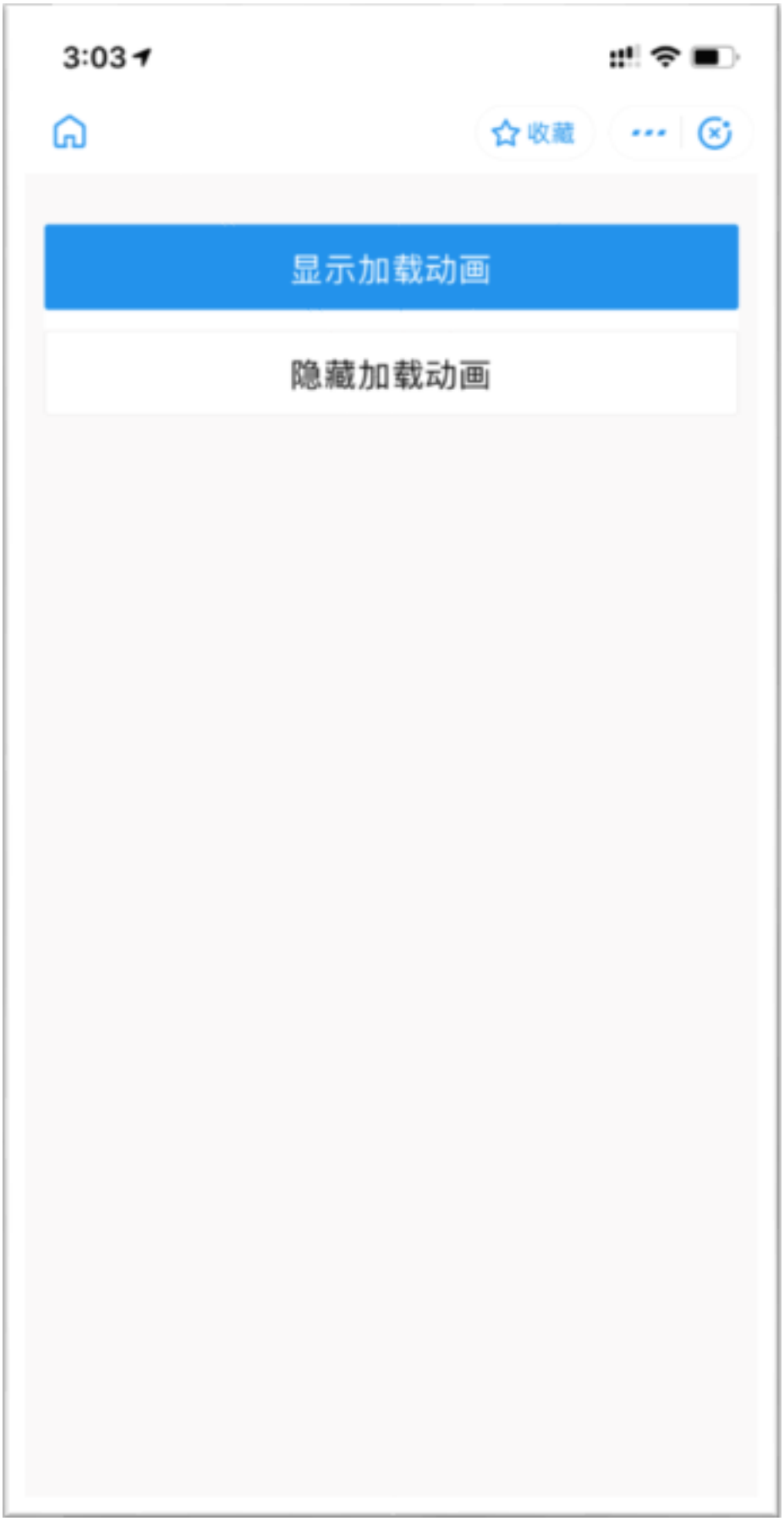
Object 类型，属性如下：

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）。

my.showNavigationBarLoading

该接口用于在当前页面显示导航条的加载动画。

效果示例


3:03
收藏
显示加载动画
隐藏加载动画

代码示例

```
// 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）。

my.setNavigationBar

该接口用于设置导航栏样式：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。

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。

效果示例

代码示例

```
// 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，borderBottomColor 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	是否重置导航栏为支付宝默认配色，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）。

相关信息

更多关于 iOS @3x 分辨率标准的信息，参见 Image Size and Resolution。

11.2.2 TabBar

my.switchTab

说明：mPaaS 10.1.32 及以上版本支持该接口。

跳转到指定 **tabBar** 页面，并关闭其他所有非 **tabBar** 页面。

如果您的小程序是一个多 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。

说明：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。另外，tabBar 的第一个页面必须是首页。

入参

参数	类型	必填	说明
url	String	是	跳转的 tabBar 页面的路径（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）。

代码示例

```
// app.json
{
"tabBar": {
"items": [{
"pagePath":"home",
"name":"首页"
},{
"pagePath":"user",
"name":"用户"
}]
}
}


my.switchTab({
url: '/home'
})
```

tabBar 配置

属性	类型	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	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。

说明：icon 推荐大小为 60 px * 60 px，系统会对任意传入的图片进行非等比拉伸或缩放。

配置示例

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

11.2.3 路由

my.switchTab

该接口用于跳转到指定标签页（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。

相关问题参见 路由 FAQ 。

效果示例


2:44
收藏
跳转新页面
返回上一页
在当前页面打开 - 获取用户信息
跳转 Tab - 组件
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。
跳转到小程序
跳回小程序

代码示例

```
// app.json
{
"tabBar": {
"items": [{
"pagePath":"page/home/index",
"name":"首页"
},{
"pagePath":"page/user/index",
"name":"用户"
}]
}
}


//.js
my.switchTab({
url: 'page/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）。

tabBar 配置

属性	类型	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	每个标签（tab）配置。

item 配置：

属性	类型	必填	描述
pagePath	String	是	设置页面路径。
name	String	是	名称。

icon	String	否	普通图标路径。推荐大小为 60*60 px ，系统会对任意传入的图片非等比拉伸或缩放。
activeIcon	String	否	高亮图标路径。

配置示例

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

my.reLaunch

该接口用于关闭当前所有页面，跳转到应用内的某个指定页面。

版本要求：基础库 1.4.0 或更高版本；支付宝客户端 10.1.8 或更高版本，若版本较低，建议做 兼容处理。

相关问题请参见 路由 FAQ 。

效果示例


2:44
收藏
跳转新页面
返回上一页
在当前页面打开 - 获取用户信息
跳转 Tab - 组件
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。
跳转到小程序
跳回小程序

代码示例

```
my.reLaunch({
url: '/page/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）。

my.redirectTo

该接口用于关闭当前页面，跳转到应用内的某个指定页面。

相关问题请参见 路由 FAQ 。

说明：使用 my.redirectTo 跳转到某个页面的同时，会关闭当前页面再跳转到下个页面，所以在页面上没有返回箭头。

效果示例


2:44
收藏
跳转新页面
返回上一页
在当前页面打开 - 获取用户信息
跳转 Tab - 组件
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。
跳转到小程序
跳回小程序

代码示例

```
my.redirectTo({
url: 'new_page?count=100'
})
```

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）。

my.navigateTo

该接口用于从当前页面，跳转到应用内的某个指定页面。

- 您可使用 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 。

效果示例


2:44
收藏
跳转新页面
返回上一页
在当前页面打开 - 获取用户信息
跳转 Tab - 组件
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。
跳转到小程序
跳回小程序

代码示例

```
// 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）。

my.navigateBack

该接口用于关闭当前页面，返回上一级或多级页面。可通过 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 。

效果示例


2:44
收藏
跳转新页面
返回上一页
在当前页面打开 - 获取用户信息
跳转 Tab - 组件
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。
跳转到小程序
跳回小程序

示例代码示例

```
// 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。

路由 FAQ

使用 my.reLaunch 跳转的页面为什么不显示底部的 tab 栏？

该 API 不允许跳转到选项卡页面。若要跳转到 tab 页面，请使用 my.swithTab 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，则不会有返回按钮。

11.2.4 交互反馈

my.alert

说明：mPaaS 10.1.32 及以上版本支持该接口。

alert 警告框。

入参

名称	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）。

代码示例

```
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」',
});
},
});
```

my.confirm

说明：mPaaS 10.1.32 及以上版本支持该接口。

confirm 确认框。

入参

名称	类型	必填	描述
title	String	否	确认框的标题。
content	String	否	确认框的内容。
confirmButtonText	String	否	确认的按钮文字，默认为 **确定**。
cancelButtonText	String	否	取消的按钮文字，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 返回值

名称	类型	描述
confirm	Boolean	点击 **确定** 返回 true，点击 **cancel** 返回 false。

代码示例

```
my.confirm({
title: '温馨提示',
content: '您是否想查询快递单号：
1234567890',
confirmButtonText: '马上查询',
cancelButtonText: '暂不需要',
success: (result) => {
my.alert({
title: `${result.confirm}`,
});
},
});
```

my.prompt

重要：基础库 1.7.2 及以上版本，mPaaS 10.1.32 及以上版本支持该接口。

弹出一个对话框，让用户在对话框内输入文本。

入参

名称	类型	必填	描述
title	String	否	prompt 框的标题。
message	Strin	是	prompt 框文本，默认为 **请输入内容**。

	g		
placeholder	String	否	输入框内的提示文案。
align	String	否	message 的对齐方式，可用枚举 left/center/right，例如 iOS ‘center’，android ‘left’ ，表示在 iOS 客户端上居中对齐，在 Android 客户端上靠左对齐。
okButtonText	String	否	确认按钮文字，默认为 **确定**。
cancelButtonText	String	否	确认按钮文字，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 返回值

名称	类型	描述
ok	Boolean	点击 **ok** 返回 true，点击 **cancel** 返回 false。
inputValue	String	当 ok 为 true 时，返回用户输入的内容。

代码示例

```
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),
});
},
});
```

my.showToast

说明：mPaaS 10.1.32 及以上版本支持该接口。

显示一个弱提示，可选择多少秒之后消失。

入参

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，默认为 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，默认为 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）。

代码示例

```
my.showToast({
type: 'success',
content: '操作成功',
duration: 3000,
success: () => {
my.alert({
title: 'toast 消失了',
});
},
});
```

my.hideToast

说明：mPaaS 10.1.32 及以上版本支持该接口。

隐藏弱提示。

代码示例

```
my.hideToast()
```

my.showLoading

说明：mPaaS 10.1.32 及以上版本支持该接口。

显示加载提示。

入参

名称	类型	必填	描述
content	String	否	加载提示的文字内容。
delay	Number	否	延迟显示，单位为 ms，默认为 0。若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）。

代码示例

```
my.showLoading({
content: '加载中...',
delay: 1000,
});
```

my.hideLoading

说明：mPaaS 10.1.32 及以上版本支持该接口。

隐藏加载提示。

入参

名称	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。

代码示例

```
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);
}
})
```

my.showNavigationBarLoading

说明：mPaaS 10.1.32 及以上版本支持该接口。

显示导航栏加载状态。

代码示例

```
my.showNavigationBarLoading()
```

如果页面的标题文本长度设置得过长，将有可能导致加载图标不显示。

my.hideNavigationBarLoading

说明：mPaaS 10.1.32 及以上版本支持该接口。

隐藏导航栏加载状态。

代码示例

```
my.hideNavigationBarLoading()
```

my.showActionSheet

说明：mPaaS 10.1.32 及以上版本支持该接口。

显示操作菜单。

入参

名称	类型	必填	描述	基础库最低版本
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：在 Android 平台上，此字段无效，不会显示取消按钮。	
destructiveBtnIndex	Number	是	（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，默认为红色。	
badges	Object Array	否	需飘红选项的数组，数组内部对象字段见下表。	1.9.0
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数组内部对象字段

名称	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： 1、飘红类型为 none/point/more 时本文案可不填。 2、飘红类型为 num 时，本文案为小数或 <=0 时，文案均不显示, 本文案 >100 时， 文案显示为 ... 。

代码示例

```
my.showActionSheet({
title: '支付宝-ActionSheet',
items: ['菜单一', '菜单二', '菜单三', '菜单四', '菜单五'],
badges: [
```

```
{ index: 0, type: 'none' },
{ index: 1, type: 'point' },
{ index: 2, type: 'num', text: '99' },
{ index: 3, type: 'text', text: '推荐' },
{ index: 4, type: 'more' },],
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`
});
},
});
```

11.2.5 下拉刷新

说明：window 中 pullRefresh 和 allowsBounceVertical 属性的设置会影响下拉刷新的使用。

属性	类型	必填	描述
pullRefresh	Boolean	否	是否允许下拉刷新。默认为 false，备注：下拉刷新生效的前提是 allowsBounceVertical 的值为 YES。
allowsBounceVertical	String	否	页面是否支持纵向拽拉超出实际内容。默认为 YES, 支持 YES / NO。

onPullDownRefresh

说明：mPaaS 10.1.32 及以上版本支持该接口。

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。

- 需要在页面对应的 .json 配置文件中配置 "pullRefresh": true 选项，才能开启下拉刷新事件。
- 当处理完数据刷新后，调用 my.stopPullDownRefresh 可以停止当前页面的下拉刷新。

代码示例

pull-down-refresh.json 配置文件中的代码配置如下：

```
{
"pullRefresh": true
}
```

Page 中定义 onPullDownRefresh 处理函数：

```
onPullDownRefresh() {
console.log('onPullDownRefresh', new Date())
}
```

my.stopPullDownRefresh

说明：mPaaS 10.1.32 及以上版本支持该接口。

停止当前页面的下拉刷新。

代码示例

```
Page({
onPullDownRefresh(){
my.stopPullDownRefresh()
}
})
```

my.startPullDownRefresh

说明：mPaaS 10.1.32 及以上版本支持该接口。

开始下拉刷新。代码执行后触发下拉刷新动画，效果与用户手动下拉刷新保持一致。

代码示例

```
my.startPullDownRefresh()
```

11.2.6 联系人

my.choosePhoneContact

选择本地系统通信录中某个联系人的电话。

入参

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 返回值

名称	类型	描述
name	String	选中的联系人姓名
mobile	String	选中的联系人手机号

错误码

error	描述
10	没有权限
11	用户取消操作（或设备未授权使用通讯录）

代码示例

```
my.choosePhoneContact({
success: (res) => {
my.alert({
content: '姓名：' + res.name + '\n号码：' + res.mobile
});
},
});
```

11.2.7 选择日期

my.datePicker

说明：mPaaS 10.1.32 及以上版本支持该接口。

打开日期选择列表。

入参

名称	类型	必填	描述
format	String	否	返回的日期格式， 1. yyyy-MM-dd（默认） 2. HH:mm 3. yyyy-MM-dd HH:mm 4. yyyy-MM（最低基础库： 1.1.1 ， 可用 **canIUse(‘datePicker.object.format.yyyy-MM’)** 判断） 5. yyyy（最低基础库： 1.1.1 ，可用 **canIUse(‘datePicker.object.format.yyyy’)** 判断）
currentDate	String	否	初始选择的日期时间，默认为当前时间。
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）。

success 返回值

名称	类型	描述
date	String	选择的日期。

错误码

error	描述
11	用户取消操作。

代码示例

```
my.datePicker({
format: 'yyyy-MM-dd',
currentDate: '2012-12-12',
startDate: '2012-12-10',
endDate: '2012-12-15',
success: (res) => {
my.alert({
content: res.date,
});
},
});
```

11.2.8 动画

my.createAnimation

说明：mPaaS 10.1.32 及以上版本支持该接口。

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。

重要：export 方法调用后会清掉之前的动画操作。

入参

参数	类型	必填	说明
duration	Integer	否	动画的持续时间，单位 ms，默认值为 400。
timingFunction	String	否	定义动画的效果，默认值为 linear，有效值为 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，默认值为 0。
transformOrigin	String	否	设置 transform-origin，默认值为 50% 50% 0。

```
const animation = my.createAnimation({
transformOrigin:"top right",
duration: 3000,
timingFunction:"ease-in-out",
delay: 100,
})
```

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。

样式

方法	参数	说明
opacity	value	透明度，参数范围为 0~1。
backgroundColor	color	颜色值。
width	length	长度值，如果传入数字则默认单位为 px，可传入其他自定义单位的长度值。
height	length	长度值，如果传入数字则默认单位为 px，可传入其他自定义单位的长度值。
top	length	同上
left	length	长度值，如果传入数字则默认单位为 px，可传入其他自定义单位的长度值。
bottom	length	长度值，如果传入数字则默认单位为 px，可传入其他自定义单位的长度值。
right	length	长度值，如果传入数字则默认单位为 px，可传入其他自定义单位的长度值。

旋转

方法	参数	说明
rotate	deg	deg 范围为 -180 ~ 180，从原点顺时针旋转一个 deg 角度。
rotateX	deg	deg 范围为 -180 ~ 180，在 X 轴旋转一个 deg 角度。
rotateY	deg	deg 范围为 -180 ~ 180，在 Y 轴旋转一个 deg 角度。
rotateZ	deg	deg 范围为 -180 ~ 180，在 Z 轴旋转一个 deg 角度。
rotate3d	(x, y , z, deg)	同 transform-function rotate3d (英文)。

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；有两个参数时表示在 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，单位为 px。
translateY	ty	在 Y 轴偏移 tx，单位为 px。
translateZ	tz	在 Z 轴偏移 tx，单位为 p。
translate3d	(tx,ty,tz)	在 X 轴偏移 tx，在 Y 轴偏移t y，在Z轴偏移 tz，单位为 px。

倾斜

方法	参数	说明
skew	ax,[ay]	参数范围为 -180 ~ 180。只有一个参数时，Y 轴坐标不变，X 轴坐标沿顺时针倾斜 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	参数范围为 -180 ~ 180。Y 轴坐标不变，X 轴坐标沿顺时针倾斜 ax 度。
skewY	ay	参数范围为 -180~180。X 轴坐标不变，Y 轴坐标沿顺时针倾斜 ay 度。

矩形变阵

方法	参数	说明
matrix	(a,b,c,d,tx,ty)	同 transform-function（英文）
matrix3d		同 transform-function matrix3d（英文）

动画队列

调用动画操作方法后需要要调用 step() 来表示一组动画完成。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。

代码示例

```
<view animation="{{animationInfo}}"style="background:yellow;height:100rpx;width:100rpx"></view>


Page({
data: {
animationInfo: {}
},
onShow(){
var animation = my.createAnimation({
duration: 1000,
timeFunction: 'ease-in-out',
});

this.animation = animation;

animation.scale(3,3).rotate(60).step();
```

```
this.setData({
animationInfo:animation.export()
});

setTimeout(function() {
animation.translate(35).step();
this.setData({
animationInfo:animation.export(),
});
}.bind(this), 1500);
},
rotateAndScale () {
// 旋转同时放大
this.animation.rotate(60).scale(3, 3).step();
this.setData({
animationInfo: this.animation.export(),
});
},
rotateThenScale () {
// 先旋转后放大
this.animation.rotate(60).step();
this.animation.scale(3, 3).step();
this.setData({
animationInfo: this.animation.export(),
});
},
rotateAndScaleThenTranslate () {
// 先旋转同时放大，然后平移
this.animation.rotate(60).scale(3, 3).step();
this.animation.translate(100, 100).step({ duration: 2000 });
this.setData({
animationInfo: this.animation.export()
});
}
})
```

11.2.9 画布

文档解析出错,请参照金融科技上面的文档，文档编号为(67567),也请把括号中的文档编号发给我们查找问题,谢谢!

11.2.10 地图

my.createMapContext(mapId)

说明：mPaaS 10.1.32 及以上版本支持该接口。

创建并返回一个地图上下文对象 mapContext。

PageContext.setData(Object)

说明：mPaaS 10.1.32 及以上版本支持本接口。

初始化或重置地图数据，参数可选。

```
this.setData({
scale: 14,
longitude: 120.131441,
latitude: 30.279383,
'show-location':true,
// 地图贴图 10.1.35新增
'ground-overlays':[{
'include-points':[{// 右上
latitude: 39.935029,
longitude: 116.384377,
},{// 左下
latitude: 39.939577,
longitude: 116.388331,
}],
image:'/image/groundoverlay.png',
alpha:0.75,
zIndex:0,
}],
// 网格贴图 10.1.35新增
'tile-overlay':{
url:'http://xixi.fullspeed.cn/public/map',
type:0,
tileWidth:256,
tileHeight:256,
zIndex:1,
},
markers:[{},{}],
include-points:[{},{}],
// 10.1.35新增 全览逻辑
include-padding:{left:0, right:0, top:0, bottom:0},
polyline: [{},{}],
circles: [{},{}],
controls: [{},{}],
polygon: [{},{}],
'include-padding':{},
// 初始化支持地图设置 10.1.50新增
setting:{
// 手势
gestureEnable:0/1,
// 比例尺
showScale:0/1,
// 指南针
showCompass:0/1,
// 双手下滑
tiltGesturesEnabled:0/1,
// 交通路况展示
trafficEnabled:0/1,
// 地图POI信息
showMapText:0/1,
// 高德地图logo位置
logoPosition:{centerX:150, centerY:90},
},
});
```

MapContext.moveToLocation()

说明：mPaaS 10.1.32 及以上版本支持本接口。

移动视野到定位点并恢复到默认缩放级别，需要配合地图组件的 show-location 使用。

```
this.mapCtx.moveToLocation();
```

MapContext.gestureEnable(Object)

说明：mPaaS 10.1.32 及以上版本支持本接口。

设置所有手势是否可用，1：可用 ，0：不可用。

```
this.mapCtx.gestureEnable({isGestureEnable:1});
```

MapContext.showsScale(Object)

说明：mPaaS 10.1.32 及以上版本支持本接口。

设置比例尺控件是否可见，1：可见 ，0：不可见。

```
this.mapCtx.showsScale({isShowsScale:1});
```

MapContext.showsCompass(Object)

说明：mPaaS 10.1.32 及以上版本支持本接口。

设置指南针是否可见，1：可见 ，0：不可见。

```
this.mapCtx.showsCompass({isShowsCompass:1});
```

MapContext.showRoute(Object)

说明：mPaaS 10.1.32 及以上版本支持本接口。

默认规划步行路线，只能显示一条。
mPaaS 10.1.60 以上版本可以规划步行、公交、骑行和驾车四种路线。

```
this.mapCtx.showRoute({
searchType:"bus", // 搜索类型：10.1.60新增，有"walk","bus","drive","ride", 默认值为walk
startLat:30.257839, // 起点纬度
startLng:120.062726, // 起点经度
endLat:30.256718, // 终点纬度
endLng:120.059985, // 终点经度
throughPoints:[{lat: 39.866958,lng:116.494231},{lat: 39.9357,lng:116.581092}],//途径点：10.1.60新增,仅驾车规划有效，searchType=“drive”
routeColor:'#FFB90F', // 路线颜色 10.1.60之后，该值仅在2d地图中生效
iconPath:"/image/texture.png", // 路线纹理 10.1.32 3d地图其优先级高于routeColor，即纹理会覆盖颜色值；10.1.60建议不再设置，在3d地图下提供了默认的纹理图案。
iconWidth:10, // 纹理宽度 10.1.32 iconPath设置时才生效。10.1.60建议不再设置，在3d地图下提供了默认的纹理宽度。
```

```
routeWidth:10, // 路线宽度 在不设置纹理时有效。 10.1.60建议不再设置，在2d地图下提供了默认值，3d不需要设置。
zIndex:4 // 覆盖物 Z 轴坐标 10.1.32
mode:0 // 只有驾车模式和公交模式支持，可选,具体值见下表
city:'hangzhou', // 公交模式下必填
destinationCity:'hangzhou', // 公交跨城模式下必填
});
```

mode	bus	drive
0	最快捷模式	速度优先（时间）。
1	最经济模式	费用优先（不走收费路段的最快道路）。
2	最少换乘模式	距离优先。
3	最少步行模式	不走快速路。
4	最舒适模式	结合实时交通（躲避拥堵）。
5	不乘地铁模式	多策略（同时使用速度优先、费用优先、距离优先三个策略）。
6	无	不走高速。
7	无	不走高速且避免收费。
8	无	躲避收费和拥堵。
0	无	不走高速且躲避收费和拥堵。

MapContext.clearRoute()

说明：mPaaS 10.1.32 及以上版本支持本接口。

清除地图上的步行导航路线。

```
this.mapCtx.clearRoute();
```

MapContext.getCenterLocation(Callback)

说明：mPaaS 10.1.32 及以上版本支持本接口。

获取当前地图中心位置。

```
this.mapCtx.getCenterLocation({
success: res => {
my.alert({
content: 'longitude:' + res.longitude + '\nlatitude:' + res.latitude,
});
console.log(res.longitude);
console.log(res.latitude);
}
});
```

MapContext.updateComponents(Object)

说明：mPaaS 10.1.60 及以上版本支持本接口。

增量更新地图接口。

```
this.mapCtx.updateComponents({
scale: 14,
longitude: 120.131441,
latitude: 30.279383,
command:{
// marker动画
markerAnim:[
{
type:0 // 跳动动画 10.1.35
markerId:xxx,
}
],
},
setting:{
// 手势
gestureEnable:0/1,
// 比例尺
showScale:0/1,
// 指南针
showCompass:0/1,
// 双手下滑
tiltGesturesEnabled:0/1,
// 交通路况展示
trafficEnabled:0/1,
// 地图POI信息
showMapText:0/1,
// 高德地图logo位置
logoPosition:{centerX:150, centerY:90},
},
markers:[{},{}],
polyline:[{},{}],
include-points:[{},{}],
include-padding:{left:0, right:0, top:0, bottom:0},
});
```

MapContext.translateMarker(Object)

说明：mPaaS 10.1.60 及以上版本支持本接口。

平移 marker 接口。

对同一个 markerId, 在 translateMarker 没 animationEnd 之前，再次调用会被丢掉，下一次动画需要在 animationEnd 之后再调用才有效。

```
this.mapCtx.translateMarker({
markerId:xxx, // 必填
destination:{
longitude:xxx,latitude:xxx // 必填
},
autoRotate:true/false, // 选填，默认true
rotate:xxx, // 选填，在autoRotate为false的情况下才有效，不填默认是0
duration:1000, // 选填，单位ms，默认1000ms
animationEnd:xxx //function 动画结束回调
});
```

Marker 图鉴

说明：mPaaS 10.1.60 及以上版本支持本功能。

Marker 样式优先级说明

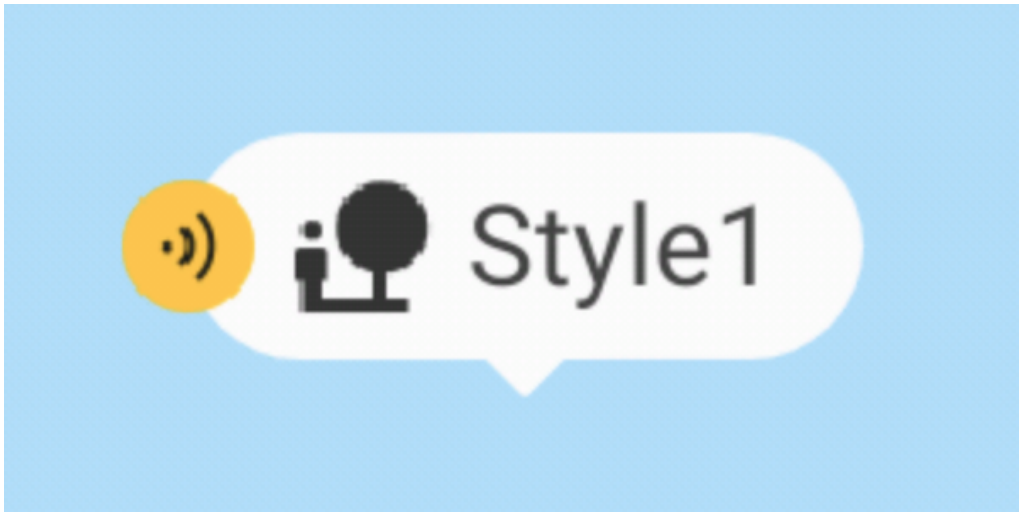
customcallout、callout 和 label 互斥，优先级：label > customcallout > callout。

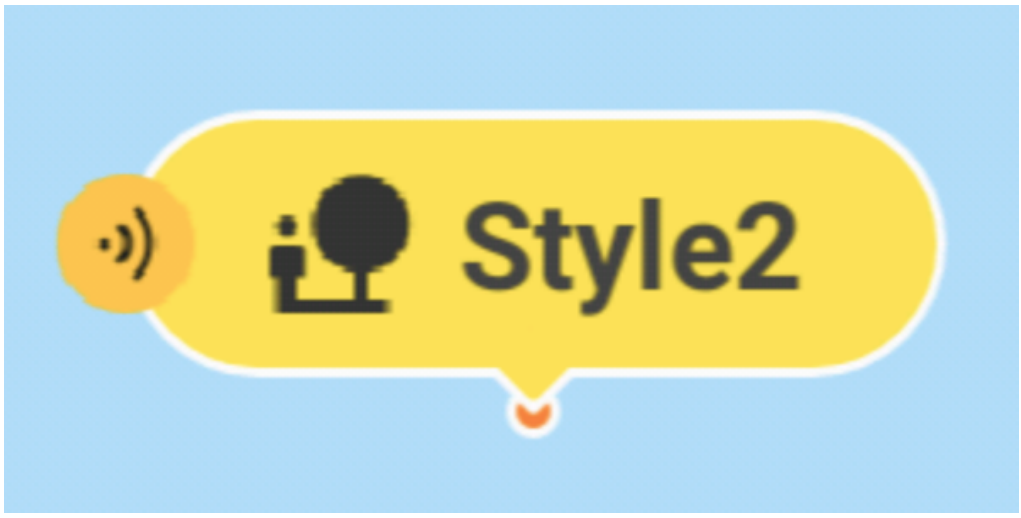
style 与 icon 互斥，优先级：

- style > iconAppendStr。
- style > icon。

style

结构	图鉴

{
type:1,
text1:" Style1 ",
icon1:' xxx' ,
icon2:' xxx'
}
	

{
type:2,
text1:" Style2 ",
icon1:' xxx' ,
icon2:' xxx'
}
	

{ type:3, icon:xxx, //选填 text:xxx, //必填 color:xxx, //默认#33B276	

bgColor:xxx, //默认#FFFFFF gravity:" left/ center/right " , //默认 center fontType:" s mall/standard /large" //默 认standard }	

customcallout

结构	图鉴
{ "ty pe " : 0, "ti me " : "3 " , "d esc List " : [{ "d esc " : "点 击立 即打 车 " , "d esc Col or " : "#f fffff " }], "is Sho w" : 1 }	
{ "ty pe " : 1,	

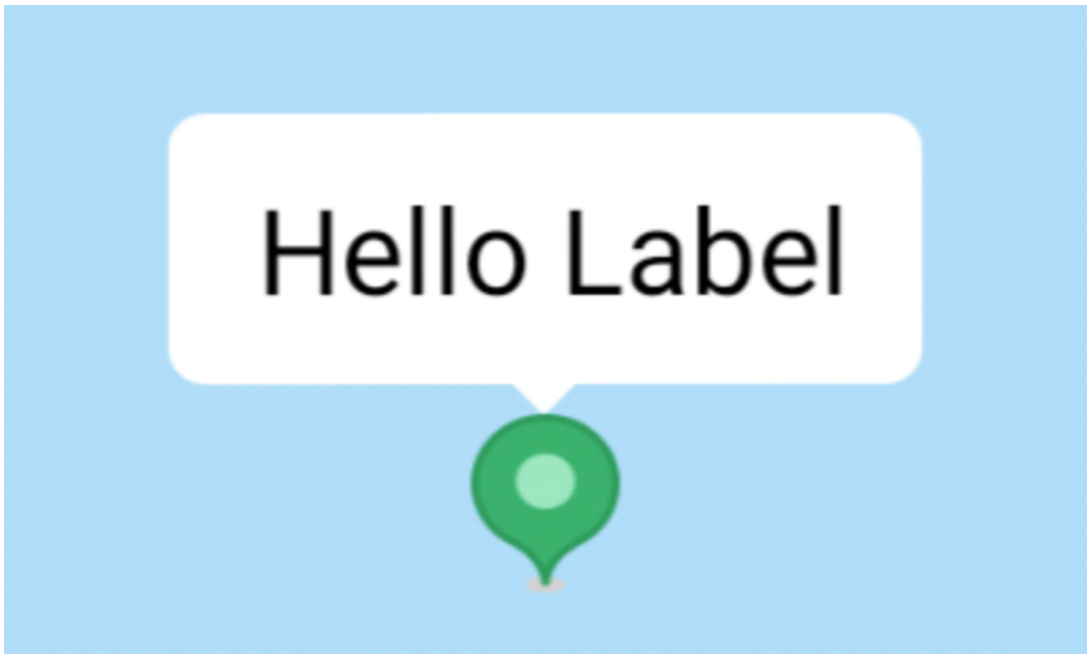
“time”: “3”, “descList”: [{ “desc”: “点击立即打车”, “descColor”: “#333333” }], “isShow”: 1 }	
{ “type”: 2, “descList”: [{ “desc”: “预计”, “descColor”: “#333333” }, { “desc”: “5分钟”, “descColor”: “#	

108
EE9
”
}, {
“d
esc
” :
“到
达
”,
“d
esc
Col
or
” :
“#
333
333
”
}],
“is
Sho
w”
: 1
}	

label

- content：必填
- color：选填，默认为 #000000
- fontsize：选填，默认为 14
- borderRadius：选填，默认为 20
- bgColor：选填，默认为 #FFFFFF
- padding：选填，默认为 10

结构	图鉴

{
conten
t:” Hel
lo
Label
”,
color:
” #000
000”,
fontSiz
e:16,
border
Radius:
5,
bgCol
or:” #f
fffff”,
paddin
g:12,
}
	

11.2.11 键盘

my.hideKeyboard

说明：mPaaS 10.1.32 及以上版本支持该接口。
隐藏键盘。

代码示例

```
my.hideKeyboard();
```

11.2.12 滚动

my.pageScrollTo

说明：mPaaS 10.1.32 及以上版本支持该接口。
滚动到页面的目标位置。

参数说明

参数名	类型	说明
scrollTop	Number	滚动到页面的目标位置，单位为 px。

代码示例

```
my.pageScrollTo({
scrollTop: 100
})
```

11.2.13 节点查询

my.createSelectorQuery

说明：基础库 1.4.0 及以上版本，mPaaS 10.1.32 及以上版本支持该接口。
获取一个节点查询对象 SelectorQuery。

参数说明

参数名	类型	说明
params	object	可以指定 page 属性，默认为当前页面。

SelectorQuery

节点查询对象类，包含以下方法：

selectorQuery.select(selector)

选择当前第一个匹配选择器的节点，选择器支持 ID 选择器以及 class 选择器。

selectorQuery.selectAll(selector)

选择所有匹配选择器的节点，选择器支持 ID 选择器以及 class 选择器。

selectorQuery.selectViewport()

选择窗口对象。

selectorQuery.boundingClientRect()

将当前选择节点的位置信息放入查询结果，类似 dom 的 getBoundingClientRect， 返回对象包含 width、height、left、top、bottom、right。如果当前节点为窗口对象，则只返回 width、height。

selectorQuery.scrollOffset()

将当前选择节点的滚动信息放入查询结果，返回对象包含 scrollTop、scrollLeft。

selectorQuery.exec(callback)

将查询结果放入 callback 回调中。查询结果为数组，每项为一次查询的结果，如果当前是节点列表，则单次查询结果也为数组。注意，exec 必须放到 Page onReady 后调用。

代码示例

```
<view className="all">节点 all1</view>

<view className="all">节点 all2</view>

<view id="one">节点 one</view>

<view id="scroll"style="height:200px;overflow: auto">
<view style="height:400px">独立滚动区域</view>
</view>


Page({
onReady() {
my.createSelectorQuery()
.select('#non-exists').boundingClientRect()
.select('#one').boundingClientRect()
.selectAll('.all').boundingClientRect()
.select('#scroll').scrollOffset()
.selectViewport().boundingClientRect()
.selectViewport().scrollOffset().exec((ret) => {
console.log(JSON.stringify(ret, null, 2));
});
},
});
```

结果 ret：

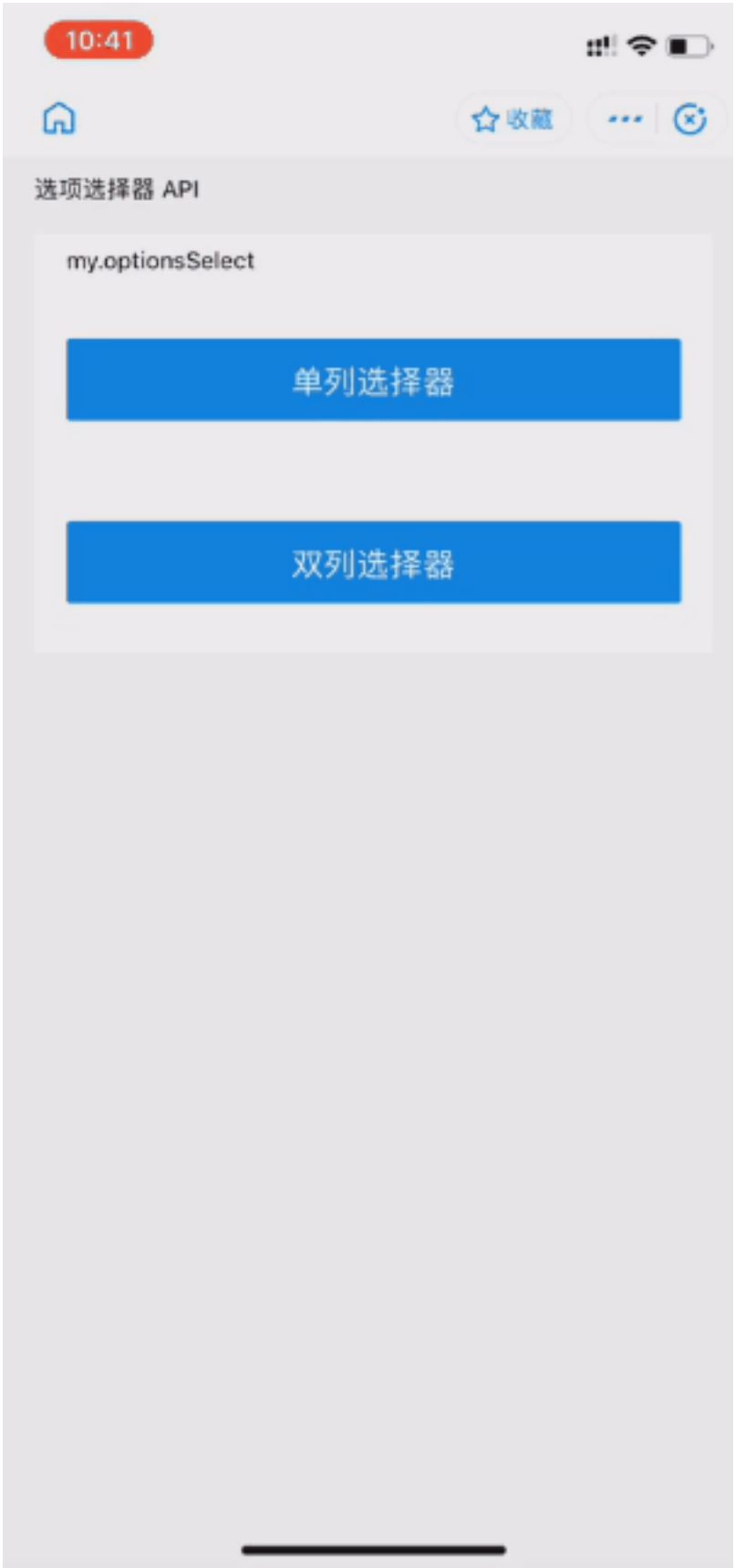
```
[
null,
{
"x": 1,
"y": 2,
"width": 1367,
"height": 18,
"top": 2,
"right": 1368,
"bottom": 20,
"left": 1
},
[
{
"x": 1,
"y": -34,
"width": 1367,
"height": 18,
"top": -34,
"right": 1368,
"bottom": -16,
"left": 1
},
{
"x": 1,
"y": -16,
"width": 1367,
"height": 18,
"top": -16,
"right": 1368,
"bottom": 2,
"left": 1
}
],
{
"scrollTop": 0,
"scrollLeft": 0
},
{
"width": 1384,
"height": 360
},
{
"scrollTop": 35,
"scrollLeft": 0
}
]
```


11.2.14 选项选择器

my.optionsSelect

类似于 safari 原生 select 的组件，但是功能更加强大，一般用来替代 select，或者 2 级数据的选择。注意不支持 2 级数据之间的联动。

效果示例



代码示例

```
// API-DEMO page/API/options-select/options-select.json
{
"defaultTitle":"选项选择器"
}


<!-- API-DEMO page/API/options-select/options-select.axml-->
<view class="page">
<view class="page-description">选项选择器 API</view>
<view class="page-section">
<view class="page-section-title">my.optionsSelect</view>
<view class="page-section-demo">
<button type="primary"onTap="openOne">单列选择器</button>
</view>
<view class="page-section-demo">
<button type="primary"onTap="openTwo">双列选择器</button>
</view>
</view>
</view>


// API-DEMO page/API/options-select/options-select.js
Page({
openOne() {
my.optionsSelect({
title:"还款日选择",
optionsOne: ["每周一","每周二","每周三","每周四","每周五","每周六","每周日"],
selectedOneIndex: 2,
success(res) {
my.alert({
content: JSON.stringify(res, null, 2),
});
}
});
},
openTwo() {
my.optionsSelect({
title:"出生年月选择",
optionsOne: ["2014年","2013年","2012年","2011年","2010年","2009年","2008年"],
optionsTwo: ["一月", '二月', '三月', '四月', '五月', '六月', '七月', '八月', '九月', '十月', '十一月', '十二月'],
selectedOneIndex: 3,
selectedTwoIndex: 5,
success(res) {
my.alert({
content: JSON.stringify(res, null, 2),
});
}
});
},
});
```

入参

入参为 Object 类型，属性如下：

名称	类型	描述	必填	默认值
title	string	头部标题信息	否	
optionsOne	string[]	选项一列表	是	
optionsTwo	string[]	选项二列表	否	
selectedOneIndex	number	选项一默认选中	否	0
selectedTwoIndex	number	选项二默认选中	否	0
positiveString	string	确定按钮文案	否	确定
negativeString	string	取消按钮文档	否	取消

success 回调函数

入参为 Object 类型，属性如下：

名称	类型	描述	备注
selectedOneIndex	number	选项一选择的值	若选择取消，返回空字符串
selectedOneOption	string	选项一选择的内容	若选择取消，返回空字符串
selectedTwoIndex	number	选项二选择的值	若选择取消，返回空字符串
selectedTwoOption	string	选项二选择的内容	若选择取消，返回空字符串

11.2.15 级联选择

my.multiLevelSelect(Object)

说明：mPaaS 10.1.32 及以上版本支持该接口。

级联选择功能主要用于多级关联数据选择的业务场景，比如说省市区的信息选择。

入参说明

名称	类型	必填	描述
title	String	否	标题
list	Object Array	是	选择数据列表
name	String	是	条目名称
subList	JsonArray	否	子条目列表
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）

出参说明

名称	类型	描述

success	Boolean	是否选择完成，取消则返回 false
result	String Array	选择的结果，如 [{“name”:”杭州市”},{“name”:”上城区”},{“name”:”古翠街道”}]

代码示例

```
my.multiLevelSelect({
title: 'nihao',//级联选择标题
list: [
{
name:"杭州市",//条目名称
subList: [
{
name:"西湖区",
subList: [
{
name:"古翠街道"
},
{
name:"文新街道"
}
]
},
{
name:"上城区",
subList: [
{
name:"延安街道"
},
{
name:"龙翔桥街道"
}
]
}
]//级联子数据列表
}
]//级联数据列表
}),
```

11.2.16 设置背景窗口

my.setBackgroundColor

说明：mPaaS 10.1.32 及以上版本支持该接口。

动态设置窗口的背景色。

入参

名称	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）

代码示例

```
/*设置窗口背景色*/
my.setBackgroundColor({
backgroundColor: '#ffffff'
})

/*分别设置顶部窗口和底部窗口背景色*/
my.setBackgroundColor({
backgroundColorTop: '#ffffff',
backgroundColorBottom: '#ffffff'
})
```

my.setBackgroundTextStyle

说明：mPaaS 10.1.32 及以上版本支持该接口。

动态设置下拉背景的字体、加载图形的样式。

入参

名称	类型	必填	描述
textStyle	String	是	下拉背景的字体、加载图形的样式，仅支持 dark和 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）

代码示例

```
my.setBackgroundTextStyle({
textStyle: 'dark', // 下拉背景字体、loading 图的样式为dark
})
```

11.2.17 设置页面是否支持下拉

my.setCanPullDown

说明：mPaaS 10.1.32 及以上版本支持该接口。

设置页面是否支持下拉（小程序内页面默认支持下拉）。

入参

参数	类型	必填	说明

canPullDown	Boolean	是	是否支持下拉

示例代码

```
my.setCanPullDown({
canPullDown:true
})
```

11.2.18 设置 optionMenu

my.setOptionMenu

说明：mPaaS 10.1.32 及以上版本支持该接口。

配置 optionMenu 导航栏的额外图标，点击后触发 onOptionMenuClick。

入参

名称	类型	必填	描述
icon	String	是	自定义 optionMenu 所用图标的 url（以 https/http 开头）或 base64 字符串，大小建议 30*30。(暂不支持 base64 图片)

代码示例

```
my.setOptionMenu({
icon: 'https://img.alicdn.com/tps/i3/T1OjaVFl4dXXa.JOZB-114-114.png',
});
```

11.3 多媒体

11.3.1 图片

my.chooseImage

说明：mPaaS 10.1.32 及以上版本支持该接口。

拍照或从手机相册中选择图片。

入参

名称	类型	必填	描述
count	Number	否	最大可选照片数，默认为 1 张
sizeType	StringArray	否	original 原图，compressed 压缩图，默认二者都有
sourceType	String Array	否	相册选取或者拍照，默认为 [‘camera’，‘album’]
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 返回值

名称	类型	描述
apFilePaths	String Array	图片文件描述

错误码

error	描述
11	用户取消操作

代码示例

```
my.chooseImage({
count: 2,
success: (res) => {
img.src = res.apFilePaths[0];
},
});
```

my.previewImage

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于预览图片。暂不支持浏览本地图片。

入参

名称	类型	必填	描述
urls	Array	是	要预览的图片链接列表
current	Number	否	当前显示图片索引，默认为 0
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）

代码示例

```
my.previewImage({
current: 2,
urls: [
'https://img.alicdn.com/tps/TB1sXGYIFXXXXc5XpXXXXXXXXXX.jpg',
'https://img.alicdn.com/tps/TB1pfG4IFXXXXc6XXXXXXXXXXXX.jpg',
'https://img.alicdn.com/tps/TB1h9xxIFXXXXbKXXXXXXXXXXXX.jpg'
],
});
```

my.saveImage

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于将在线图片保存至手机相册。

入参

名称	类型	必填	描述
url	String	是	要保存的图片链接
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）

错误码

error	描述
2	参数无效，没有传 url 参数
15	没有开启相册权限（iOS only）
16	手机相册存储空间不足（iOS only）
17	保存图片过程中的其他错误

代码示例

```
my.saveImage({url:'https://img.alicdn.com/tps/TB1sXGYIFXXXXc5XpXXXXXXXXXX.jpg'});
```

my.compressImage

说明：基础库 1.14.1 及以上版本支持该接口，低版本需做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.60 及以上版本支持该接口。

此接口用于压缩图片。

入参

名称	类型	必填	描述
apFilePaths	String Array	是	要压缩的图片地址数组
compressLevel	Number	否	压缩级别，支持 0 ~ 4 的整数，默认为 4。详见 compressLevel 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）

success 返回值

名称	类型	描述

apFilePaths	String Array	压缩后的路径数组

compressLevel 表

compressLevel	说明
0	低质量
1	中等质量
2	高质量
3	不压缩
4	根据网络适应

代码示例

```
my.compressImage({
apFilePaths:['https://resource/apmlcc0ed184daffc5a0d8da86b2f518cf7b.image'],
compressLevel:1,
success:(res)=>{
console.log(JSON.stringify(res))
}
})
```

my.getImageInfo

说明：基础库 1.4.0 及以上版本支持本接口，低版本需做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.32 及以上版本支持该接口。

此接口用于获取图片信息。

入参

名称	类型	必填	描述
src	String	否	图片路径，目前支持：
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）

success 返回值

名称	类型	描述
width	Number	图片宽度（单位为 px）
height	Number	图片高度（单位为 px）
path	String	图片本地路径
orientation	String	返回图片的方向，有效值见下表

type	String	返回图片的格式

orientation 参数说明

枚举值	说明
up	默认值
down	180 度旋转
left	逆时针旋转 90 度
right	顺时针旋转 90 度
up-mirrored	同 up，但水平翻转
down-mirrored	同 down，但水平翻转
left-mirrored	同 left，但垂直翻转
right-mirrored	同 right，但垂直翻转

代码示例

```
//网络图片路径
my.getImageInfo({
src:'https://img.alicdn.com/tps/TB1sXGYIFXXXXc5XpXXXXXXXXXX.jpg',
success:(res)=>{
console.log(JSON.stringify(res))
}
})

//apFilePath
my.chooseImage({
success: (res) => {
my.getImageInfo({
src:res.apFilePaths[0],
success:(res)=>{
console.log(JSON.stringify(res))
}
})
},
})

//相对路径
my.getImageInfo({
src:'image/api.png',
success:(res)=>{
console.log(JSON.stringify(res))
}
})
```

11.4 缓存

my.setStorage

将数据存储在本地缓存中指定的 key 中，会覆盖原来该 key 对应的数据。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是异步接口。
- 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。

入参

名称	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）

代码示例

```
my.setStorage({
key: 'currentCity',
data: {
cityName: '杭州',
adCode: '330100',
spell: ' hangzhou',
},
success: function() {
my.alert({content: '写入成功'});
}
});
```

说明：单条数据转换成字符串后，字符串长度最大200*1024。同一个客户端用户，同一个小程序缓存总上限为10 MB。

my.setStorageSync

同步将数据存储在本地缓存中指定的 key 中。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是同步接口。

入参

名称	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

代码示例

```
my.setStorageSync({
key: 'currentCity',
data: {
cityName: '杭州',
adCode: '330100',
spell: ' hangzhou',
}
});
```

my.getStorage

获取缓存数据。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是异步接口。
- 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。

入参

名称	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 返回值

名称	类型	说明
data	Object/String	key 对应的内容

代码示例

```
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});
}
});
```

my.getStorageSync

同步获取缓存数据。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是同步接口。

入参

名称	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

代码示例

```
let res = my.getStorageSync({ key: 'currentCity' });
my.alert({
content: JSON.stringify(res.data),
});
```

my.removeStorage

删除缓存数据。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是异步接口。
- 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。

入参

名称	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）

代码示例

```
my.removeStorage({
key: 'currentCity',
success: function(){
my.alert({content: '删除成功'});
```

```
}
});
```

my.removeStorageSync

删除缓存数据。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是同步接口。

入参

名称	类型	必填	描述
key	String	是	缓存数据的 key

代码示例

```
my.removeStorageSync({
key: 'currentCity',
});
```

my.clearStorage

清除本地数据缓存。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是异步接口。
- 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。

代码示例

```
my.clearStorage()
```

my.clearStorageSync

清除本地数据缓存。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是同步接口。

代码示例

```
my.clearStorageSync()
```

my.getStorageInfo

异步获取当前 storage 的相关信息。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是异步接口。
- 在内嵌 webview 内获取当前 storage 的相关信息不会获取到当前小程序 storage 的相关信息。

入参

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 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, 单位为 KB
limitSize	Number	限制的空间大小，单位为 KB

代码示例

```
my.getStorageInfo({
success: function(res) {
console.log(res.keys)
console.log(res.currentSize)
console.log(res.limitSize)
}
})
```

my.getStorageInfoSync

同步获取当前 storage 的相关信息。

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 这是同步接口。

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, 单位为 KB

limitSize	Number	限制的空间大小，单位为 KB

代码示例

```
var res = my.getStorageInfoSync()
console.log(res.keys)
console.log(res.currentSize)
console.log(res.limitSize)
```

Tips

- 缓存数据本地加密存储，通过 API 读取时会自动解密返回。
- 覆盖安装应用(不是先删除再安装)，不会导致小程序缓存失效。
- 应用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应用账号和小程序 ID 两级隔离。
- iOS 客户端支持 iTunes 备份。

11.5 文件

my.saveFile

说明：基础库 1.3.0 及以上版本支持该接口，低版本需做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

该接口用于保存文件到本地。（本地文件大小总容量限制：10M）

入参

名称	类型	必填	描述
apFilePath	String	是	文件路径
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 返回值说明

名称	类型	描述
apFilePath	String	文件保存路径

代码示例

```
my.chooseImage({
success: (res) => {
my.saveFile({
```

```
apFilePath: res.apFilePaths[0],
success: (res) => {
console.log(JSON.stringify(res))
},
});
},
});
```

my.getFileInfo

说明：基础库 1.4.0 及以上版本支持该接口，低版本需做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

入参说明

名称	类型	必填	描述
apFilePath	String	是	文件路径
digestAlgorithm	String	否	摘要算法，支持 md5 和 sha1 ，默认为 md5
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 返回值说明

名称	类型	描述
size	Number	文件大小
digest	String	摘要结果

代码示例

```
my.getFileInfo({
apFilePath:'https://resource/apml953bb093ebd2834530196f50a4413a87.video',
digestAlgorithm:'sha1',
success:(res)=>{
console.log(JSON.stringify(res))
}
})
```

my.getSavedFileInfo

说明：基础库 1.3.0 及以上版本支持该接口，低版本需做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

该接口用于获取保存的文件信息。

入参

名称	类型	必填	描述
apFilePath	String	是	文件路径

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 返回值说明

名称	类型	描述
size	Number	文件大小
createTime	Number	创建时间

代码示例

```
my.getSavedFileInfo({
apFilePath:'https://resource/apml953bb093ebd2834530196f50a4413a87.video',
success:(res)=>{
console.log(JSON.stringify(res))
}
})
```

my.getSavedFileList

说明：基础库 1.3.0 及以上版本支持该接口，低版本需做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

该接口用于获取保存的所有文件。

入参

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 返回值说明

名称	类型	描述
fileList	List	文件列表

File 对象属性说明

名称	类型	描述
size	Number	文件大小
createTime	Number	创建时间
apFilePath	String	文件路径

代码示例

```
my.getSavedFileList({
success:(res)=>{
console.log(JSON.stringfy(res))
}
});
```

my.removeSavedFile

说明：基础库 1.3.0 及以上版本支持该接口，低版本需做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

该接口用于将删除某个保存的文件。

入参

名称	类型	必填	描述
apFilePath	String	是	文件路径
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）

代码示例

```
my.getSavedFileList({
success:(res)=>{
my.removeSavedFile({
apFilePath:res.fileList[0].apFilePath,
success:(res)=>{
console.log('remove success')
}
})
}
});
```

11.6 位置

my.chooseLocation

说明：暂无国外地图数据，在中国以外的地区可能无法正常调用此 API。

该接口用于使用内置地图选择地理位置。

效果示例

代码示例

```
// API-DEMO page/API/choose-location/choose-location.json
{
"defaultTitle":"选择位置"
}


<!-- API-DEMO page/API/choose-location/choose-location.axml-->
```

```
<view class="page">
<view class="page-section">
<view class="page-section-demo">
<text>经度:</text>
<input value="{{longitude}}"></input>
</view>
<view class="page-section-demo">
<text>纬度:</text>
<input value="{{latitude}}"></input>
</view>
<view class="page-section-demo">
<text>位置名称:</text>
<input value="{{name}}"></input>
</view>
<view class="page-section-demo">
<text>详细位置:</text>
<input value="{{address}}"></input>
</view>
<view class="page-section-btns">
<view onTap="chooseLocation">选择位置</view>
</view>
</view>
</view>
```

```
// API-DEMO page/API/choose-location/choose-location.js
Page({
data: {
longitude: '120.126293',
latitude: '30.274653',
name: '黄龙万科中心',
address: '学院路77号',
},
chooseLocation() {
var that = this
my.chooseLocation({
success:(res)=>{
console.log(JSON.stringify(res))
that.setData({
longitude:res.longitude,
latitude:res.latitude,
name:res.name,
address:res.address
})
},
fail:(error)=>{
my.alert({content: '调用失败：'+JSON.stringify(error), });
},
});
},
})
```

```
/* API-DEMO page/API/choose-location/choose-location.acss */
.page-body-info {
```

```
height: 250rpx;
}
.page-body-text-location {
display: flex;
font-size: 50rpx;
}
.page-body-text-location text {
margin: 10rpx;
}
.page-section-location-text{
color: #49a9ee;
}
```

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		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
name	String	位置名称。
address	String	详细地址。
latitude	Number	纬度，浮点数，范围为-90~90，负数表示南纬。
longitude	Number	经度，浮点数，范围为-180~180，负数表示西经。

my.getLocation(OBJECT)

该接口用于获取用户当前的地理位置信息。

入参

名称	类型	必填	描述	最低版本
cacheTimeout	Number	否	mPaaS 客户端经纬度定位缓存过期时间，单位为秒。默认为30s。使用缓存会加快定位速度，缓存过期会重新定位。	
type	Number	否	0：默认，获取经纬度 1：获取经纬度和详细到区县级别的逆地理编码数据 2：获取经纬度和详细到街道级别的逆地	1.1.1

			理编码数据，不推荐使用 3：获取经纬度和详细到 POI 级别的逆地理编码数据，不推荐使用	
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 返回值

名称	类型	描述	最低版本
longitude	String	经度	
latitude	String	纬度	
accuracy	String	精确度，单位为 m	
horizontalAccuracy	String	水平精确度，单位为 m	
country	String	国家（type>0 生效）	1.1.1
countryCode	String	国家编号（type>0 生效）	1.1.1
province	String	省份（type>0 生效）	1.1.1
city	String	城市（type>0 生效）	1.1.1
cityAdcode	String	城市级别的地区代码（type>0 生效）	1.1.1
district	String	区县（type>0 生效）	1.1.1
districtAdcode	String	区县级别的地区代码（type>0 生效）	1.1.1
streetNumber	Object	需要街道级别逆地理编码数据时，才会返回该字段。街道门牌信息，结构是：{street, number}（type>1 生效）	1.1.1
pois	array	需要 POI 级别逆地理编码数据时，才会返回该字段。定位点附近的 POI 信息，结构是：{name, address}（type>2 生效）	1.1.1

错误码

错误码	描述	解决方案
11	请确认定位相关权限已开启	提示用户打开定位权限
12	网络异常，请稍后再试	提示用户检查当前网络
13	定位失败，请稍后再试	
14	业务定位超时	提示用户再次尝试

代码示例

```
my.getLocation({
success(res) {
my.hideLoading();
console.log(res)
that.setData({
hasLocation: true,
location: formatLocation(res.longitude, res.latitude)
})
},
fail() {
my.hideLoading();
my.alert({ title: '定位失败' });
},
})
```

my.openLocation

该接口用于使用 mPaaS 小程序内置地图查看位置。

入参

名称	类型	必填	描述
longitude	String	是	经度
latitude	String	是	纬度
name	String	是	位置名称
address	String	是	地址的详细说明
scale	Number	否	缩放比例，范围 3~19，默认为 15
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）

代码示例

```
my.openLocation({
longitude: '121.549697',
latitude: '31.227250',
name: '支付宝',
address: '杨高路地铁站',
});
```

11.7 网络

my.request

小程序网络请求。

说明：

- 基础库 1.11.0 及以上版本支持该接口 可以使用 my.canIUse('request') 做兼容性处理。详见 小程序基础库说明 。
- mPaaS 10.1.60 及以上版本支持该接口。
- 接口 my.httpRequest 将被废弃，请使用 my.request 来代替。
- my.request 的请求头默认值为 **{'content-type': 'application/json'}**，而不是 {'content-type': 'application/x-www-form-urlencoded'}。

入参

名称	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，默认为 {'content-type': 'application/json'}
method	String	否	默认为 GET，目前支持 GET/POST
data	Object	否	data 参数说明
timeout	Number	否	超时时间，单位为 ms，默认为 30000
dataType	String	否	期望返回的数据格式，默认为 json，支持 json、text 和 base64
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，如果数据不是 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 返回值

名称	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

error	描述
11	无权跨域
12	网络出错
13	超时
14	解码失败
19	HTTP 错误
20	请求已被停止/服务端限流

说明：当入参 dataType 值为 json 时，小程序框架会先对返回结果做 JSON.prase 操作，如果解析失败，则会返回 error 为 14 的错误。当入参 dataType 值为 text 时，如果返回的内容格式不符，也会返回 error 为 14 的错误。遇到此错误时，请先检查 dataType 的设置是否正确。

返回值

RequestTask

网络请求任务对象

说明：基础库 1.11.0 以上才支持, 可以使用 my.canIUse('request') 做兼容性处理。参见 小程序基础库说明 。

方法

RequestTask.abort()

说明：基础库 1.11.0 以上才支持, 可以使用 my.canIUse('request') 做兼容性处理。参见 小程序基础库说明 。

代码示例

```
my.request({
url: 'https://httpbin.org/post',
method: 'POST',
data: {
from: '支付宝',
production: 'AlipayJSAPI',
},
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'});
}
});

// 返回RequestTask，可以调用abort方法取消请求
const task = my.request({url: 'https://httpbin.org/post'})
task.abort()
```

my.uploadFile

说明：mPaaS 10.1.32 及以上版本支持该接口。

上传本地资源到开发者服务器。

入参

名称	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，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 返回值

名称	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

错误码

error	描述
11	文件不存在
12	上传文件失败
13	没有权限

代码示例

```
my.uploadFile({
url: '请使用自己服务器地址',
fileType: 'image',
fileName: 'file',
filePath: '...',
success: (res) => {
my.alert({
content: '上传成功'
});
```

```
},
});
```

my.downloadFile

说明：mPaaS 10.1.32 及以上版本支持该接口。

下载文件资源到本地

入参

名称	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 返回值

名称	类型	描述
apFilePath	String	文件临时存放的位置

错误码

error	描述
12	下载失败
13	没有权限

代码示例

```
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,
});
},
});
```

my.connectSocket

说明：mPaaS 10.1.60 及以上版本支持该接口。

创建一个 WebSocket 的连接。

一个支付宝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。

入参

名称	类型	必填	描述
url	String	是	目标服务器 url
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）

错误码

error	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

代码示例

```
my.connectSocket({
url: 'test.php',
data: {},
header:{
'content-type': 'application/json'
},
});
```

my.onSocketOpen

说明：mPaaS 10.1.60 及以上版本支持该接口。

监听 WebSocket 连接打开事件。

代码示例

```
my.connectSocket({
url: 'test.php',
});

my.onSocketOpen(function(res) {
console.log('WebSocket 连接已打开！');
});
```

my.offSocketOpen

说明：mPaaS 10.1.60 及以上版本支持该接口。

取消监听 WebSocket 连接打开事件。

```
Page({
onLoad() {
this.callback = this.callback.bind(this);
my.onSocketOpen(this.callback);
},
onUnload() {
my.offSocketOpen(this.callback);
},
callback(res) {
},
})
```

my.onSocketError

说明：mPaaS 10.1.60 及以上版本支持该接口。

监听 WebSocket 错误。

代码示例

```
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！');
});
```

my.offSocketError

说明：mPaaS 10.1.60 及以上版本支持该接口。

取消监听 WebSocket 错误。

代码示例

```
Page({
onLoad() {
this.callback = this.callback.bind(this);
my.onSocketError(this.callback);
},
onUnload() {
my.offSocketError(this.callback);
},
callback(res) {
},
})
```

my.sendSocketMessage

说明：mPaaS 10.1.60 及以上版本支持该接口。

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。

入参

名称	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，若为普通的文本内容 String，则不需要设置此字段。
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）。

代码示例

```
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});
},
});
```

my.onSocketMessage

说明：mPaaS 10.1.60 及以上版本支持该接口。

监听 WebSocket 接受到服务器的消息事件。

回调返回值

名称	类型	描述
data	String/Array Buffer	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。

代码示例

```
my.connectSocket({
url: '服务器地址'
})

my.onSocketMessage(function(res) {
console.log('收到服务器内容：' + res.data)
})
```

my.offSocketMessage

说明：mPaaS 10.1.60 及以上版本支持该接口。

取消监听 WebSocket 接受到服务器的消息事件。

my.closeSocket

说明：mPaaS 10.1.60 及以上版本支持该接口。

入参

关闭 WebSocket 连接。

名称	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）

代码示例

```
my.onSocketOpen(function() {
my.closeSocket()
})

my.onSocketClose(function(res) {
console.log('WebSocket 已关闭！')
})
```

my.onSocketClose

说明：mPaaS 10.1.60 及以上版本支持该接口。

监听 WebSocket 关闭。

代码示例

```
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
});
}
});
},
```

my.offSocketClose

说明：mPaaS 10.1.60 及以上版本支持该接口。

取消监听 WebSocket 关闭。

my.httpRequest

说明：

- mPaaS 10.1.32 及以上版本支持该接口。
- 在 10.1.32 版本中，如果发送的请求数据为 JSON 格式，data 字段必须先使用 JSON.stringify 方法序列化为 string 类型。

向指定服务器发起一个跨域 HTTP 请求。

入参

名称	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，默认为 {'Content-Type': 'application/x-www-form-urlencoded'}
method	String	否	默认为 GET，目前支持 GET，POST
data	Object	否	请求参数
timeout	Number	否	超时时间，单位为 ms，默认为 30000
dataType	String	否	期望返回的数据格式，默认为 JSON，支持 JSON、text 和 base64
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 返回值

名称	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
11	无权跨域
12	网络出错
13	超时
14	解码失败
19	HTTP 错误

代码示例

```
my.httpRequest({
url: 'http://httpbin.org/post',
```

```
method: 'POST',
data: {
from: '支付宝',
production: 'AlipayJSAPI',
},
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'});
}
});
```

11.8 设备

11.8.1 canIUse

my.canIUse(String)

说明：mPaaS 10.1.32 及以上版本支持该接口。

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。

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

- API 表示 API 名字。
- 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。

代码示例

```
my.canIUse('getFileInfo')
my.canIUse('closeSocket.object.code')
my.canIUse('getLocation.object.type')
my.canIUse('getSystemInfo.return.brand')
my.canIUse('lifestyle')
my.canIUse('button.open-type.share')
```

11.8.2 获取基础库版本号

my.SDKVersion

说明：mPaaS 10.1.32 及以上版本支持该接口。
此接口用于获取基础库版本号。仅供参考，代码逻辑请不要依赖此值。

代码示例

```
console.log(my.SDKVersion);
```

11.8.3 系统信息

my.getSystemInfo

说明：mPaaS 10.1.32 及以上版本支持该接口。
此接口用于获取手机系统信息。

入参

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 返回值

名称	类型	描述	最低版本
model	String	手机型号	
pixelRatio	Number	设备像素比	
windowWidth	Number	窗口宽度	
windowHeight	Number	窗口高度	
language	String	应用设置的语言	
version	String	应用版本号	
storage	String	设备磁盘容量	1.1.1
currentBattery	String	当前电量百分比	1.1.1
system	String	系统版本	1.1.1
platform	String	系统名：Android，iOS	1.1.1
titleBarHeight	Number	标题栏高度 **说明**：该返回值仅 10.1.60 版本支持。	1.1.1
statusBarHeight	Number	状态栏高度 **说明**：该返回值仅 10.1.60 版本支持。	1.1.1
screeWidth	Number	屏幕宽度	1.1.1

screenHeight	Number	屏幕高度	1.1.0
brand	String	手机品牌	1.4.0
fontSizeSetting	Number	用户设置字体大小 **说明**：该返回值仅 10.1.60 版本支持。	1.4.0
app	String	当前运行的客户端	

代码示例

```
Page({
data: {
systemInfo: {}
},
getSystemInfoPage() {
my.getSystemInfo({
success: (res) => {
this.setData({
systemInfo: res
})
}
})
},
})
```

my.getSystemInfoSync

说明：mPaaS 10.1.32 及以上版本支持该接口。

返回值同 getSystemInfo 的 success 回调参数。

代码示例

```
Page({
data: {
systemInfo: {}
},
getSystemInfoSyncPage() {
this.setData({
systemInfo: my.getSystemInfoSync()
})
}
})
```

11.8.4 网络状态

my.getNetworkType

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于获取当前网络状态。

入参

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 返回值

名称	类型	描述
networkAvailable	Boolean	网络是否可用
networkType	String	网络类型值：UNKNOWN、NOTREACHABLE、WIFI、3G、2G、4G、WWAN

代码示例

```
Page({
data: {
hasNetworkType: false
},
getNetworkType() {
my.getNetworkType({
success: (res) => {
this.setData({
hasNetworkType: true,
networkType: res.networkType
})
}
})
},
clear() {
this.setData({
hasNetworkType: false,
networkType: ''
})
},
});
```

my.onNetworkStatusChange(CALLBACK)

说明：mPaaS 10.1.32 及以上版本支持该接口。

开始监听网络状态的变化。

返回值

名称	类型	描述
isConnected	Boolean	网络是否可用
networkType	String	网络类型值： UNKNOWN、 NOTREACHABLE、 WIFI、3G、2G、4G、WWAN

代码示例

```
my.onNetworkStatusChange(function(res){
console.log(JSON.stringify(res))
})
```

my.offNetworkStatusChange

说明：mPaaS 10.1.32 及以上版本支持该接口。

取消监听网络状态的变化。

代码示例

```
my.offNetworkStatusChange()
```

11.8.5 剪贴板

my.getClipboard

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于获取剪贴板数据。

入参

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 返回值

名称	类型	描述
text	String	剪贴板数据

代码示例

```
Page({
data: {
text: '3.1415926',
copy: '',
},

handlePaste() {
my.getClipboard({
success: ({ text }) => {
```

```
this.setData({ copy: text });
},
});
},
});
```

my.setClipboard

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于设置剪贴板数据。

入参

名称	类型	必填	描述
text	String	是	剪贴板数据
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）

代码示例

```
Page({
data: {
text: '3.1415926',
copy: '',
},

handleCopy() {
my.setClipboard({
text: this.data.text,
});
},
});
```

11.8.6 摇一摇

my.watchShake(OBJECT)

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于摇一摇功能。每次调用 API，在摇一摇手机后触发回调，若需再次监听，则需再次调用此 API。

代码示例

```
Page({
watchShake() {
my.watchShake({
success: function() {
console.log('动起来了')
```

```
my.alert({ title:'动起来了 o.o'});
}
});
},
});
```

11.8.7 振动

my.vibrate(OBJECT)

说明：mPaaS 10.1.60 及以上版本支持该接口。

此接口用于调用振动功能。

代码示例

```
Page({
vibrate() {
my.vibrate({
success: () => {
my.alert({ title: '振动起来了'});
}
});
},
})
```

my.vibrateLong(OBJECT)

说明：mPaaS 10.1.60 及以上版本支持该接口。

较长时间的振动 (400 ms)。

代码示例

```
Page({
vibrateLong() {
my.vibrateLong({
success: () => {
my.alert({ title: '振动起来了'});
}
});
},
})
```

my.vibrateShort(OBJECT)

说明：mPaaS 10.1.60 及以上版本支持该接口。

较短时间的振动 (40 ms)。

代码示例

```
Page({
vibrateShort() {
my.vibrateShort({
success: () => {
my.alert({ title: '振动起来了'});
}
});
},
})
```

11.8.8 加速度计

my.onAccelerometerChange(function callback)

说明：基础库 1.9.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.60 及以上版本支持该接口。

监听加速度数据，回调间隔为 500 ms，接口调用后会自动开始监听，可使用 my.offAccelermeterChange() 停止监听。

参数

function callback ，加速度数据变化事件的回调函数

CALLBACK 返回参数

参数	类型	说明
x	Number	X 轴
y	Number	Y 轴
z	Number	Z 轴

代码示例

```
my.onAccelerometerChange(function(res) {
console.log(res.x)
console.log(res.y)
console.log(res.z)
})
```

my.offAccelerometerChange()

说明：基础库 1.9.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.60 及以上版本支持该接口。

停止监听加速度数据。

代码示例

```
my.offAccelerometerChange()
```

11.8.9 陀螺仪

my.onGyroscopeChange(function callback)

说明：基础库 1.9.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.60 及以上版本支持该接口。

监听陀螺仪数据变化事件，接口调用后会自动开始监听，回调间隔为 500 ms，可使用 my.offGyroscopeChange() 停止监听。

参数

function callback ，陀螺仪数据变化事件的回调函数。

CALLBACK 出参说明

名称	类型	描述
x	number	X 轴方向角速度
y	number	Y 轴方向角速度
z	number	Z 轴方向角速度

代码示例

```
my.onGyroscopeChange((res)=>{
console.log('gyroData.rotationRate.x = ' + res.x);
console.log('gyroData.rotationRate.y = ' + res.y);
console.log('gyroData.rotationRate.z = ' + res.z);
});
```

my.offGyroscopeChange()

说明：基础库 1.9.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.60 及以上版本支持该接口。

停止监听陀螺仪数据。

代码示例

```
my.offGyroscopeChange();
```

11.8.10 罗盘

my.onCompassChange(function callback)

说明：基础库 1.9.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.60 及以上版本支持该接口。

监听罗盘数据，接口调用后会自动开始监听，回调间隔为 500 ms，可使用 my.offCompassChange 停止监听。

参数

function callback ，陀螺仪数据变化事件的回调函数。

CALLBACK 返回参数

参数	类型	说明
direction	Number	面对的方向与正北方向的度数：[0,360)

代码示例

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

my.offCompassChange()

说明：基础库 1.9.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 ，mPaaS 10.1.60 及以上版本支持该接口。

停止监听罗盘数据。

代码示例

```
my.offCompassChange()
```

11.8.11 拨打电话

my.makePhoneCall(OBJECT)

说明：mPaaS 10.1.32 及以上版本支持该接口。

拨打电话。

入参

名称	类型	必填	描述
number	String	是	电话号码

代码示例

```
Page({
makePhoneCall() {
my.makePhoneCall({ number: '95888' });
},
});
```

11.8.12 用户截屏事件

my.onUserCaptureScreen(CALLBACK)

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于监听用户发起的主动截屏事件，可以接收到系统以及第三方截屏工具的截屏事件通知。

代码示例

```
my.onUserCaptureScreen(function() {
my.alert({
content: '收到用户截屏事件'
});
});
```

my.offUserCaptureScreen()

说明：mPaaS 10.1.32 及以上版本支持该接口。

此接口用于取消监听截屏事件。一般需要与 my.onUserCaptureScreen 成对出现。

示例代码

```
my.offUserCaptureScreen();
```

11.8.13 屏幕亮度

my.setKeepScreenOn(OBJECT)

说明：基础库 1.3.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

此接口用于设置是否保持屏幕长亮状态。仅在当前小程序生效，离开小程序后失效。

OBJECT 参数说明

参数	类型	必填	说明
keepScreenOn	Boolean	是	是否保持屏幕长亮状态
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

代码示例

```
my.setKeepScreenOn({
keepScreenOn: true,
success: (res) => {
},
fail: (res) => {
```

```
},
})
```

my.getScreenBrightness(OBJECT)

说明：基础库 1.4.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

此接口用于获取屏幕亮度。

OBJECT 参数说明

参数	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）

代码示例

```
my.getScreenBrightness({
success: (res) => {
console.log(JSON.stringify(res))
},
fail: (res) => {
},
})
```

my.setScreenBrightness(OBJECT)

说明：基础库 1.4.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 。mPaaS 10.1.32 及以上版本支持该接口。

此接口用于设置屏幕亮度。

OBJECT 参数说明

参数	类型	必填	说明
brightness	Number	是	需要设置的屏幕亮度，取值范围为 0-1
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

代码示例

```
my.setScreenBrightness({
brightness:0.5,
success: (res) => {
console.log(JSON.stringify(res))
```

```
},
fail: (res) => {
},
})
```

11.8.14 添加手机联系人

my.addPhoneContact()

说明：基础库 1.10.0 及以上版本支持该接口，低版本需要做 兼容处理 ，mPaaS 10.1.60 及以上版本支持该接口。

用户可以选择将该表单以 **创建新联系人** 或 **添加到现有联系人** 的方式，写入到手机系统的通讯录。

入参

参数	类型	必填	说明
photoFilePath	String	否	头像本地文件路径
nickName	String	否	昵称
lastName	String	否	姓氏
middleName	String	否	中间名
firstName	String	否	名字
remark	String	否	备注
mobilePhoneNumber	String	否	手机号
alipayAccount	String	否	支付宝账号
addressCountry	String	否	联系地址国家
addressState	String	否	联系地址省份
addressCity	String	否	联系地址城市
addressStreet	String	否	联系地址街道
addressPostalCode	String	否	联系地址邮政编码
organization	String	否	公司
title	String	否	职位
workFaxNumber	String	否	工作传真
workPhoneNumber	String	否	工作电话
hostNumber	String	否	公司电话
email	String	否	电子邮件
url	String	否	网站
workAddressCountry	String	否	工作地址国家
workAddressState	String	否	工作地址省份
workAddressCity	String	否	工作地址城市
workAddressStreet	String	否	工作地址街道
workAddressPostalCode	String	否	工作地址邮政编码

homeFaxNumber	String	否	住宅传真
homePhoneNumber	String	否	住宅电话
homeAddressCountry	String	否	住宅地址国家
homeAddressState	String	否	住宅地址省份
homeAddressCity	String	否	住宅地址城市
homeAddressStreet	String	否	住宅地址街道
homeAddressPostalCode	String	否	住宅地址邮政编码
success	Function	否	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）

不同手机对应用联系人的以上字段的支持程度不同, 可能不支持 Emoji 表情和颜文字, 当不支持时, 此项会被忽略。

返回值

成功

success = true

失败

error	errorMessage	说明
11	fail cancel	用户取消操作
3	fail ${detail}	调用失败，detail 加上详细信息

代码示例

```
Page({
data:{
"photoFilePath":"/sdcard/DCIM/Camera/a.png",
"nickName":"七月流火",
"lastName":"Last",
"middleName":"Middle",
"firstName":"u001DFirst",
"remark":"这里是备注",
"mobilePhoneNumber":"13800000000",
"homePhoneNumber":"11111115",
"workPhoneNumber":"11111112",
"homeFaxNumber":"11111114",
"workFaxNumber":"11111111",
"hostNumber":"11111113",
"alipayAccount":"alipay@alipay.com",
"addressCountry":"US",
"addressState":"California",
"addressCity":"San Francisco",
"addressStreet":"Mountain View",
```

```
"addressPostalCode":"94016",
"workAddressCountry":"China",
"workAddressState":"Zhejiang",
"workAddressCity":"Hangzhou",
"workAddressStreet":"Tianmushan Road",
"workAddressPostalCode":"361005",
"homeAddressCountry":"Canada",
"homeAddressState":"Ontairo",
"homeAddressCity":"Toronto",
"homeAddressStreet":"No.234 Road",
"homeAddressPostalCode":"123456",
"organization":"AntFin",
"title":"Developer",
"email":"liuhuo01@XXXXXX.com",
"url":"www.alipay.com"
},
onInput(e) {
this.data[e.currentTarget.id] = e.detail.value;
},
addPhoneContact() {
my.addPhoneContact(this.data);
}
});
```

11.8.15 扫码

my.scan

说明：mPaaS 10.1.32 及以上版本支持该接口。

调用扫一扫功能。

入参

名称	类型	必填	描述
type	String	否	扫码样式(默认为 qr)： 1. qr，扫码框样式为二维码扫码框 2. bar，扫码样式为条形码扫码框
hideAlbum	Boolean	否	是否隐藏相册（不允许从相册选择图片，只能从相机扫码）
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 返回值

名称	类型	描述
code	String	扫码所得数据
qrCode	String	扫描二维码时返回二维码数据

barCode	String	扫描条形码时返回条形码数据

错误码

error	描述
10	用户取消
11	操作失败

代码示例

```
Page({
scan() {
my.scan({
type: 'qr',
success: (res) => {
my.alert({ title: res.code });
},
});
}
})
```

11.8.16 接入蓝牙

基本流程

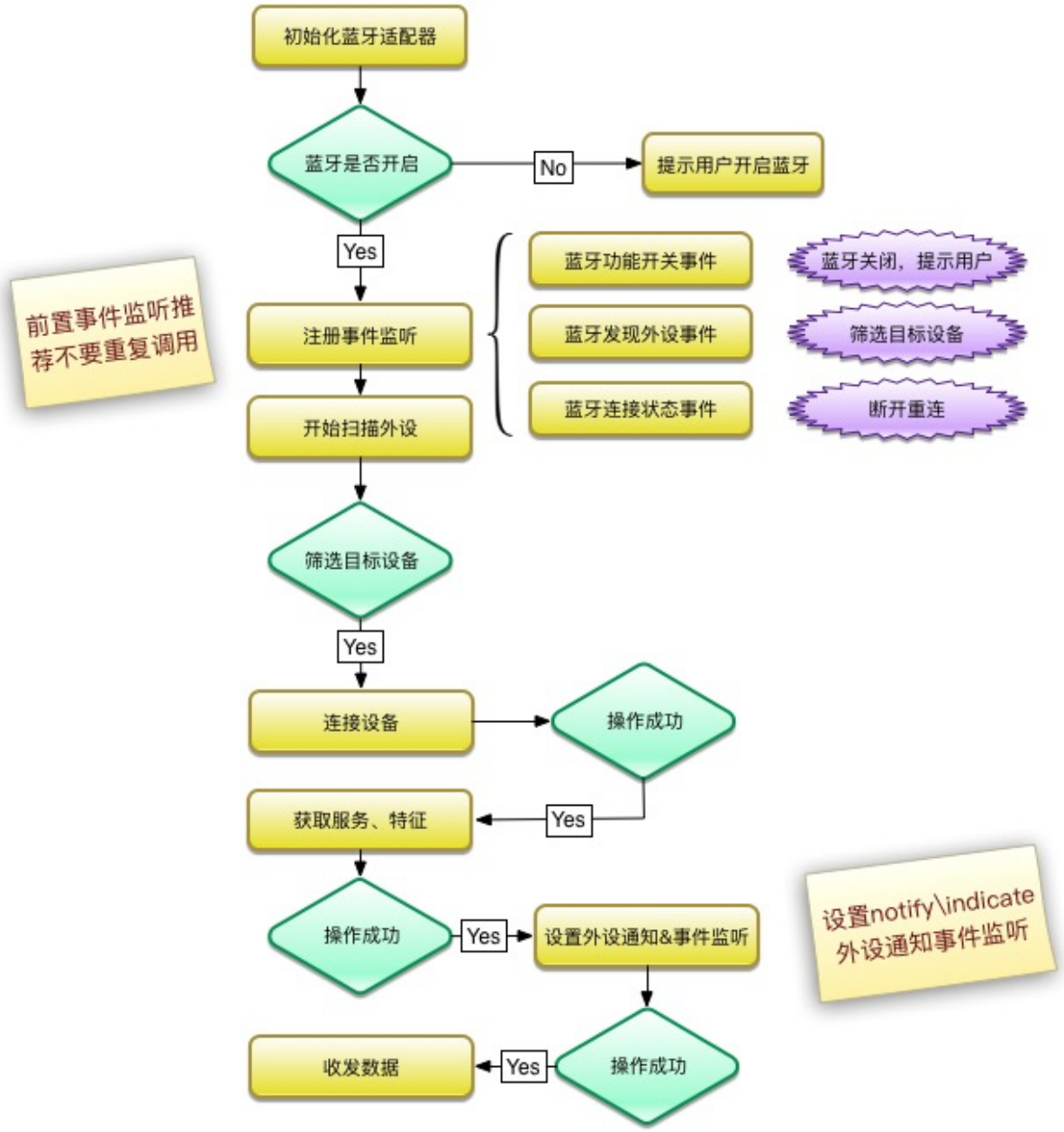
对于有蓝牙接入需求的开发者，需要通过以下步骤完成蓝牙接入：

1. 初始化蓝牙接口(my.openBluetoothAdapter)
2. 初始化事件监听
    - 蓝牙适配器状态监听(my.onBluetoothAdapterStateChange)
    - 蓝牙发现事件监听(my.onBluetoothDeviceFound)
    - 蓝牙连接状态事件监听(my.onBLEConnectionStateChanged)
3. 搜索设备(my.startBluetoothDevicesDiscovery)
4. 查找设备并连接(my.connectBLEDevice)
5. 停止搜索设备(my.stopBluetoothDevicesDiscovery)
6. 遍历蓝牙外设服务和特征
    - 获取服务(my.getBLEDeviceServices)
    - 获取特征(my.getBLEDeviceCharacteristics)
7. 监听特征值变化事件通知(my.onBLECharacteristicValueChange)
8. 设置读特征通知模式(my.notifyBLECharacteristicValueChange)
9. 读写数据
    - 向设备的特征值写数据(my.writeBLECharacteristicValue)

- 向设备的特征值读数据(my.readBLECharacteristicValue)

10. 断开连接(my.disconnectBLEDevice)
11. 关闭蓝牙适配器(my.closeBluetoothAdapter)

业务流程图



调用示例

```
//初始化
my.openBluetoothAdapter({
success: (res) => {
console.log(res);
```

```
}
});
//注册发现事件
my.onBluetoothDeviceFound({
success: (res) => {
let device = res.devices[0];
//连接发现的设备
my.connectBLEDevice({
deviceId: deviceId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
//停止搜索
my.stopBluetoothDevicesDiscovery({
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
}
});
//注册连接事件
my.onBLEConnectionStateChanged({
success: (res) => {
console.log(res);
if (res.connected) {
//开始读写notify等操作
my.notifyBLECharacteristicValueChange({
deviceId: deviceId,
serviceId: serviceId,
characteristicId: characteristicId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
}
}
});
//注册接收read或notify的数据
my.onBLECharacteristicValueChange({
success: (res) => {
console.log(res);
}
});
```

```
//开始搜索
my.startBluetoothDevicesDiscovery({
services: ['fff0'],
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});

//断开连接
my.disconnectBLEDevice({
deviceId: deviceId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});

//注销事件
my.offBluetoothDeviceFound();
my.offBLEConnectionStateChanged();
my.offBLECharacteristicValueChange();

//退出蓝牙模块
my.closeBluetoothAdapter({
success: (res) => {
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

常见问题

1. 支持 android4.3 及以上版本。
2. 支持 ios6 以上版本。
3. deviceId，Android 取到的是蓝牙的 mac 地址(11:22:33:44:55:66)，iOS 取到的是 UUID(格式：00000000-0000-0000-0000-000000000000)。
4. 调 startBluetoothDevicesDiscovery 接口搜索不到设备时，请确保设备发出了广播。如果接口中有传入 services，请确保设备的广播内容中包含了 service 的 UUID.
5. 连接设备失败时，请确认传入的 deviceId 是否正确，以及设备发出的信号是否足够强，信号弱时，可能会出现连接不上的情况。
6. 写数据失败时，查看传入的 deviceId、serviceId、characteristicId 格式是否正确，deviceId 是否已连接上(onBLEConnectionStateChanged 这个事件中可以监听连接状态变化，getConnectedBluetoothDevices 这个方法可以拿到)，确保是在连接状态下调写入方法，查看 characteristicId 是否属于这个 service，以及这个特征值是否支持写。

7. 读数据失败时，同上，查看这个特征值是否支持读。
8. 收不到数据通知时，请确认调了 notifyBLECharacteristicValueChange 这个方法以及传入的参数是否正确，传入的 characteristicId 特征值是否支持 notify 或 indicate，以及确认硬件是否发出了通知。注意调用 notifyBLECharacteristicValueChange 方法的以及注册 onBluetoothDeviceFound 事件的顺序，最好是在连接之后就调用 notifyBLECharacteristicValueChange 方法。
9. 事件回调多次调用，是由于方法多次调用了注册监听同一事件的匿名函数，因此建议每次调用 on 方法监听事件之前，先调用 off 方法，关闭之前的事件监听。

11.8.17 蓝牙 API 列表

说明：

- mPaaS 10.1.32 及以上版本支持蓝牙接口。
- 目前不支持在开发者工具上进行调试，需要使用真机才能正常调用小程序蓝牙接口。

my.openBluetoothAdapter

初始化小程序蓝牙模块，生效周期为调用 my.openBluetoothAdapter 至调用 my.closeBluetoothAdapter 或小程序被销毁为止。 在小程序蓝牙适配器模块生效期间，开发者可以正常调用下面的小程序 API，并会收到蓝牙模块相关的 on 事件回调。

入参

名称	类型	必填	描述
autoClose	Boolean	否	不传的话默认是 true，表示是否在离开当前页面时自动断开蓝牙(仅对 Android 有效)
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 返回值

名称	类型	描述
isSupportBLE	Boolean	是否支持 BLE

错误码描述

error	描述
12	蓝牙未打开
13	与系统服务的链接暂时丢失
14	未授权客户端使用蓝牙功能
15	未知错误

代码示例

```
my.openBluetoothAdapter({
success: (res) => {
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：在调用 my.openBluetoothAdapter API之前，调用小程序其它蓝牙模块相关 API，API 会返回错误，错误码：10000，错误描述:未初始化蓝牙适配器

Bug: 在用户蓝牙开关未开启或者手机不支持蓝牙功能的情况下，调用 my.openBluetoothAdapter 会返回错误，错误码见错误码描述；此时小程序蓝牙模块已经初始化完成，可通过 my.onBluetoothAdapterStateChange 监听手机蓝牙状态的改变。

my.closeBluetoothAdapter

关闭本机蓝牙模块。

入参

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）

代码示例

```
my.closeBluetoothAdapter({
success: (res) => {
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：调用该方法将断开所有已建立的蓝牙连接并释放系统资源。

Tip：建议在结束小程序蓝牙流程时调用，与 my.openBluetoothAdapter 成对调用。

Tip：调用 my.closeBluetoothAdapter 释放资源为异步操作，不建议使用 my.closeBluetoothAdapter 和

my.openBluetoothAdapter 作为异常处理流程（相当于先关闭再开启，重新初始化，效率低，易发生线程同步问题）。

my.getBluetoothAdapterState

获取本机蓝牙模块状态。

入参

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 返回值

名称	类型	描述
discovering	Boolean	是否正在搜索设备
available	Boolean	蓝牙模块是否可用(需支持 BLE 并且蓝牙是打开状态)

代码示例

```
my.getBluetoothAdapterState({
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

my.startBluetoothDevicesDiscovery

开始搜寻附近的蓝牙外围设备。搜索结果将在 my.onBluetoothDeviceFound 事件中返回。

入参

名称	类型	必填	描述
services	Array	否	蓝牙设备主 service 的 uuid 列表。
allowDuplicatesKey	Boolean	否	是否允许重复上报同一设备， 如果允许重复上报，则 onBluetoothDeviceFound 方法会多次上报同一设备，但是 RSSI 值会有不同。
interval	Integer	否	上报设备的间隔，默认为 0，意思是找到新设备立即上报，否则根据传入的间隔上报。
success	Function	否	调用成功的回调函数。
fail	Funct	否	调用失败的回调函数。

	ion		
complete	Function	否	调用结束的回调函数（调用成功、失败都会执行）。

代码示例

```
my.startBluetoothDevicesDiscovery({
services: ['fff0'],
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

- Tip：该操作比较耗费系统资源，请在搜索并连接到设备后调用 stop 方法停止搜索。

my.stopBluetoothDevicesDiscovery

停止搜寻附近的蓝牙外围设备。

入参

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）

代码示例

```
my.stopBluetoothDevicesDiscovery({
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

my.getBluetoothDevices

获取所有已发现的蓝牙设备，包括已经和本机处于连接状态的蓝牙设备。

入参

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 返回值

名称	类型	描述
devices	Array	已发现的设备列表

device 对象

名称	类型	描述
name	String	蓝牙设备名称，某些设备可能没有
deviceName(兼容旧版本)	String	值与 name 一致
localName	String	广播设备名称
deviceId	String	设备 Id
RSSI	Number	设备信号强度
advertisData	Hex String	设备的广播内容
manufacturerData	Hex String	设备的 manufacturerData

代码示例

```
my.getBluetoothDevices({
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：模拟器可能无法获取 advertisData 及 RSSI，请使用真机调试。

Tip：开发者工具和 Android 上获取到的 deviceId 为设备 MAC 地址，iOS 上则为设备 uuid。因此 deviceId 不能硬编码到代码中，需要分平台处理，iOS 可根据设备属性（ localName/advertisData/ manufacturerData 等属性）进行动态匹配。

my.getConnectedBluetoothDevices

获取处于已连接状态的设备。

入参

名称	类型	必填	描述
services	Array	否	蓝牙设备主 service 的 uuid 列表
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）

success 返回值

名称	类型	描述
devices	Array	已连接的设备列表

device 对象

名称	类型	描述
name	String	蓝牙设备名称，某些设备可能没有
deviceName(兼容旧版本)	String	值与 name 一致
localName	String	广播设备名称
deviceId	String	设备 Id
RSSI	Number	设备信号强度
advertisData	Hex String	设备的广播内容
manufacturerData	Hex String	设备的 manufacturerData

代码示例

```
my.getConnectedBluetoothDevices({
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：如果传递的 services 为空，则返回所有的已经连接的设备。

Tip：Android 上获取到的 deviceId 为设备 MAC 地址，iOS 上则为设备 uuid。因此 deviceId 不能硬编码到代码中，需要区分处理。

my.connectBLEDevice

连接低功耗蓝牙设备。

入参

名称	类型	必填	描述
deviceId	String	是	蓝牙设备 id
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）

代码示例

```
my.connectBLEDevice({
// 这里的 deviceId 需要在上面的 getBluetoothDevices 或 onBluetoothDeviceFound 接口中获取
deviceId: deviceId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

> Tip：若小程序在之前已有搜索过某个蓝牙设备，可直接传入之前搜索获取的 deviceId 直接尝试连接该设备，无需进行搜索操作。

> Tip：若指定的蓝牙设备已经连接，重复连接直接返回成功。

my.disconnectBLEDevice

断开与低功耗蓝牙设备的连接。

入参

名称	类型	必填	描述
deviceId	String	是	蓝牙设备 id
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）

代码示例

```
my.disconnectBLEDevice({
deviceId: deviceId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：蓝牙连接随时可能断开，建议监听 my.onBLEConnectionStateChanged 回调事件，当蓝牙设备断开时按需执行重连操作。

Tip：若对未连接的设备或已断开连接的设备调用数据读写操作的接口，会返回 10006 错误，详见 错误码 ，建议进行重连操作。

my.writeBLECharacteristicValue

向低功耗蓝牙设备特征值中写入数据。

入参

名称	类型	必填	描述
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙特征值对应 service 的 uuid
characteristicId	String	是	蓝牙特征值的 uuid
value	Hex String	是	蓝牙设备特征值对应的值，16 进制字符串，限制在 20 字节内
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）

代码示例

```
my.writeBLECharacteristicValue({
deviceId: deviceId,
serviceId: serviceId,
characteristicId: characteristicId,
value: 'fffe',
success: (res) => {
console.log(res)
},
fail:(res) => {
```

```
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：设备的特征值必须支持 write 才可以成功调用，具体参照 characteristic 的 properties 属性。

Tip：写入的二进制数据需要进行 hex 编码。

my.readBLECharacteristicValue

读取低功耗蓝牙设备特征值中的数据。调用后在 my.onBLECharacteristicValueChange() 事件中接收数据返回。

入参

名称	类型	必填	描述
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙特征值对应 service 的 uuid
characteristicId	String	是	蓝牙特征值的 uuid
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 返回值

名称	类型	描述
characteristic	Object	设备特征值信息

characteristic 对象

蓝牙设备 characteristic(特征值)信息。

名称	类型	描述
characteristicId	String	蓝牙设备特征值的 uuid
serviceId	String	蓝牙设备特征值对应服务的 uuid
value	Hex String	蓝牙设备特征值的 value

代码示例

```
my.readBLECharacteristicValue({
deviceId: deviceId,
serviceId: serviceId,
characteristicId: characteristicId,
```

```
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：设备的特征值必须支持 read 才可以成功调用，具体参照 characteristic 的 properties 属性`。

Tip：并行多次调用读写接口存在读写失败的可能性。

Tip：如果读取超时，错误码 10015，my.onBLECharacteristicValueChange 接口之后可能返回数据，需要接入方酌情处理。

my.notifyBLECharacteristicValueChange

启用低功耗蓝牙设备特征值变化时的 notify 功能。注意：设备的特征值必须支持 notify/indicate 才可以成功调用，具体参照 characteristic 的 properties 属性。另外，必须先启用 notify 才能监听到设备 characteristicValueChange 事件。

入参

名称	类型	必填	描述
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙特征值对应 service 的 uuid
characteristicId	String	是	蓝牙特征值的 uuid
descriptorId	String	否	notify 的 descriptor 的 uuid（只有 Android 会用到，非必填，默认值为 00002902-0000-10008000-00805f9b34fb）
state	Boolean	否	是否启用 notify 或 indicate
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）

代码示例

```
my.notifyBLECharacteristicValueChange({
```

```
deviceId: deviceId,
serviceId: serviceId,
characteristicId: characteristicId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

Tip：订阅操作成功后需要设备主动更新特征值的 value，才会触发 my.onBLECharacteristicValueChange 。

Tip：订阅方式效率比较高，推荐使用订阅代替 read 方式。

my.getBLEDeviceServices

获取蓝牙设备所有 service（服务）。

入参

名称	类型	必填	描述
deviceId	String	是	蓝牙设备 id，参考 device 对象
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 返回值

名称	类型	描述
services	Array	设备 service 对象列表，详见下表特征值信息

service对象

蓝牙设备 service (服务)信息。

名称	类型	描述
serviceId	String	蓝牙设备服务的 uuid
isPrimary	Boolean	该服务是否为主服务

代码示例

```
my.getBLEDeviceServices({
deviceId: deviceId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

- Tip：建立连接后先执行 my.getBLEDeviceServices 与 my.getBLEDeviceCharacteristics 后再进行与蓝牙设备的数据交互。

my.getBLEDeviceCharacteristics

获取蓝牙设备所有 characteristic（特征值）。

入参

名称	类型	必填	描述
deviceId	String	是	蓝牙设备 id，参考 device 对象
serviceId	String	是	蓝牙特征值对应 service 的 uuid
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 返回值

名称	类型	描述
characteristics	Array	设备特征值列

characteristic 对象

蓝牙设备 characteristic （特征值）信息。

名称	类型	描述
characteristicId	String	蓝牙设备特征值的 uuid
serviceId	String	蓝牙设备特征值对应服务的 uuid
value	Hex String	蓝牙设备特征值对应的 16 进制值
properties	Object	该特征值支持的操作类型

properties 对象

名称	类型	描述

read	boolean	该特征值是否支持 read 操作
write	boolean	该特征值是否支持 write 操作
notify	boolean	该特征值是否支持 notify 操作
indicate	boolean	该特征值是否支持 indicate 操作

代码示例

```
my.getBLEDeviceCharacteristics({
deviceId: deviceId,
serviceId: serviceId,
success: (res) => {
console.log(res)
},
fail:(res) => {
},
complete: (res)=>{
}
});
```

Bug & Tip

- Tip：建立连接后先执行 my.getBLEDeviceServices 与 my.getBLEDeviceCharacteristics 后再进行与蓝牙设备的数据交互。

my.onBluetoothDeviceFound(callback)

搜索到新的蓝牙设备时触发此事件。

入参

名称	类型	必填	描述
callback	Function	是	事件发生时回调

callback 返回值

名称	类型	描述
devices	Array	新搜索到的设备列表

device 对象

名称	类型	描述
name	String	蓝牙设备名称，某些设备可能没有
deviceName(兼容旧版本)	String	值与 name 一致
localName	String	广播设备名称
deviceId	String	设备 Id
RSSI	Number	设备信号强度

advertisData	Hex String	设备的广播内容

代码示例

```
Page({
onLoad() {
this.callback = this.callback.bind(this);
my.onBluetoothDeviceFound(this.callback);
},
onUnload() {
my.offBluetoothDeviceFound(this.callback);
},
callback(res) {
console.log(res);
},
})
```

Bug & Tip

Tip：模拟器可能无法获取 advertisData 及 RSSI ，请使用真机调试。

Tip：开发者工具和 Android 上获取到的 deviceId 为设备 MAC 地址，iOS 上则为设备 uuid。因此 deviceId 不能硬编码到代码中，需要分平台处理，iOS 可根据设备属性（localName/advertisData/manufacturerData 等）进行动态匹配。

Tip：若在 my.onBluetoothDeviceFound 回调中包含了某个蓝牙设备，则此设备会添加到 my.getBluetoothDevices 接口获取到的数组中。

my.offBluetoothDeviceFound

移除寻找到新的蓝牙设备事件的监听。

代码示例

```
my.offBluetoothDeviceFound();
```

Bug & Tip

- Tip：为防止多次注册事件监听导致一次事件多次回调，建议每次调用 on 方法监听事件之前，先调用 off 方法，关闭之前的事件监听。

my.onBLECharacteristicValueChange(callback)

监听低功耗蓝牙设备的特征值变化的事件。

入参

名称	类型	必填	描述

callback	Function	是	事件回调函数

callback 返回值

名称	类型	描述
deviceId	String	蓝牙设备 id，参考 device 对象
serviceId	String	蓝牙特征值对应 service 的 uuid
characteristicId	String	蓝牙特征值的 uuid
value	Hex String	特征值最新的 16 进制值

代码示例

```
Page({
onLoad() {
this.callback = this.callback.bind(this);
my.onBLECharacteristicValueChange(this.callback);
},
onUnload() {
my.offBLECharacteristicValueChange(this.callback);
},
callback(res) {
console.log(res);
},
})
```

my.offBLECharacteristicValueChange

移除低功耗蓝牙设备的特征值变化事件的监听。

代码示例

```
my.offBLECharacteristicValueChange();
```

my.onBLEConnectionStateChanged(callback)

监听低功耗蓝牙连接的错误事件，包括设备丢失，连接异常断开等。

名称	类型	必填	描述
callback	Function	是	事件回调函数

callback 返回值

名称	类型	描述
deviceId	String	蓝牙设备 id，参考 device 对象
connected	Boolean	连接目前的状态

Bug & Tip

- Tip：为防止多次注册事件监听导致一次事件多次回调，建议每次调用 on 方法监听事件之前，先调用 off 方法，关闭之前的事件监听。

my.offBLEConnectionStateChanged

移除低功耗蓝牙连接状态变化事件的监听。

代码示例

```
my.offBLEConnectionStateChanged();
```

Bug & Tip

- Tip：为防止多次注册事件监听导致一次事件多次回调，建议每次调用 on 方法监听事件之前，先调用 off 方法，关闭之前的事件监听。

my.onBluetoothAdapterStateChange(callback)

监听本机蓝牙状态变化的事件。

名称	类型	必填	描述
callback	Function	是	事件回调函数

callback 返回值

名称	类型	描述
available	Boolean	蓝牙模块是否可用
discovering	Boolean	蓝牙模块是否处于搜索状态

my.offBluetoothAdapterStateChange

移除本机蓝牙状态变化的事件的监听。

代码示例

```
my.offBluetoothAdapterStateChange();
```

Bug & Tip

- Tip：为防止多次注册事件监听导致一次事件多次回调，建议每次调用 on 方法监听事件之前，先调用 off 方法，关闭之前的事件监听。

错误码

错误	说明	解决方案

码		
10000	未初始化蓝牙适配器	调用 my.openBluetoothAdapter
10001	当前蓝牙适配器不可用	检查当前设备是否支持 BLE 并开启蓝牙功能
10002	没有找到指定设备	检查 deviceId 是否错误，或者是否开启外设广播
10003	连接失败	检查 deviceId 是否错误，目标蓝牙外设是否开启广播
10004	没有找到指定服务	检查 serviceId，确认目标外设是否拥有该服务
10005	没有找到指定特征值	检查 characteristId 是否正确、检查目标外设特定 service 下是否具备该特征
10006	当前连接已断开	连接断开，重新连接
10007	当前特征值不支持此操作	检查特征是否具备 读\写\通知 等功能
10008	其余所有系统上报的异常	其他未知错误，具体问题具体分析
10009	Android 系统特有，系统版本低于 4.3 时不支持 BLE	提示用户不支持
10010	没有找到指定描述符	检查 serviceId、characteristId 是否正确
10011	设备 id 不可用/为空	使用正确的 deviceId
10012	服务 id 不可用/为空	使用正确的 serviceId
10013	特征 id 不可用/为空	使用正确的 characteristId
10014	发送的数据为空或格式错误	检查写数据或者 HEX 转化是否错误
10015	操作超时	重新操作
10016	缺少参数	检查调用的参数，并重新操作
10017	写入特征值失败	写失败，检查外设特征是否支持写操作，是否断开连接
10018	读取特征值失败	读失败，检查外设特征是否支持读操作，是否断开连接

11.9 数据安全

my.rsa

说明：mPaaS 10.1.32 及以上版本支持该接口。

非对称加密。

加密与解密过程分别放置在客户端与服务端，且私钥放在服务端，私钥放在客户端易泄露将导致安全问题。

入参

名称	类型	必填	描述
action	String	是	使用 RSA 加密还是 RSA 解密。encrypy 表示加密；decrypt 表示解密。
text	String	是	要处理的文本，加密为原始文本，解密为 Base64 编码格式文本
key	String	是	RSA 秘钥，加密使用公钥，解密使用私钥
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 返回值

名称	类型	描述
text	String	经过处理过后得到的文本，加密为 Base64 编码文本，解密为原始文本

错误码

错误码	描述
10	参数错误
11	key 错误

代码示例

```
Page({
data: {
inputValue: '',
outputValue: '',
},
onInput: function (e) {
this.setData({ inputValue: e.detail.value });
},
onEncrypt: function () {
my.rsa({
action: 'encrypt',
text: this.data.outputValue,
//设置公钥
key: 'MIGfMA0GCSqXXXXXXAQUAA4GNADCBiQKBgQDKmi0dUSVQ04hL6GZGPMFK8+d6\n' +
'GzulagP27qSUBYxIJfE04KT+OHVeFFb6XXXXXXea5mkmZrIgp022zZXXXXXXNM62\n' +
'3ouBXXXsfm2ekey8PpQxfXaj8lhM9t8rJlXXXXXXs8Qp7Q5/uYrowQbT9m6t7BFK\n' +
'3egOO2xOKzLpYSqfbQIDAQAB',
success: (result) => {
this.setData({ outputValue: result.text });
},
fail(e) {
my.alert({
content: e.errorMessage || e.error,
});
},
});
},
```

```
onDecrypt: function () {
my.rsa({
action: 'decrypt',
text: this.data.inputValue,
//设置私钥
key: 'MIICdwIXXXXXXgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAMqaLR1RJVDTiEvo\n' +
'ZkY8wUrz53obO6VqA/bupJQFjEgl8TTgpP44dVXXXXXX7ydYN5rmaSZmsiCnTbbN\n' +
'lUOB1Y80zrbeXXXXXXx+bZ6R7Lw+lDF9dqPyWEz23ysmULgURzSzxCntDn+5iujB\n' +
'BtP2bq3sEUrd6A47bE4rMulhKp9tAgMBAAECgYBjsfRLXXXXXX9hou1Y2KKg+F5K\n' +
'ZsY2AnIK+6l+XXXXXXAx7e0ir7OJZObb2eyn5rAOCB1r6RL0IH+XXXXXXZANNG9g\n' +
'pXvRgcZzFY0oqdMZDuSJjpMTj7OXXXXXXGncBfvjAg0zdt9QGAG1at9Jr3i0Xr4X\n' +
'6WrFhtfVlmQUY1VsoQJBAPK2Qj/ClkZNtrSDfoXXXXXXLcNICqFIIGkNQ+XeuTwl\n' +
'+Gq4USTyaTOEe68MHluiciQ+QKvRAUd4E1zeZRZ02ikCQQDVscINBPTtTJt1JfAo\n' +
'wRfTzA0Lvgig136xLLeQXREcgq1lzgkf+tGyUGYoy9BXsV0mOuYAT9ldja4jhJeq\n' +
'cEulAkEAuSJ5KjV9dyb0XXXXXXC8d8o5KAodwaRIxJkPv5nCZbT45j6t9qbJxDg8\n' +
'N+vghDlHI4owvl5wwVlAO8iQBy8e8QJBAJe9CVXFV0XJR/XXXXXX66FxGzJjVi0f\n' +
'185nOXXXXXXCHG5VxxT2PUCo5mHBl8ctIj+rQvalvGs515VQ6YEVDCECQE3S0AU2\n' +
'BKyFVNtTpPiTyRUWqig4EbSXwjXdr8iBBJDLsMpdWsq7DCwv/ToBoLgXXXXXXc5/\n5DChU8P30EjOiEo=',
success: (result) => {
this.setData({ outputValue: result.text });
},
fail(e) {
my.alert({
content: e.errorMessage || e.error,
});
},
});
},
});
```

服务端加密解密示例

```
private static void testJieMi(String miwen, String privateKeyStr) {
//将Base64编码后的私钥转换成PrivateKey对象
//加密后的内容Base64解码
//用私钥解密
try {
PrivateKey privateKey = RSAUtil.string2PrivateKey(privateKeyStr);
byte[] base642Byte = RSAUtil.base642Byte(miwen);
byte[] privateDecrypt = RSAUtil.privateDecrypt(base642Byte, privateKey);
System.out.println("解密后的明文:" + new String(privateDecrypt));
} catch (Exception e) {
e.printStackTrace();
}
}

private static void testJiaMi(String message, String publicKeyStr) {
try {
//将Base64编码后的公钥转换成PublicKey对象
PublicKey publicKey = RSAUtil.string2PublicKey(publicKeyStr);
//用公钥加密
byte[] publicEncrypt = RSAUtil.publicEncrypt(message.getBytes(), publicKey);
//加密后的内容Base64编码
String byte2Base64 = RSAUtil.byte2Base64(publicEncrypt);
```

```
System.out.println(byte2Base64);
} catch (Exception e) {
e.printStackTrace();
}

}
```

11.10 分享

此功能仅在 10.1.60 以及以上基线版本中支持，客户端需在原生代码中自行实现分享功能。

onShareAppMessage

在 Page 中定义 onShareAppMessage 函数，设置该页面的分享信息。

- 每个 Page 页面的右上角菜单中默认会显示 **分享** 按钮，重写了 onShareAppMessage 函数，只是自定义分享内容。
- 该接口在用户点击 **分享** 按钮时调用。
- 此事件需要返回一个 Object，用于自定义分享内容。

Object 参数说明

参数	说明	最低版本
from	触发来源： button：页面页分享按钮触发； menu：右上角分享按钮触发。	1.10.0
target	如果 from 值为 button，则 target 为触发这次分享的 button，否则为 undefined。	1.10.0
webViewUrl	页面中包含 web-view 组件时，返回当前 web-view 的 URL。	1.6.0

返回值

名称	描述	最低版本
title	自定义分享标题。	无
desc	自定义分享描述：由于分享到微博只支持最大长度 140 个字，因此建议长度不要超过该限制。	无
path	自定义分享页面的路径，path 中的自定义参数可在小程序生命周期的 onLoad 方法中获取（参数传递遵循 http get 的传参规则）。客户端实现分享场景时，该字段在原生代码的名称对应为 **page** 。	无
imageUrl	自定义分享小图标元素，支持网络图片路径、apFilePath 路径和相对路径。使用场景详见下方说明。	1.4.0
bgImgUrl	自定义分享预览大图，建议尺寸 750x825，支持网络图片路径、apFilePath 路径和相对路径。使用场景详见下方说明。	1.9.

		0
success	分享成功后回调。	1.4.0
fail	分享失败后回调。	1.4.0

代码示例

```
Page({
onShareAppMessage() {
return {
title: '小程序示例',
desc: '小程序官方示例 Demo，展示已支持的接口能力及组件。',
path: 'page/component/component-pages/view/view?param=123'
};
},
});
```

页面内发起分享

说明：基础库版本 1.1.0 开始支持。

过给 button 组件设置属性 open-type="share"，可以在用户点击按钮后触发 Page.onShareAppMessage() 事件，并唤起分享面板，如果当前页面没有定义此事件，则点击后无效果。相关组件： button 。

App.onShareAppMessage

可以在 App(Object) 构造函数中设置全局的分享 onShareAppMessage 配置，当调用分享时，如果未配置页面级的分享设置则会使用全局的分享设置。

my.hideShareMenu(Object)

说明：基础库版本 1.7.0 开始支持，低版本需做 兼容处理 。

隐藏分享按钮。

Object 参数说明

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

代码示例

```
my.hideShareMenu();
```

客户端扩展实现

由于分享的实现自定义程度比较高，因此目前暂不提供原生的小程序分享功能，需要您自己实现。

实现方式

1. 小程序端发起分享时会调用 shareTinyAppMsg 的 JSAPI，它会将 JS 中 onShareAppMessage 方法返回的对象中的参数透传给客户端。
2. 客户端侧需实现自定义 JS 插件处理 shareTinyAppMsg 事件。关于 JS 插件如何实现，需分别参考 Android 自定义 JSAPI 和 iOS 自定义 JSAPI 。

Android 代码示例

说明：小程序右上角选项菜单中的分享按钮默认隐藏，需要通过配置容器开关显示，详情参见 容器配置 。

- 在小程序代码中发起分享示例如下：

```
Page({
data: {
height: 0,
title: '感谢体验!\n有任何建议欢迎反馈'
},
onLoad() {
const { windowHeight, windowWidth, pixelRatio } = my.getSystemInfoSync();
this.setData({
height: windowHeight * 750 / windowWidth
})
},
onShareAppMessage() {
return {
title: '结果页',
desc: '成功获取到结果',
myprop: 'hello', //自定义参数，如果文档中字段不满足需求，可自行增加，会被透传至客户端
path: 'pages/result/result'
}
}
});
```

客户端侧 H5 插件代码示例如下：

```
package com.mpaas.demo.nebula;

import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.antui.dialog.AUNoticeDialog;
import com.alipay.mobile.h5container.api.H5BridgeContext;
import com.alipay.mobile.h5container.api.H5Event;
import com.alipay.mobile.h5container.api.H5EventFilter;
import com.alipay.mobile.h5container.api.H5SimplePlugin;

public class ShareTinyMsgPlugin extends H5SimplePlugin {
```

```
private static final String ACTION_SHARE ="shareTinyAppMsg";

@Override
public void onPrepare(H5EventFilter filter) {
super.onPrepare(filter);
filter.addAction(ACTION_SHARE);
}

@Override
public boolean handleEvent(H5Event event, final H5BridgeContext context) {
String action = event.getAction();
if (ACTION_SHARE.equals(action)) {
JSONObject param = event.getParam();
String title = param.getString("title");
String desc = param.getString("desc");
String myprop = param.getString("myprop");
String path = param.getString("page");
String appId = event.getH5page().getParams().getString("appId");

// 此处可调用分享组件，实现后续功能
String message ="应用ID：" + appId +"\n"
+"title:" + title +"\n"
+"desc:" + desc +"\n"
+"myprop:" + myprop +"\n"
+"path:" + path +"\n";

AUNoticeDialog dialog = new AUNoticeDialog(event.getActivity(),
"分享结果", message,"分享成功","分享失败");
dialog.setPositiveListener(new AUNoticeDialog.OnClickPositiveListener() {
@Override
public void onClick() {
JSONObject result = new JSONObject();
result.put("success", true);
context.sendBridgeResult(result);
}
});
dialog.setNegativeListener(new AUNoticeDialog.OnClickNegativeListener() {
@Override
public void onClick() {
context.sendError(11,"分享失败");
}
});
dialog.show();
//
return true;
}
return false;
}
}
```

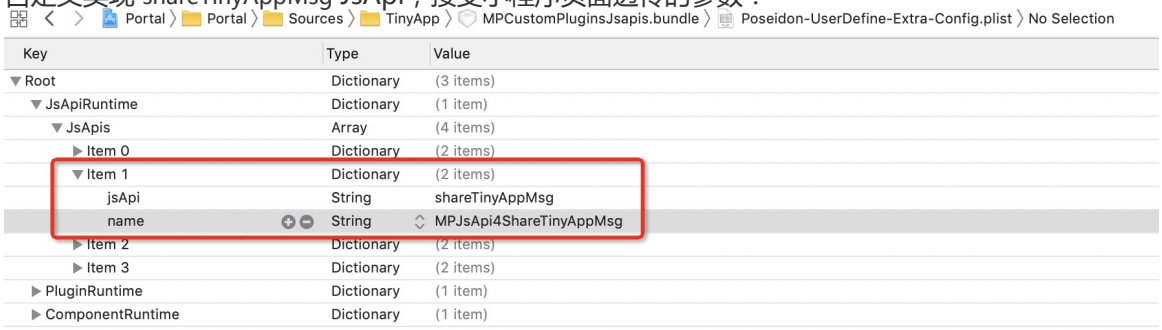
iOS 代码示例

小程序右上角的分享按钮默认为隐藏，您可以在容器初始化时，设置以下接口，以显示分享按钮：

说明：需手动引入对应头文件 #import <TinyappService/TASUtils.h>。

```
- (void)application:(UIApplication *)application afterDidFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {

。。。

[TASUtils sharedInstance].shoulShowSettingMenu = YES;

。。。
}
```

自定义实现 shareTinyAppMsg JsApi，接受小程序页面透传的参数：



在 MPJsApi4ShareTinyAppMsg 的实现类中，您可以获取到小程序页面分享的参数，进行业务处理。示例代码如下：

```
#import <NebulaPoseidon/NebulaPoseidon.h>
@interface MPJsApi4ShareTinyAppMsg : PSDJsApiHandler

@end

#import"MPJsApi4ShareTinyAppMsg.h"
#import <MessageUI/MessageUI.h>

@interface MPJsApi4ShareTinyAppMsg()<APSKLaunchpadDelegate>

@property(nonatomic, strong) NSString *shareUrlString;

@end

@implementation MPJsApi4ShareTinyAppMsg

- (void)handler:(NSDictionary *)data context:(PSDContext *)context
callback:(PSDJsApiResponseCallbackBlock)callback
{
[super handler:data context:context callback:callback];

NSString * appId = context.currentSession.createParam.expandParams[@"appId"];
NSString * page = data[@"page"]?:@"";
NSString * title = data[@"title"]?:@"";
NSString * desc = data[@"desc"]?:@"";

// 拼接分享的内容，调用分享 SDK
self.shareUrlString = [NSString stringWithFormat:@"http://appId=%@&page=%@&title=%@&desc=desc", appId,
```

```
page, title, desc];
[self openPannel];
}

- (void)openPannel {
NSArray *channelArr = @[kAPSKChannelWeibo, kAPSKChannelWeixin, kAPSKChannelWeixinTimeLine,
kAPSKChannelSMS, kAPSKChannelQQ, kAPSKChannelQQZone, kAPSKChannelDingTalkSession,
kAPSKChannelALPContact, kAPSKChannelALPTimeLine];
APSKLaunchpad *launchPad = [[APSKLaunchpad alloc] initWithChannels:channelArr sort:NO];
launchPad.tag = 1000;
launchPad.delegate = self;

[launchPad showForView:[[UIApplication sharedApplication] keyWindow] animated:YES];
}

#pragma mark - APSKLaunchpadDelegate
- (void)sharingLaunchpad:(APSKLaunchpad *)launchpad didSelectChannel:(NSString *)channelName {
[self shareWithChannel:channelName tag:launchpad.tag];
[launchpad dismissAnimated:YES];
}

- (void)shareWithChannel:(NSString *)channelName tag:(NSInteger)tag{
APSKMessage *message = [[APSKMessage alloc] init];
message.contentType = @"url";//类型分"text","image","url"三种
message.content = [NSURL URLWithString:self.shareUrlString];
message.icon = [UIImage imageNamed:@"MPShareKit.bundle/Icon_Laiwang@2x.png"];
message.title = @"这里是网页标题";
message.desc = @"这里是描述信息";
APSKClient *client = [[APSKClient alloc] init];
client.disableToastDisplay = YES;

[client shareMessage:message toChannel:channelName completionBlock:^(NSError *error, NSDictionary *userInfo) {
if(!error) {// 成功
[AUToast presentToastWithin:[[UIApplication sharedApplication] keyWindow]
withIcon:AUToastIconSuccess
text:@"分享成功"
duration:2
logTag:@"demo"];
} else {// 失败
NSString *desc = error.localizedFailureReason.length > 0 ? error.localizedFailureReason : @"分享失败";
[AUToast presentToastWithin:[[UIApplication sharedApplication] keyWindow]
withIcon:AUToastIconNone
text:desc
duration:2
logTag:@"demo"];
NSLog(@"error = %@", error);
}
}];
}

@end
```

11.11 小程序当前运行版本类型

my.getRunScene

说明：

- 基础库 1.10.0 及以上版本支持该接口，低版本需要做兼容处理，操作参见 小程序基础库说明 。
- mPaaS 1.10.60 及以上版本支持该接口。

该接口用于获取当前小程序的运行版本。

success 返回值

名称	类型	描述
envVersion	String	小程序当前运行的版本，枚举类型：develop（开发版）、trial（体验版）、release（发布版）

错误码

返回码	含义
3	发生未知错误

代码示例

```
my.getRunScene({
success(result) {
my.alert({
title: '小程序版本',
content:`${result.envVersion}`
});
},
})
```

11.12 自定义分析

my.reportAnalytics

说明：mPaaS 10.1.32 及以上版本支持该接口。

自定义分析数据的上报接口。

入参

名称	类型	必填	描述
eventName	String	是	自定义事件名
data	Object	是	上报的数据

代码示例

```
my.reportAnalytics('purchase', {
status: 200,
reason: 'ok'
});
```

11.13 自定义 API

若已有小程序 API 不满足需求，您可以自行扩展。小程序 API 复用 H5 容器的 JSAPI 插件机制，这意味着您可以按照 H5 容器提供的插件机制来扩展 API，并且小程序可以直接调用您已经写好的 JSAPI。

自定义 API

请参考 H5 容器的自定义 JSAPI 的文档来自定义 API：

- Android 自定义 JSAPI
- iOS 自定义 JSAPI

说明：小程序自定义 API 仅支持从页面调用 native，但不支持 native 向页面主动发送事件。

在小程序中调用 API

在小程序中使用如下方法来调用自定义的 API：

```
my.call(API, param, callback)
```

其中，

- API：自定义 API 的名称。
- param：调用 API 的参数。
- callback：API 执行的回调方法。

以调用 rpc 方法为例，调用示例代码如下：

```
my.call('rpc', {
operationType: 'com.test.mb1001',
requestData: [{
tranCode: 'MB1001',
customerType: 0,
customerId: 0,
UnitType: '7A238BD3-A90B-4458-885E-129230BCF7F1',
sessionId: 'zzzzzzzzzzzzzzzz',
serverIP: 'zzzzzzzzzzzzzzz',
mobileNo: username,
password,
optionFlag: 3,
}]
}, (res) => {
// do your business here.
})
```

您可以参考 H5 容器 JSAPI RPC 的文档来理解小程序和 H5 调用的异同。

11.14 小程序跳转

my.navigateToMiniProgram（Object）

说明：mPaaS 10.1.60 及以上版本支持该接口。

该接口用于跳转到其他小程序。

Object 入参说明

名称	类型	必填	描述
appId	String	是	要跳转的目标小程序 appId。
path	String	否	打开的页面路径，如果为空则打开首页。
extraData	Object	否	需要传递给目标小程序的数据，目标小程序可在 App.onLaunch()，App.onShow() 中获取到这份数据。
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）。

代码示例

```
my.navigateToMiniProgram({
appId: 'xxxx',
extraData:{
"data1":"test"
},
success: (res) => {
console.log(JSON.stringify(res))
},
fail: (res) => {
console.log(JSON.stringify(res))
}
});
```

my.navigateBackMiniProgram（Object）

说明：mPaaS 10.1.60 及以上版本支持该接口。

该接口用于跳转回上一个小程序，只有当另一个小程序跳转到当前小程序时才会能调用成功。

Object 入参说明

名称	类型	必填	描述
extraData	Object	否	需要传递给目标小程序的数据，目标小程序可在 App.onLaunch()，App.onShow() 中获取到这份数据。
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）。

代码示例

```
my.navigateBackMiniProgram({
extraData:{
"data1":"test"
},
success: (res) => {
console.log(JSON.stringify(res))
},
fail: (res) => {
console.log(JSON.stringify(res))
}
});
```

11.15 webview 组件控制

通过创建 webviewContext，提供从小程序向 web-view 发送消息的能力。

说明：

- 基础库 1.8.0 及以上版本支持本功能，低版本需做兼容处理，操作参见 小程序基础库说明 。
- mPaaS 10.1.32 及以上版本支持本功能。

my.createWebViewContext(webviewId)

该接口用于创建并返回 web-view 上下文 webViewContext 对象。

入参

参数	类型	必填	说明
webviewId	String	是	要创建的 web-view 所对应的 ID 属性。

webViewContext

webViewContext 通过 webviewId 跟一个 web-view 组件绑定，通过它可以实现一些功能。

webViewContext 对象的方法如下：

方法	参数	描述	兼容性
postMessage	Object	小程序向 web-view 组件发送消息，配合 web-view.js 中提供的 my.postMessage 可以实现小程序和 web-view 网页的双向通信。	1.8.0

代码示例

```
<view>
```

```
<web-view id="web-view-1"src="..."onMessage="onMessage"></web-view>
</view>


Page({
onLoad() {
this.webViewContext = my.createWebViewContext('web-view-1');
},
// 接收来自H5的消息
onMessage(e) {
console.log(e); //{'sendToMiniProgram': '0'}
// 向H5发送消息
this.webViewContext.postMessage({'sendToWebView': '1'});
}
})


// H5的js代码中需要先定义my.onMessage 用于接收来自小程序的消息。
my.onMessage = function(e) {
console.log(e); //{'sendToWebView': '1'}
}
// H5向小程序发送消息
my.postMessage({'sendToMiniProgram': '0'});
```

说明：以上的双向通信能力的流程是 H5 先发消息给小程序，小程序接收到消息后再发消息给 H5。

11.16 小程序 API 扩展

11.16.1 跳转支付宝卡包

目前 mPaaS 暂不支持跳转支付宝卡包，如果您的应用具备卡包能力并仍打算使用此 API，请参考 扩展说明 扩展下列 API。

my.openCardList

此接口用于打开卡列表。

代码示例

```
my.openCardList();
```

my.openMerchantCardList

此接口用于打开当前用户的某个商户的卡列表。

入参说明

名称	类型	必填	描述
partnerId	String	是	商户编号

代码示例

```
my.openMerchantCardList({partnerId:'2088xxxxx'});
```

my.openVoucherList

此接口用于打开券列表。

代码示例

```
my.openVoucherList();
```

my.openMerchantVoucherList

此接口用于打开当前用户的某个商户的券列表。

入参说明

名称	类型	必填	描述
partnerId	String	是	商户编号

代码示例

```
my.openMerchantVoucherList({partnerId:'2088xxxx'});
```

my.openTicketList

此接口用于打开票列表。

代码示例

```
my.openTicketList();
```

my.openMerchantTicketList

此接口用于打开某个商户的票列表。

入参说明

名称	类型	必填	描述
partnerId	String	是	商户编号

代码示例

```
my.openMerchantTicketList({partnerId:'2088xxxx'});
```

my.openCardDetail

此接口用于打开当前用户的某张卡的详情页。

入参说明

名称	类型	必填	描述
passId	String	是	卡实例Id

代码示例

```
//传入 passId 来打开
my.openCardDetail({passId:"11xxxxx"});
```

my.openVoucherDetail

此接口用于打开当前用户的某张券的详情页。

入参说明

名称	类型	必填	描述
passId	String	是	券实例 ID（如果传入了 partnerId 和 serialNumber，则不需传入 passId）
partnerId	String	是	商户编号（如果传入了 passId，则不需传入 partnerId）
serialNumber	String	是	序列号（如果传入了 passId，则不需传入 serialNumber）

代码示例

```
//传入 passId 来打开
my.openVoucherDetail({passId:"20170921"});
// 传入 partnerId 和 serialNumber 来打开
my.openVoucherDetail({
partnerId:"2018xxxx",
serialNumber:"20170921"
});
```

my.openTicketDetail

此接口用于打开当前用户的某张票的详情页。

入参说明

名称	类型	必填	描述
passId	String	是	卡实例 ID（如果传入了 partnerId 和serialNumber，则不需要传入 passId）
partnerId	String	是	商户编号（如果传入了 passId 则不需要传入 partnerId）
serialNumber	String	是	序列号（如果传入了 passId 则不需要传入 serialNumber）

代码示例

```
//传入 passId 来打开
my.openTicketDetail({passId:"20170921"});
// 传入 partnerId 和 serialNumber 来打开
my.openTicketDetail({
partnerId:"2088xxxx",
serialNumber:"20170921"
});
```

扩展说明

Android

您需要完成以下几步来扩展 API：

实现 OpenCardHandler 接口，按照接口说明打开应用中的卡包列表或详情页。

```
package com.mpaas.nebula.adapter.api;

public interface OpenCardHandler {

/**
* 打开卡列表
* @param partnerId 不为空时，表示打开某个商户的卡列表
*/
void openCardList(String partnerId);

/**
* 打开券列表
* @param partnerId 不为空时，表示打开某个商户的券列表
*/
void openVoucherList(String partnerId);

/**
* 打开票列表
* @param partnerId 不为空时，表示打开某个商户的票列表
*/
void openTicketList(String partnerId);

/**
* 打开某个卡的详情页
* @param passId 卡实例Id
*/
void openCardDetail(String passId);

/**
* 打开某个券的详情页
* @param passId 卡实例Id
* @param partnerId 商户编号
* @param serialNumber 序列号
*/
void openVoucherDetail(String passId, String partnerId, String serialNumber);

/**
```

```
* 打开某个票的详情页
* @param passId 卡实例Id
* @param partnerId 商户编号
* @param serialNumber 序列号
*/
void openTicketDetail(String passId, String partnerId, String serialNumber);
}
```

调用 MPTinyHelper.setOpenCardHandler 方法设置 OpenCardHandler 接口的实例。

```
public class MPTinyHelper {
public static MPTinyHelper getInstance();
public void setOpenCardHandler(OpenCardHandler handler);
}
```

iOS

参考 自定义 JSAPI > Plist 注册 > 注册 JSAPI 配置新的 JSAPI。

您需要完成以下几步来扩展 API：

1. 开发者实现 openCardList 等对应的 JSAPI 实现类 JsApiHandlerSample。
2. 根据接口入参、出参处理，入参会在 data 中。
3. 注册自定义 JSAPI。

自定义 JSAPI 代码示例（不包含注册）：

```
#import"JsApiHandlerSample.h"
#import <UIKit/UIKit.h>

@interface JsApiHandlerSample()<UIAlertViewDelegate>

@property (nonatomic, strong) NSString* result;

@end

@implementation JsApiHandlerSample

- (void)handler:(NSDictionary *)data context:(PSDContext *)context
callback:(PSDJsApiResponseCallbackBlock)callback
{
[super handler:data context:context callback:callback];

UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"是否打开卡包"message:@"允许打开卡包吗
 ? "delegate:self cancelButtonTitle:@"取消"otherButtonTitles:@"确定", nil];

[alertView show];

if (self.result && [self.result isKindOfClass:[NSString class]]) {
callback(self.result);
} else {
callback(@{@"error": @(11), @"errorMessage":@"用户取消"});
```

```
}

}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
if (0 == buttonIndex) {
self.result = @"xxx";
} else {
self.result = nil;
}
}

@end
```

11.16.2 用户授权

my.getAuthCode

mPaaS 不提供用户授权功能，如果您仍想使用此 API 进行用户授权，需实现相关 JSAPI 并接入自己的用户授权机制。

入参

名称	类型	必填	描述
scopes	String/Array	否	授权类型，默认 auth_base。支持 auth_base（静默授权）/ auth_user（主动授权）/ auth_zhima（芝麻信用）
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 返回值

名称	类型	描述
authCode	String	授权码
authErrorScope	Key-Value	失败的授权类型，key 是授权失败的 scope，value 是对应的错误码。
authSucessScope	Array	成功的授权 scope

代码示例

```
my.getAuthCode({
scopes: 'auth_user',
success: (res) => {
my.alert({
content: res.authCode,
});
```

```
},
});
my.getAuthCode({
scopes: ['auth_user']
success: (res) => {
my.alert({
content: res.authCode,
});
},
});
```

扩展说明

Android

您需要完成以下几步来扩展 API：

1. 实现自定义 H5Plugin，需要拦截 getAuthCode 事件来处理用户授权。
2. 入参处理：小程序 API 的 scopes 参数在客户端中将被映射为 scopeNicks 参数，客户端可依据自己的用户授权机制进行处理或者不处理。
3. 出参处理：客户端返回给小程序返回的结果中，authcode 会被映射为 authCode，其他无变化。
4. 注册自定义 H5Plugin。

自定义 H5Plugin 代码示例（不包含注册）：

```
package com.mpaas.demo.nebula;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.antui.dialog.AUNoticeDialog;
import com.alipay.mobile.common.logging.api.LoggerFactory;
import com.alipay.mobile.h5container.api.H5BridgeContext;
import com.alipay.mobile.h5container.api.H5Event;
import com.alipay.mobile.h5container.api.H5EventFilter;
import com.alipay.mobile.h5container.api.H5SimplePlugin;
import com.alipay.mobile.nebula.util.H5Utils;

import java.util.Arrays;

public class H5GetAuthCodePlugin extends H5SimplePlugin {

private static final String TAG ="H5GetAuthCodePlugin";

private static final String GET_AUTH_CODE ="getAuthCode";

@Override
public void onPrepare(H5EventFilter filter) {
super.onPrepare(filter);
filter.addAction(GET_AUTH_CODE);
}
```

```
@Override
public boolean interceptEvent(H5Event event, final H5BridgeContext context) {
if (GET_AUTH_CODE.equals(event.getAction())) {
JSONObject params = event.getParam();
JSONArray jScopes = H5Utils.getJSONArray(params,"scopeNicks", null);
if (jScopes != null && jScopes.size() > 0) {
String[] scopes = new String[jScopes.size()];
scopes = jScopes.toArray(scopes);
LoggerFactory.getTraceLogger().info(TAG, Arrays.toString(scopes));
// 可自行处理scopes，本示例直接返回结果给小程序
AUNoticeDialog dialog = new AUNoticeDialog(event.getActivity(),"授权登录","确认授权给此小程序应用吗？","确定","取消");
dialog.setPositiveListener(new AUNoticeDialog.OnClickPositiveListener() {
@Override
public void onClick() {
JSONObject result = new JSONObject();
result.put("authcode", Long.toHexString(System.currentTimeMillis()));
context.sendBridgeResult(result);
}
});
dialog.setNegativeListener(new AUNoticeDialog.OnClickNegativeListener() {
@Override
public void onClick() {
context.sendError(11,"用户取消授权");
}
});
dialog.show();
return true;
}
}
return false;
}
}
```

iOS

参考 自定义 JSAPI > Plist 注册 > 注册 JSAPI 配置新的 JSAPI。

您需要完成以下几步来扩展 API：

1. 开发者实现 getAuthCode 对应的 JSAPI 实现类 JsApiHandlerSample。
2. 入参处理：小程序 API 传入的参数皆通过 data 对象发送给 JsApiHandlerSample 的回调方法，scopes 参数在客户端中将被映射为 scopeNicks 参数，客户端可依据自己的用户授权机制进行处理或者不处处理。
3. 出参处理：客户端返回给小程序返回的结果中，authcode 会被映射为 authCode，其他无变化。
4. 注册自定义 JSAPI。
5. 当向小程序返回正确的值时，请调用回调函数中 callback，具体返回值字段参见 success 返回值 。
6. 当出现错误，需向小程序返回错误的值时，需调用回调函数中 callback 返回失败结果。

自定义 JSAPI 代码示例（不包含注册）：

```
#import"JsApiHandlerSample.h"
#import <UIKit/UIKit.h>

@interface JsApiHandlerSample()<UIAlertViewDelegate>

@property (nonatomic, strong) NSString *authCode;

@end

@implementation JsApiHandlerSample

- (void)handler:(NSDictionary *)data context:(PSDContext *)context
callback:(PSDJsApiResponseCallbackBlock)callback
{
[super handler:data context:context callback:callback];
NSArray *scopeNicks = data[@"scopeNicks"];

UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"授权登录"message:@"确认授权给此小程序应用吗
 ? "delegate:self cancelButtonTitle:@"取消"otherButtonTitles:@"确定", nil];

[alertView show];

if (self.authCode && [self.authCode isKindOfClass:[NSString class]]) {
callback(@{@"authcode": self.authCode});
} else {
callback(@{@"error": @(11), @"errorMessage":@"用户取消授权"});
}

}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
if (0 == buttonIndex) {
self.authCode = @"xxx";
} else {
self.authCode = nil;
}
}

@end
```

11.16.3 获取会员信息

my.getAuthUserInfo

mPaaS 不提供获取会员信息的具体实现，如果您仍想使用此 API 获取会员信息，需自行实现该 API 并接入自己的会员体系。

入参

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 返回值

名称	类型	描述
nickName	String	用户昵称
avatar	String	用户头像链接

代码示例

```
my.getAuthCode({
scopes: 'auth_user',
success: (res) => {
my.getAuthUserInfo({
success: (userInfo) => {
my.alert({
content: userInfo.nickName
});
my.alert({
content: userInfo.avatar
});
}
});
},
});
```

扩展说明

Android

您需要完成以下几步来扩展 API：

1. 实现自定义 H5Plugin，需要拦截 getAuthUserInfo 事件来处理用户授权。
2. 出参处理：客户端返回给小程序返回的结果中，nick 会被映射为 nickName，userAvatar 会被映射为 avatar，其他无变化。
3. 注册自定义 H5Plugin。

自定义 H5Plugin 代码示例（不包含注册）：

```
package com.mpaas.demo.nebula;

import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.antui.dialog.AUNoticeDialog;
import com.alipay.mobile.h5container.api.H5BridgeContext;
import com.alipay.mobile.h5container.api.H5Event;
import com.alipay.mobile.h5container.api.H5EventFilter;
import com.alipay.mobile.h5container.api.H5SimplePlugin;

public class H5GetAuthUserInfoPlugin extends H5SimplePlugin {

private static final String GET_AUTH_USER_INFO ="getAuthUserInfo";

@Override
public void onPrepare(H5EventFilter filter) {
```

```
super.onPrepare(filter);
filter.addAction(GET_AUTH_USER_INFO);
}

@Override
public boolean interceptEvent(H5Event event, final H5BridgeContext context) {
if (GET_AUTH_USER_INFO.equals(event.getAction())) {
AUNoticeDialog dialog = new AUNoticeDialog(event.getActivity(),"获取会员信息","允许获取会员信息吗？","确定","取消
");
dialog.setPositiveListener(new AUNoticeDialog.OnClickPositiveListener() {
@Override
public void onClick() {
JSONObject result = new JSONObject();
result.put("nick","Test User");
result.put("userAvatar","https://userurl.img");
context.sendBridgeResult(result);
}
});
dialog.setNegativeListener(new AUNoticeDialog.OnClickNegativeListener() {
@Override
public void onClick() {
context.sendError(11,"用户取消");
}
});
dialog.show();
return true;
}
return false;
}
}
```

iOS

参考 自定义 JSAPI > Plist 注册 > 注册 JSAPI 配置新的 JSAPI。

您需要完成以下几步来扩展 API：

1. 开发者实现 getAuthCode 对应的 JSAPI 实现类 JsApiHandlerSample。
2. 出参处理：客户端返回给小程序返回的结果中，nick 会被映射为 nickName，userAvatar 会被映射为 avatar，其他无变化。
3. 注册自定义 JSAPI。
4. 当向小程序返回正确的值时，请调用回调函数中 callback，具体返回值字段参见 success 返回值 。
5. 当出现错误，需向小程序返回错误的值时，需调用回调函数中 callback 返回失败结果。

自定义 JSAPI 代码示例（不包含注册）：

```
#import"JsApiHandlerSample.h"
#import <UIKit/UIKit.h>

@interface JsApiHandlerSample()<UIAlertViewDelegate>

@property (nonatomic, strong) NSMutableDictionary *authUserInfo;
```

```
@end

@implementation JsApiHandlerSample

- (void)handler:(NSDictionary *)data context:(PSDContext *)context
callback:(PSDJsApiResponseCallbackBlock)callback
{
[super handler:data context:context callback:callback];

UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"获取会员信息"message:@"允许获取会员信息吗
 ? "delegate:self cancelButtonTitle:@"取消"otherButtonTitles:@"确定", nil];

[alertView show];

if (self.authUserInfo && [self.authUserInfo isKindOfClass:[NSMutableDictionary class]]) {
callback(self.authUserInfo);
} else {
callback(@{@"error": @(11), @"errorMessage":@"用户取消获取"});
}

}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
if (0 == buttonIndex) {
self.authUserInfo = [NSMutableDictionary dictionary];
self.authUserInfo[@"nick"] = @"Test User";
self.authUserInfo[@"userAvatar"] = @"https://userurl.img";
} else {
self.authUserInfo = nil;
}
}

@end
```

11.16.4 芝麻认证

my.startZMVerify

目前 mPaaS 不支持芝麻认证，如果您的应用已经集成芝麻认证并打算继续使用此 API，请参考 扩展说明 扩展此 API。

入参说明

名称	类型	必填	描述
bizNo	string	是	认证标识
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 返回值

名称	类型	必填	描述
token	string	是	认证标识
passed	string	是	认证是否通过
reason	string	否	认证不通过原因

代码示例

```
my.startZMVerify({
bizNo: 'your-biz-no',
success: (res) => {
my.alert({ title: 'success:' + JSON.stringify(res)});
},
fail: (res) => {
my.alert({ title: 'fail: ' + JSON.stringify(res)});
},
});
```

扩展说明

Android

您需要完成以下几步来扩展 API：

1. 实现自定义 H5Plugin，需要拦截 zm-service 事件来处理用户授权。
2. 入参与出参直接按照 API 的参数字段说明处理。
3. 注册自定义 H5Plugin。

自定义 H5Plugin 代码示例（不包含注册）：

```
package com.mpaas.demo.nebula;

import android.text.TextUtils;

import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.antui.dialog.AUNoticeDialog;
import com.alipay.mobile.h5container.api.H5BridgeContext;
import com.alipay.mobile.h5container.api.H5Event;
import com.alipay.mobile.h5container.api.H5EventFilter;
import com.alipay.mobile.h5container.api.H5SimplePlugin;

public class H5ZmVerifyPlugin extends H5SimplePlugin {

private static final String ZM_VERIFY ="zm-service";

@Override
public void onPrepare(H5EventFilter filter) {
super.onPrepare(filter);
filter.addAction(ZM_VERIFY);
}
```

```
@Override
public boolean interceptEvent(H5Event event, final H5BridgeContext context) {
if (ZM_VERIFY.equals(event.getAction())) {
JSONObject param = event.getParam();
String bizNo = param.getString("bizNo");
if (!TextUtils.isEmpty(bizNo)) {
AUNoticeDialog dialog = new AUNoticeDialog(event.getActivity(),"芝麻认证","认证完成?","完成","取消");
dialog.setPositiveListener(new AUNoticeDialog.OnClickPositiveListener() {
@Override
public void onClick() {
JSONObject result = new JSONObject();
result.put("token","xxxxxx");
result.put("passed", true);
context.sendBridgeResult(result);
}
});
dialog.setNegativeListener(new AUNoticeDialog.OnClickNegativeListener() {
@Override
public void onClick() {
JSONObject result = new JSONObject();
result.put("passed", false);
result.put("reason","用户取消");
context.sendBridgeResult(result);
}
});
dialog.show();
} else {
context.sendError(event, H5Event.Error.INVALID_PARAM);
}
return true;
}
return false;
}
}
```

iOS

参考 自定义 JSAPI > Plist 注册 > 注册 JSAPI 配置新的 JSAPI。

您需要完成以下几步来扩展 API：

1. 开发者实现 startBizService 对应的 JSAPI 实现类 JsApiHandlerSample。
2. 出参处理：客户端返回给小程序返回的结果中，nick 会被映射为 nickName，userAvatar 会被映射为 avatar，其他无变化。
3. 注册自定义 JSAPI。
4. 当向小程序返回正确的值时，请调用回调函数中 callback，具体返回值字段参见 success 返回值 。
5. 当出现错误，需向小程序返回错误的值时，需调用回调函数中 callback 返回失败结果。

自定义 JSAPI 代码示例（不包含注册）：

```
#import"JsApiHandlerSample.h"
#import <UIKit/UIKit.h>
```

```
@interface JsApiHandlerSample()<UIAlertViewDelegate>

@property (nonatomic, strong) NSMutableDictionary *result;

@end

@implementation JsApiHandlerSample

- (void)handler:(NSDictionary *)data context:(PSDContext *)context
callback:(PSDJsApiResponseCallbackBlock)callback
{
[super handler:data context:context callback:callback];
NSString *serviceName = data[@"name"];
NSString *bizNo = data[@"bizNo"];
if (bizNo && [serviceName isEqualTo:@"zm-service"] && bizNo && [bizNo isKindOfClass:[NSString class]]) {
UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"芝麻认证"message:@"认证完成?"delegate:self
cancelButtonTitle:@"取消"otherButtonTitles:@"确定", nil];

[alertView show];

callback(self.result);
} else {
callback(@{@"error": @(e_inavlid_params)});
}
}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
self.result = [NSMutableDictionary dictionary];
if (0 == buttonIndex) {
self.result[@"token"] = @"xxxxxx";
self.result[@"passed"] = @"true";
} else {
self.result[@"reason"] = @"用户取消";
self.result[@"passed"] = @"false";
}
}

@end
```

11.16.5 唤起支付

my.tradePay

发起支付。mPaaS 不提供支付的实现，如果您仍需使用此 API 进行支付，可扩展此 API 并与自己的支付系统对接。

入参

名称	类型	必填	描述
tradeNO	String	条件	此参数为交易号
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 返回值

名称	类型	描述
resultCode	String	支付结果状态码，详见下表

resultCode 支付状态码说明

resultCode	描述
9000	订单支付成功
8000	正在处理中
4000	订单支付失败
6001	用户中途取消
6002	网络连接出错
6004	支付结果未知（有可能已经支付成功），请查询商户订单列表中订单的支付状态
99	用户点击忘记密码导致支付界面退出（仅 iOS）

代码示例

```
my.tradePay({
tradeNO: '20171115210011041053366792',
success: (res) => {
my.alert({
content: JSON.stringify(res),
});
},
fail: (res) => {
my.alert({
content: JSON.stringify(res),
});
}
});
```

扩展说明

Android

您需要完成以下几步来扩展 API：

1. 实现自定义 H5Plugin，需要拦截 tradePay 事件来处理用户授权。
2. 入参处理：小程序 API 的 tradeNO 参数在客户端中将被映射为 orderStr 参数。
3. 出参处理：客户端返回给小程序返回的结果按照 API 说明中的出参进行处理。
4. 注册自定义 H5Plugin。

自定义 H5Plugin 代码示例（不包含注册）：

```
package com.mpaas.demo.nebula;

import android.text.TextUtils;

import com.alibaba.fastjson.JSONObject;
import com.alipay.mobile.antui.dialog.AUNoticeDialog;
import com.alipay.mobile.h5container.api.H5BridgeContext;
import com.alipay.mobile.h5container.api.H5Event;
import com.alipay.mobile.h5container.api.H5EventFilter;
import com.alipay.mobile.h5container.api.H5SimplePlugin;

public class H5TradePayPlugin extends H5SimplePlugin {

private static final String TRADE_PAY ="tradePay";

@Override
public void onPrepare(H5EventFilter filter) {
super.onPrepare(filter);
filter.addAction(TRADE_PAY);
}

@Override
public boolean interceptEvent(H5Event event, final H5BridgeContext context) {
if (TRADE_PAY.equals(event.getAction())) {
JSONObject param = event.getParam();
String tradeNO = param.getString("orderStr");
if (!TextUtils.isEmpty(tradeNO)) {
AUNoticeDialog dialog = new AUNoticeDialog(event.getActivity(),"发起支付","确认支付"+ tradeNO +"?","支付","不支
付");
dialog.setPositiveListener(new AUNoticeDialog.OnClickPositiveListener() {
@Override
public void onClick() {
JSONObject result = new JSONObject();
result.put("resultCode", 9000);
context.sendBridgeResult(result);
}
});
dialog.setNegativeListener(new AUNoticeDialog.OnClickNegativeListener() {
@Override
public void onClick() {
JSONObject result = new JSONObject();
result.put("resultCode", 6001);
context.sendBridgeResult(result);
}
});
dialog.show();
} else {
context.sendError(event, H5Event.Error.INVALID_PARAM);
}
return true;
}
return false;
}
}
```

iOS

参考 自定义 JSAPI > Plist 注册 > 注册 JSAPI 配置新的 JSAPI。

您需要完成以下几步来扩展 API：

1. 开发者实现 getAuthCode 对应的 JSAPI 实现类 JsApiHandlerSample。
2. 入参处理：小程序 API 传入的参数皆通过 data 对象发送给 JsApiHandlerSample 的回调方法，小程序 API 的 tradeNO 参数在客户端中将被映射为 orderStr 参数。
3. 出参处理：客户端返回给小程序返回的结果按照 API 说明中的出参进行处理。
4. 注册自定义 JSAPI。
5. 当向小程序返回正确的值时，请调用回调函数中 callback，具体返回值字段参见 success 返回值 。
6. 当出现错误，需向小程序返回错误的值时，需调用回调函数中 callback 返回失败结果。

自定义 JSAPI 代码示例（不包含注册）：

```
#import"JsApiHandlerSample.h"
#import <UIKit/UIKit.h>

@interface JsApiHandlerSample()<UIAlertViewDelegate>

@property (nonatomic, strong) NSString *result;

@end

@implementation JsApiHandlerSample

- (void)handler:(NSDictionary *)data context:(PSDContext *)context
callback:(PSDJsApiResponseCallbackBlock)callback
{
[super handler:data context:context callback:callback];
NSString *tradeNO = data[@"orderStr"];

if (tradeNO) {
UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"发起支付"message:[NSString stringWithFormat:@"确认支付 %@ ?", tradeNO] delegate:self cancelButtonTitle:@"不支付"otherButtonTitles:@"支付", nil];
[alertView show];

if (self.result && [self.result isKindOfClass:[NSString class]]) {
callback(@{@"resultCode": 9000});
} else {
callback(@{@"resultCode": 6001});
}
} else {
callback(@{@"error": @(2), @"errorMessage":@"参数无效"});
}
}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
if (0 == buttonIndex) {
self.result = @"xxx";
} else {
self.result = nil;
```

```
}
}

@end
```

12 设计指南

12.1 设计原则

12.1.1 简单清晰

在设计 mPaaS 小程序时，只有让产品做得足够简单易用，才能让更多的用户使用。用户越多，意味着市场越广、收益更大。

一个页面只做一件事情

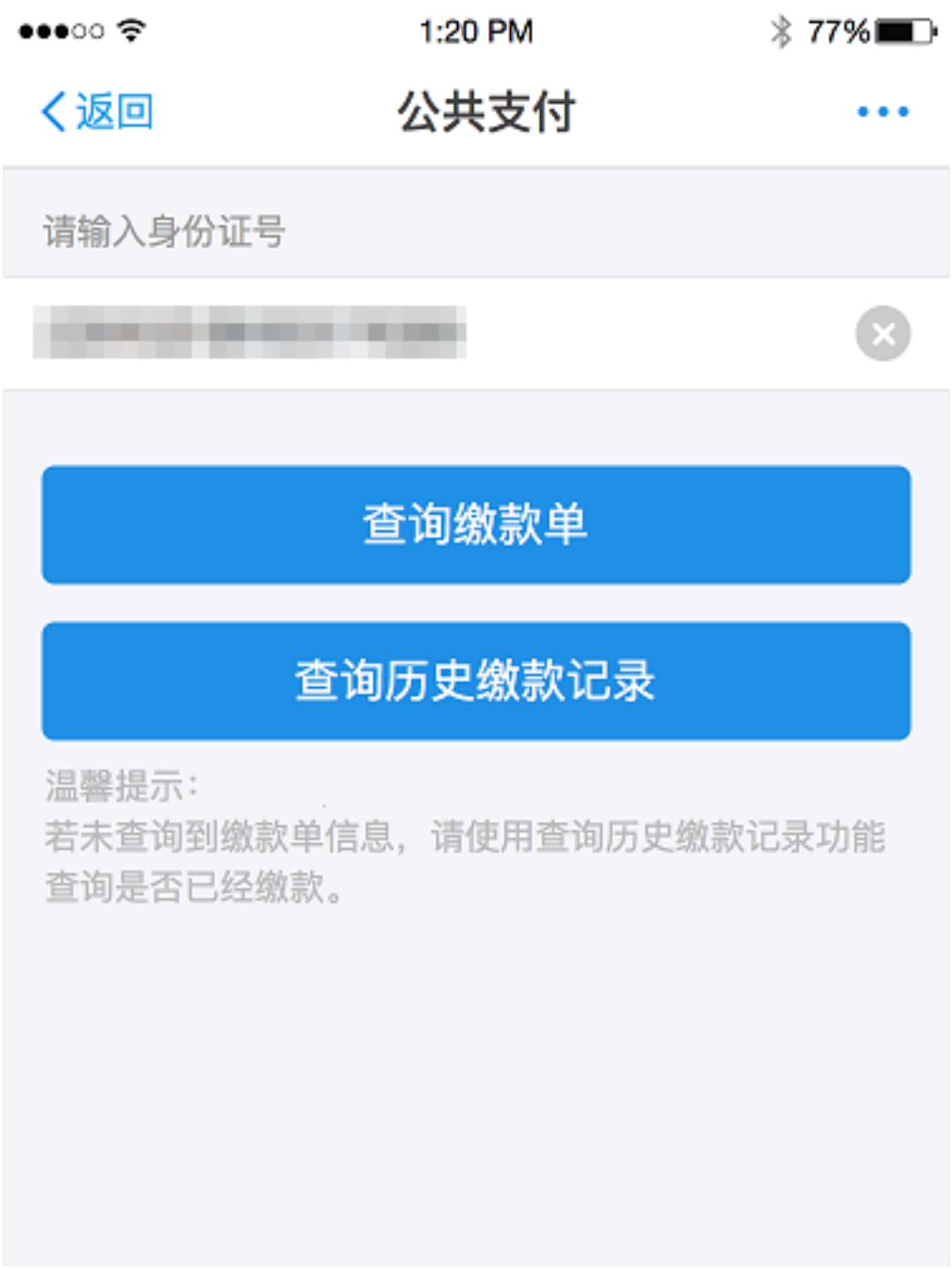
App 的一个页面能展示的信息非常有限。由于手机的使用环境非常不稳定，经常受各种因素影响，例如人们的行为活动（如：走路、坐车等)、网络信号等，这进一步限制了页面的信息量。一个页面最好能突出一个重点，让用户能够快速理解和完成任务。避免页面上出现其它与用户的决策和操作无关的干扰因素。

mPaaS 小程序服务大多是以任务为导向的，旨在帮助用户达成某个确定的任务目标，如：转账、缴费等。在任务导向类的页面中，这个原则显得尤为重要，因为我们希望用户可以专注而且快速地完成当前任务。

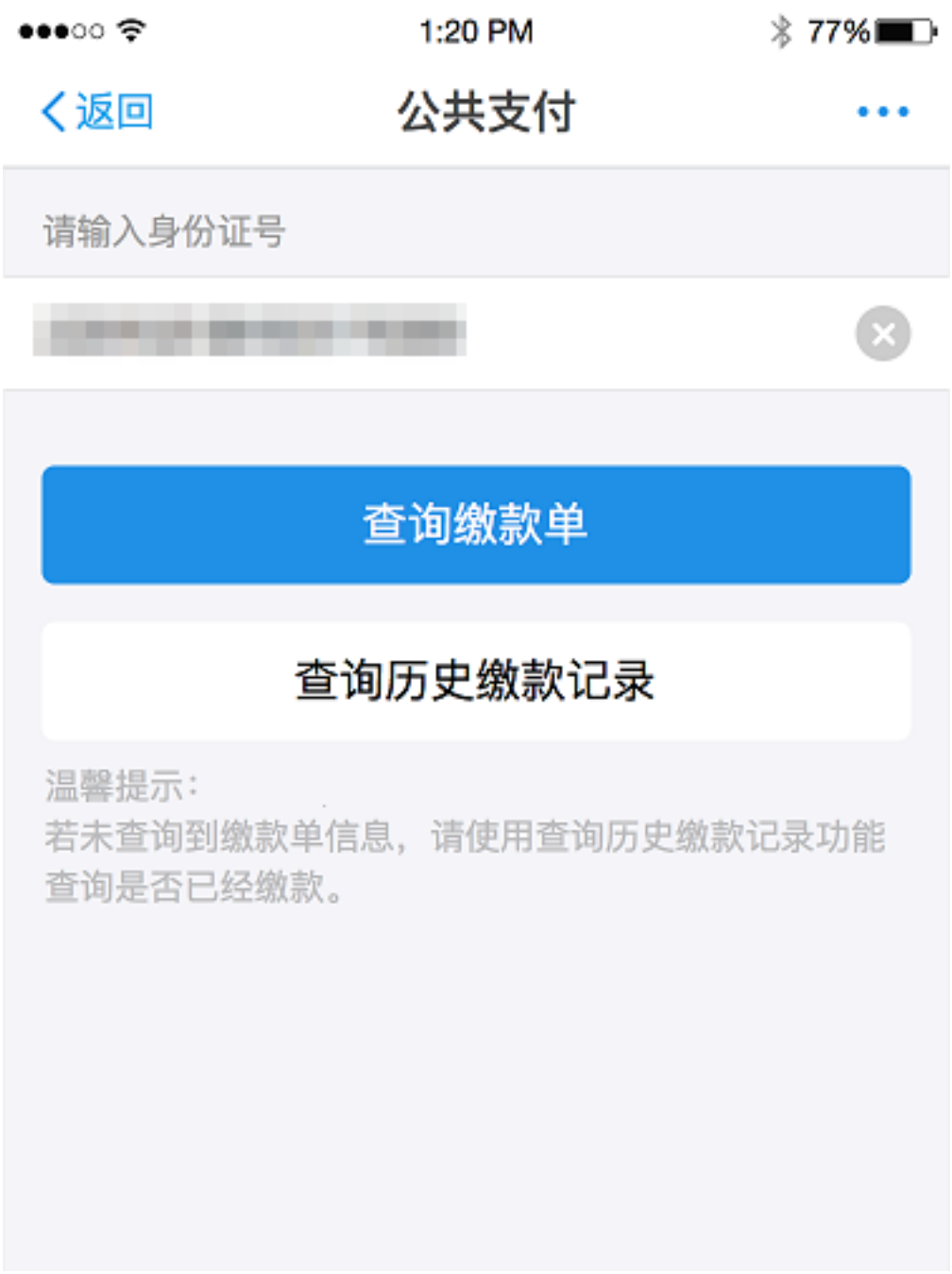
示例

如果一个页面中设置多个主行动按钮，操作没有主次，会让用户产生疑惑，无从选择。

反例示意：


1:20 PM
77%
返回
公共支付
请输入身份证号
查询缴款单
查询历史缴款记录
温馨提示：
若未查询到缴款单信息，请使用查询历史缴款记录功能查询是否已经缴款。

纠正示例：



适当删除和隐藏

人们在处理信息、学习规程和记忆细节方面的能力是有限的。现实中，人们可能还面临各种中断和打扰，这些都进一步限制了人们的认知能力。界面中过多的小细节会增加用户的认知负担，就像路障一样降低用户的使用效率。

删除可有可无的功能、多余的选项、冗余的文字、花哨的修饰，隐藏非核心功能，减轻用户的认知负担，让用户专心做自己想做的事。

导航明确

导航是确保用户在网页中浏览跳转时不迷路的最关键因素。导航需要告诉用户，当前在哪，可以去哪，如何回去等问题。

首先，要在 mPaaS 小程序的所有页面上提供导航栏，统一解决当前在哪，如何回去的问题，有助于用户在小程序内形成统一的体验和交互认知，无需在页面切换中新增学习成本或改变使用习惯。

12.1.2 高效贴心

作为服务提供者，我们应该帮助用户提升效率、创造价值，提供贴心的使用体验。随着生活的节奏越来越快，高效是一款产品必备的品质，而贴心无疑会让您的产品在众多的服务中脱颖而出，让用户越来越依赖您的产品。

要设计出一款高效贴心的产品，需要把握以下几点：

一秒钟等待

减少等待、稳定快捷，才能帮您留住用户。研究表明，用户能够忍受的最长等待时间的中位数，在 6～8 秒之间。这就是说，8 秒是一个临界值，如果页面打开速度过慢，需等待 8 秒以上，大部分用户会离你而去。

转移注意力

转移注意力是减轻等待的负面影响的常用手段。在现实生活中，转移注意力这种策略很常见。例如，一些餐饮企业在顾客排队等待就餐时，提供免费零食和休闲服务来转移顾客注意力，让顾客享受排队，减少等待时的焦躁情绪。

这种方式在应用设计中也同样管用。

一次点击

产品在使用过程中经常会有一些多余的点击，对于用户而言，这些不必要的操作都是附加工作。附加工作消耗用户的精力，但是不直接实现用户的目标。消除附加工作，可以提升操作效率，改善产品的可用性。交互设计师应该对产品中的附加工作高度敏感，才能把产品设计得更高效。

以手机充值小程序为例：


中国电信
下午2:16
支付宝
手机充值
180 8335
账户绑定号码
浙江电信
100元
优惠价：98.90
立即充值
1

支付宝 9.2 版本以前，手机充值从选择金额到付款需要四次点击：

1. 点击金额换起选择器。
2. 滑动选择金额。
3. 点击 **完成** 关闭选择器。
4. 点击 **立即充值** 进入付款页面。

9.2 版本后，充值金额平铺展现在用户面前，用户只需要一次点击选择充值金额即可进入付款页面。

减少输入

由于手机键盘区域小且密集，输入困难还易引起输入错误，因此在设计手机端页面时应尽量减少用户输入，利用现有接口或其他一些容易操作的选择控件来改善用户输入的体验。

示例

智能手机提供了各种智能传感器：摄像头、麦克风、陀螺仪。除此之外，mPaaS 小程序团队还对外开放如地理位置、账户信息等各种授权接口，充分利用这些接口可以大大减少用户的手动输入，提升产品体验。

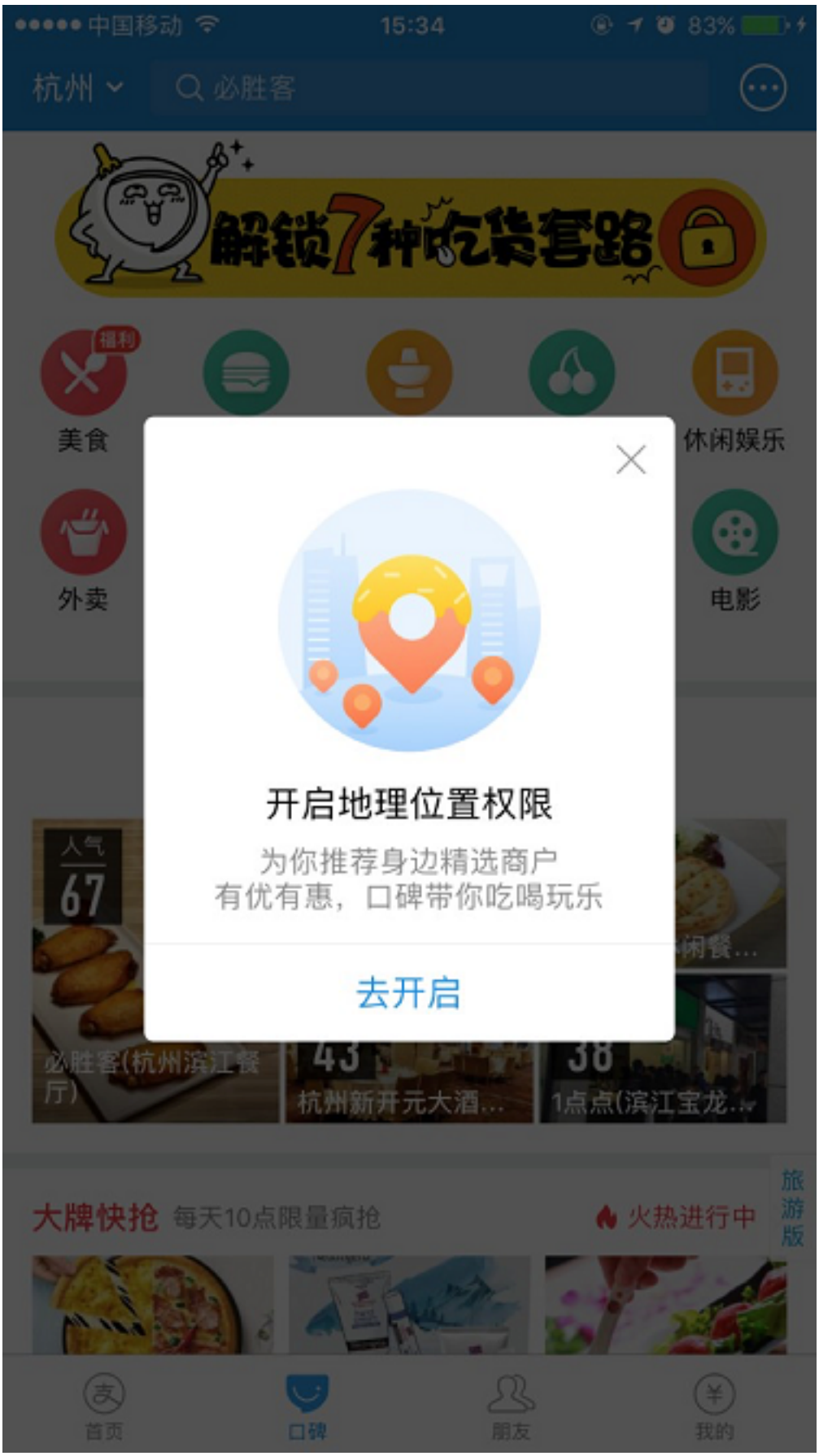
案例 1：登录页面 —— 利用摄像头设备的面部识别代替密码输入。

案例 2：添加银行卡信息页面 —— 姓名、证件号、手机号，这三个信息直接调用用户账户中的认证信息，无需手动输入。

反馈明确

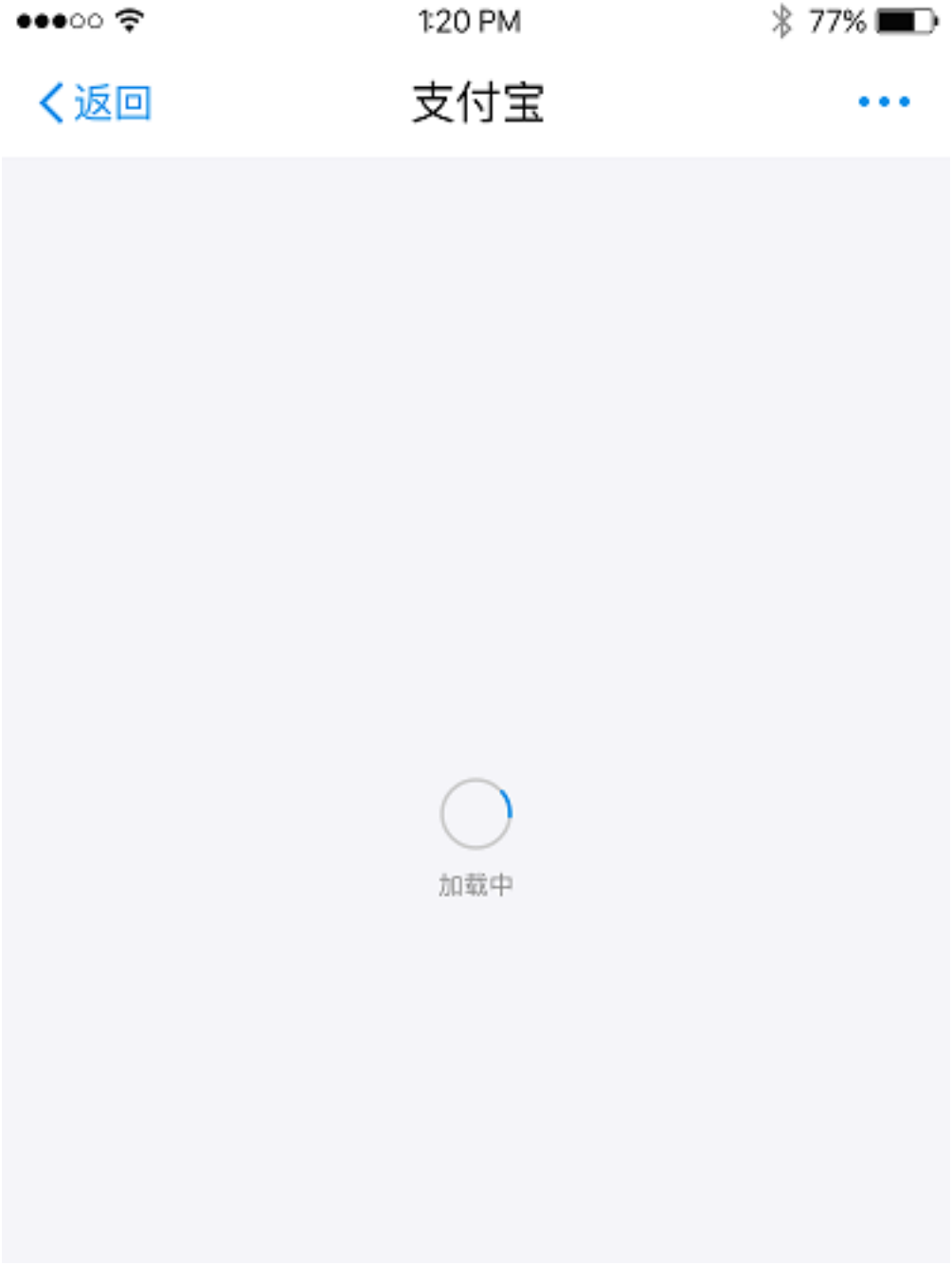
及时恰当的反馈能告诉用户下一步该做什么，帮助用户做出判断和决定，让用户知道系统运行良好稳定。所以，要营造和谐的人机对话环境，必须做到适时明确的反馈。

开启定位提示：页面无法完成自动定位服务的时候，给用户明确的反馈，并且告知用户应该去系统开启定位服务。



页面加载反馈：空白页面加载的时候，用户可能会不知道发生了什么，所以要提供明确的加载反馈告知用户加

载状态。



12.1.3 安全可控

mPaaS 小程序账户下不仅有用户的私人信息，还有大量资金。因此，账户安全十分重要。若要消除用户的顾虑，安全感的营造非常重要。作为服务提供者，您必须确保所提供的服务和应用安全可控。

信息脱敏

很多应用和服务都需要填入或者展示用户的私密信息，如：账户密码、手机号、身份证号、银行卡号等。展示和输入的时候对这些信息进行脱敏处理，隐藏信息的某一部分，而不是完全暴露在外。这样，用户会觉得您在保护他们的信息安全，才会有安全感。

输入提示

要求用户输入敏感信息的时候，明确告知用户信息的用途，消除用户的顾虑。


无 SIM 卡
上午10:57
返回
添加银行卡
请绑定持卡人本人的银行卡
持卡人
卡号
银行卡号
支付宝智能加密，保障你的用卡安全
下一步
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0

展示权威

如果服务提供渠道非常有权威，但在线上还不是广为人知。那么为了快速赢得用户的信任，打消顾虑，您可以向用户展示该渠道已有的权威。


无 SIM 卡
下午11:27
返回
存证云-合同服务
更多服务
个人房屋租赁合同
个人车位租赁合同
个人借款合同
家教雇佣合同
本服务由第三方电子证据服务平台-存证云提供
咨询热线：4008015888
存证云-合同服务
您身边的证据专家！
添加

操作可控

时刻谨记是用户控制应用，而非应用控制用户，不要冒然替用户做决定。

好设计应该是值得信任，也容易被相信的。在要求用户执行某一动作时，尽量帮他们理解为什么这个操作是必要的。每一步都需要借助诚实和清晰的表述来建立信任，逐步的积累，一点点提升用户的信任感。

建立信任的基础就是用户自己拥有控制权。

二次确认

要让用户感觉到是自己在操控，应用应该使用熟悉且可预知的交互元素。同时，在用户进行具有破坏性的操作行为时，提供二次确认，避免造成不可挽回的损失。

让用户做主

您可以对一系列用户行为提供建议，对可能造成严重后果的行为发出警告，但不要替用户做决定。好的设计会在让用户主导和避免不想要的结果中找到平衡。

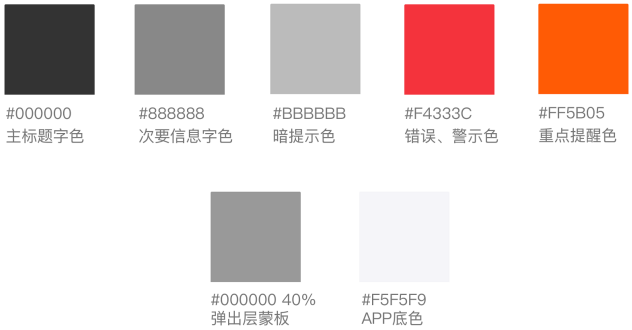
用户已经习惯于掌控其 APP 的使用体验。甚至有部分用户开始追求定制化的体验和切合自身需求的 APP。所以，保留用户选择的空间，即使是再小的选项都给予他们选择的权利，比如通知系统的开关。


无 SIM 卡
11:52
我的
设置
账户与安全
手机号
180****8335
支付设置
密码设置
生活动态管理
新消息提醒
隐私
通用
关于
退出登录

12.2 视觉规范

12.2.1 颜色

mPaaS 小程序拥有完整的色彩运用规范，下图就主要文字内容、背景等的基本用色进行说明。



12.2.2 字体

为确保 mPaaS 小程序的通用性，首选 PingFang SC 作为中文字体，以兼顾 Web 版和移动端。

字体大小与使用场景规范如下：

12.2.3 图标

图标是图形界面中的重要组成部分，具有高度浓缩并快速传达、便于记忆的特性。为让用户能更容易辨识图标信息且达成图标设计的一致性，mPaaS 小程序提供统一风格的图标设计原则。

设计原则

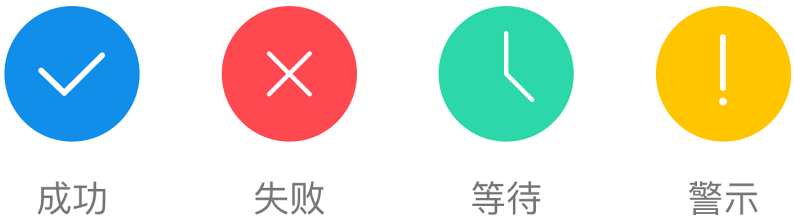
需形状鲜明，将信息化繁为简；采用几何形状、设计对称的图标来进行设计。

- **系统图标设计原则**

广泛使用圆角，避免突兀和锯齿感。以放大 36 x 36px 尺寸的图标为例，线条结构为 3px，外圆角弧度为 4px。

- **状态图标**

主要用于结果页的状态显示。



12.3 组件规范

12.3.1 导航

导航是确保用户在网页中浏览跳转时不迷路的最关键因素。导航需要告诉用户，当前在哪，可以去哪，如何回去等问题。mPaaS 小程序提供了导航栏、分段控件、标签栏这几个导航组件。

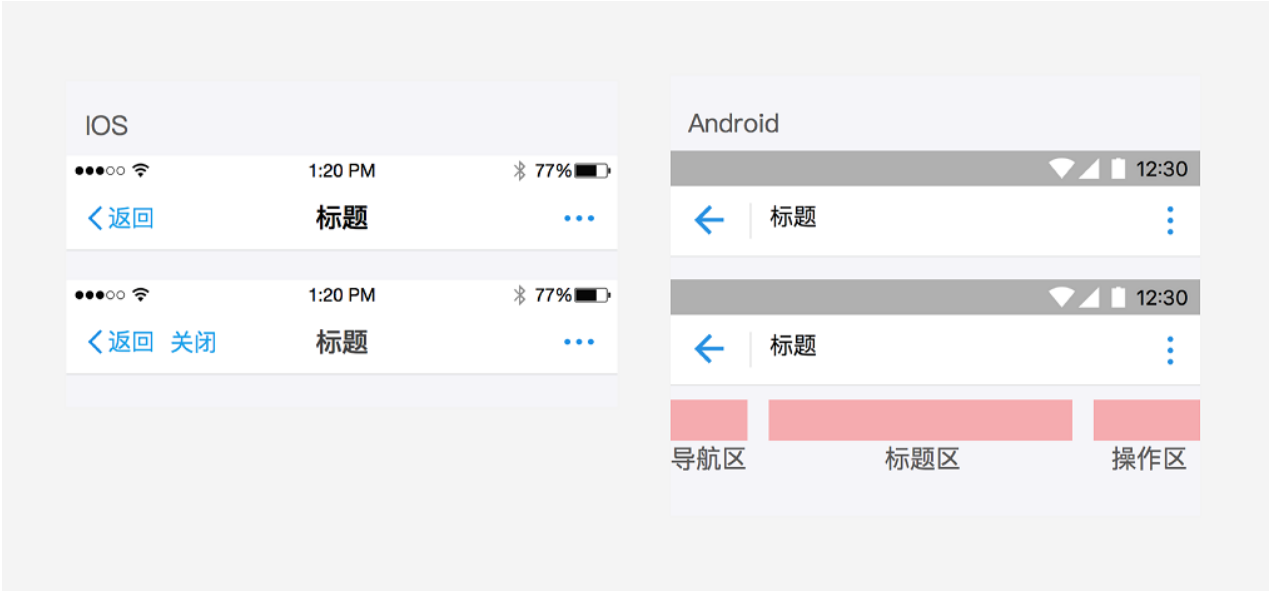
导航栏

您的页面，需要通过嵌入 mPaaS 小程序的导航框架，然后再展示给用户。mPaaS 小程序导航栏直接继承于客户端，您无需也不可以对其中的内容进行自定义。但您需要定义各个页面之间的跳转关系，让导航系统能够以合理的方式工作。

mPaaS 小程序导航栏分为导航区域、标题区域以及操作区域。

- **导航区**

导航区控制程序页面进程。在 iOS 和 Android 客户端展示有所不同，如下图所示：



在导航区，通常只有一个操作，即返回上一级界面。您只能定义其内容。当用户进入您的页面层级超过三级以后，同时显示 **返回** 和 **关闭** 按钮；点击 **关闭** 则离开当前页面，回到 mPaaS 小程序。

- **标题区**

您可以为每个页面定义标题，标题的文字展示区域固定，超出长度则会被省略。如果缺省则默认显示您的服务或应用的名称。

- **操作区**

操作区默认为 **更多** 图标，点击唤起操作菜单。您也可以自定义操作区，最多定义两个图标的操作按钮或者一个文字（两个字）的操作按钮。

自定义颜色

mPaaS 小程序导航栏支持基本的背景颜色自定义功能。选择的颜色需要在满足可用性前提下，和谐搭配 mPaaS 小程序提供的三套主导航栏图标，标题颜色会跟随背景颜色自动调整为白色或黑色。建议参考以下选色效果，选色方案示例：

正确

错误

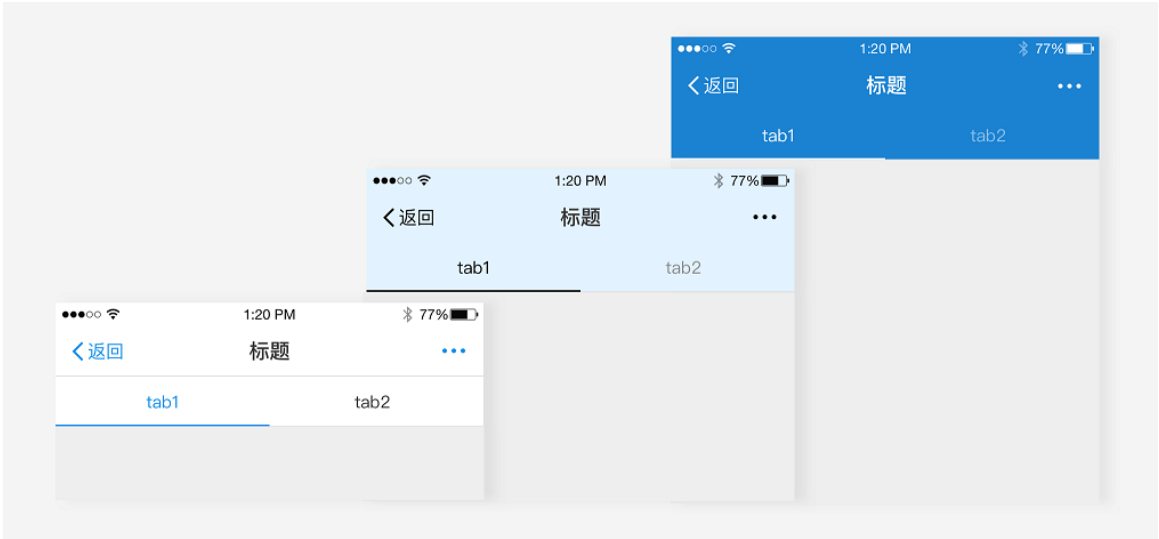
分段控件

分段控件一般出现页面的顶部或者页面中，让用户可以在页面内的不同内容之间快速切换。

设计原则：

突出显示当前选项，让用户知道所处的位置，最多设置 5 个选项，超出的选项可以滑动展示。

顶部分段控件颜色可自定义。在选择自定义颜色时，务必保证分页栏标签的可用性、可视性和可操作性。



反例示意：

标签栏

标签栏用来组织信息架构在一个层级的页面。它可以让您的应用信息层级更扁平，这样有利于用户探索到更多的内容。

标签栏位于页面底部，让用户可以在不同的页面之间快速切换。

设计原则：

最多设置 5 个 标签，当标签数超过 5 个时，最后一个用 **更多** 标签，在 **更多** 标签页里展示更多的导航选项。

标签栏的图标和文字可以自定义，但颜色不能做修改，只能使用 mPaaS 小程序提供的标准链接色。

切换页面的时候要在当前页面切换，不能新开页面。

12.3.2 信息录入

用户在和应用交互的过程中，经常需要输入、编辑、删除某些信息。多样化且有针对性的输入组件可以帮助用户快速明确地完成任务，提升产品的用户体验。

按钮

按钮用于开始一个即时操作，提交表单中的一组输入数据。


1:20 PM
77%
返回
标题
新手机号
186
校验码
7462
确定

定义及原则

按钮作为页面中的主要行动点，引导用户进行相应的主要操作。行动按钮应该醒目突出，有吸引用户点击的冲动，并且在用户进行相应的点击操作后有相应的反馈。

按钮分为主按钮、次按钮和辅助按钮。

- 主按钮：一个页面中只能出现一个主按钮，表示当前最主要的用户转化点。
- 次按钮：一个页面中可以有多个次按钮，作为当前场景的补充操作。
- 辅助按钮：位于列表右侧的操作按钮，通过按钮的形式更强烈地引导用户点击列表。

视觉样式

大按钮

大按钮出现的主要目的是鼓励用户进行操作行为。大按钮使用规范如下：

- 按钮文字需上下左右居中。
- 按钮高度固定为 94px (47pt)，圆角为 10px (5pt)。

  注意：主按钮在一个页面内只能出现一次。

小按钮

小按钮用于页面内某项内容或选项的操作/选择，可以被重复使用。小按钮使用规范如下：

- 按钮文字需上下左右居中。
- 按钮高度固定为 60px (30pt)，最小宽度为 112px (56pt)，边框粗为 2px (1pt)，圆角为 6px (3pt)。
- 按钮内文字与边框间距为 30px (15pt)，文字不够放则左右延伸宽度。

示例

按钮和页面内容一起呈现才有意义。

主按钮：


无 SIM 卡
下午7:48
返回
转到支付宝账户
转账记录
晴晴(陈牧晴)
min***@hotmail.com
转账金额
￥20
添加备注（20字以内）
确认转账
无可用付款方式，请添加银行卡

辅助按钮：


无 SIM 卡
上午10:29
返回
可能认识
宇枭
已实名
81位共同好友
加好友
怡静
已实名
27位共同好友
加好友
胜川
已实名
25位共同好友
加好友
忙果
已实名
24位共同好友
加好友
邻雨
已实名
22位共同好友
加好友
joycce
已实名
20位共同好友
加好友
瑞
已实名
19位共同好友
加好友

多选框

多选控件让用户可以同时选择多个元素。

定义及原则

多选控件一般出现在需要编辑的列表中，当用户选择完成以后统一对选中的元素进行编辑处理。多选分为选中和未选中两种状态。

文本输入框

文本输入框是最简单的输入组件，它允许用户通过键盘、选择器等组件录入单行的数据。

定义及原则

单行输入框都有信息输入长度的限制，通常最多 15 个字符。您还可以有针对性的限制输入框可输入的信息类型，如：中文、英文、数字、邮箱地址等。

激活不同类型的输入框的同时，需要弹出相应类型的键盘：文字键盘、英文键盘、数字键盘、邮箱键盘等，这样有利于提高用户的输入效率。

输入框一般由 **标签区**、**输入区**、**辅助操作区** 三个部分组成。**标签区** 有字数限制，最多 4 个字；**输入区** 一般会设置 **暗提示**；**辅助操作区** 可以放辅助输入的操作按钮，或者更详细的输入说明按钮。

如果输入的数据内容为敏感信息，应该进行脱敏处理，如：密码、手机号等。

视觉样式

标签、图标、辅助输入按钮不同的部件组成了多种样式的输入框。

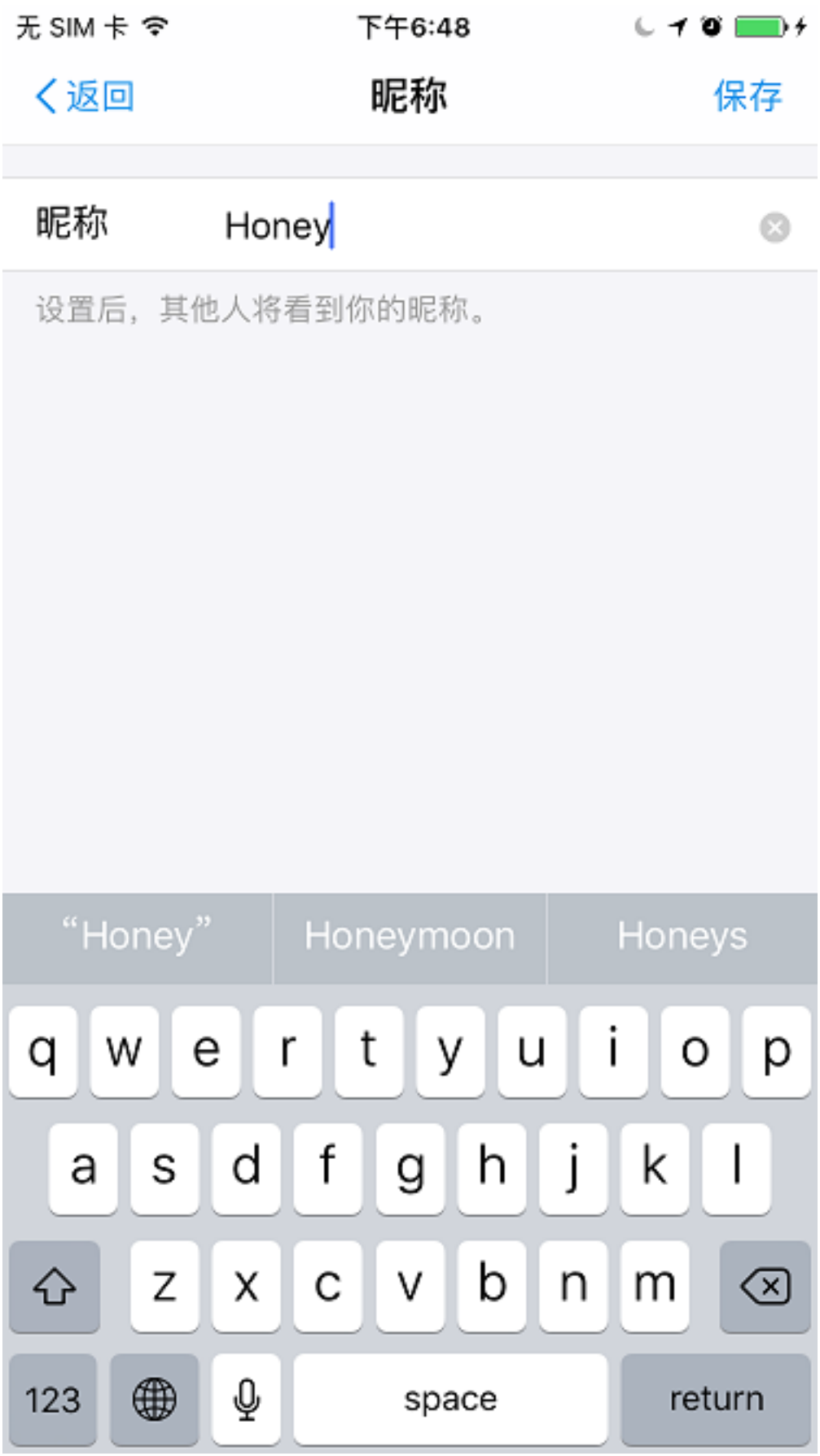

1:20 PM
77%
返回
输入框
文本
单行输入
单行输入
暗提示
取消输入
单行输入
666666
多行输入
多项输入
暗提示
最长可六个字
暗提示右移与标题间隔10px
多项输入
多项输入
暗提示
多项输入
暗提示
多项输入
暗提示


1:20 PM
77%
返回
输入框
文本
单行输入
输入文字–17pt #000
15pt
多项输入
暗提示
最长可六个字
6pt
暗提示右移与标题间隔10px
多项输入
暗提示
多项输入
暗提示
发送校验码
文本输入框
输入内容

示例

根据输入数据类型唤起相应的系统键盘。iOS、Android 系统都为不同类型的信息输入准备了相应的键盘，有助于提升用户的输入效率，进而提升用户体验。

文字键盘案例：


无 SIM 卡
下午6:48
返回
昵称
保存
昵称
Honey
设置后，其他人将看到你的昵称。
“Honey”
Honeymoon
Honeys
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
return

数字键盘案例：


无 SIM 卡
下午6:45
返回
充值中心
请输入手机号码
充话费
10元
售价:9.98元
20元
售价:19.96元
30元
售价:29.94元
50元
售价:49.90元
100元
售价:99.80元
200元
售价:199.60元
300元
售价:299.40元
500元
售价:498.99元
完成
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
+ * #
0

选择器

选择器提供一组预设的数据，让用户通过选择完成输入或者设置。

定义及原则

通过点击页面中的某个输入框会触发选择器，选择器出现时应在页面上盖上一层半透明的蒙层，让用户聚焦到选择器的选择上。

选择器中的数据最好是有一定逻辑关系，符合用户预期的数据。因为选择器中可能无法一下展示全部数据，需要用户滑动选择，符合用户预期的逻辑顺序能帮助用户快速找到想要的选型。

选择器可以设置多列数据的组合选择，最多四列，但是最长列的文字不能超过宽度限制。

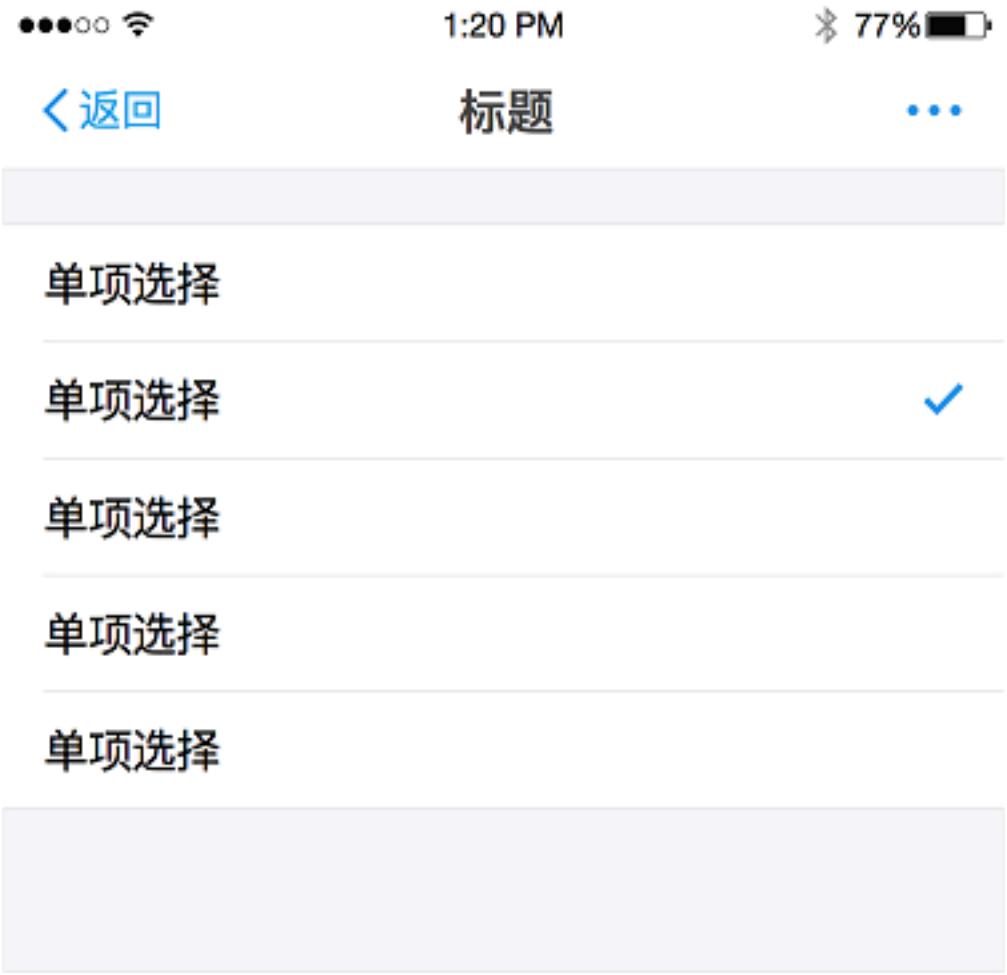
日期选择器

时间选择器可以让用户快速选择某个时间，从年、月、日到小时、分钟、秒，都可以设置。


无 SIM 卡
下午7:07
支出
取消
11月16日
选择账户
一般
餐饮
购物
服饰
交通
娱乐
社交
居家
通讯
零食
美容
运动
旅行
数码
学习
取消
确定
2013年 8月 13日
2014年 9月 14日
2015年 10月 15日
2016年 11月 16日
2017年 12月 17日
2018年 1月 18日
2019年 2月 19日

单选框

单选控件让用户选择一个元素。



定义及原则

单选控件一般出现在列表的右侧，出现一个对勾表示当前选中的选项。

搜索栏

搜索栏让用户可以在大量的信息中快速找到自己想要的内容。

定义及原则

搜索栏一般位于导航栏下方，点击激活的时候唤起系统键盘，通过 **取消** 按钮退出激活状态。

如果默认展示输入框，可以提供暗提示文案，帮助用户输入，如：关键词。在搜索栏下方，可提供有用的标签文案，帮助用户通过点击直接完成输入，如：最近搜索的内容。

滑动开关

开关是将两种状态可视化表达的一种控件。

定义及原则

开关控件只能在列表中使用，所以开关只能在列表中出现，用来表示两个互斥的选项。

12.3.3 信息展示

有序的信息展示可以帮助用户更好地理解和查找内容，从而让您的应用变得方便好用。mPaaS 小程序提供了不同的信息展示组件，您可以根据页面需求选择相应的组件展示不同类型的信息。

列表

列表是一种常用的信息组织形式，它将内容划分成一排一排，每一排都可跳转到对应的详情页面展示更多信息。


1:20 PM
77%
返回
选择项
文本
单行列表
单行列表
详细信息
单行图标+列表
标题文字
单行列表
多行列表
多行列表
多行列表
双行列表
双行列表
描述信息

列表可以通过滑动展示超过屏幕长度的信息量，还可以把有相关性的内容进行分组。

视觉样式

列表由标题、副标题、图标组成，您可以根据需求决定带不带图标。

示例

简单的功能集合类页面，用列表的形式将众多功能集合展示在一个页面上，用户通过列表的导航形式找到不同的功能项。

设置页面：


无 SIM 卡
下午8:59
我的
设置
账户管理
支付设置
密码设置
手机号
183****9304
安全中心
动态管理
朋友消息提醒
隐私
通用
我的客服
关于
退出登录

个人中心页：

通告栏

顶部公告在导航栏和页面内容之间插入，用于提醒用户一些重要信息和公告。

定义及原则

通告栏用于展示相对重要的异常、通告，如：产品即将维护；某个银行渠道什么时段不可用。

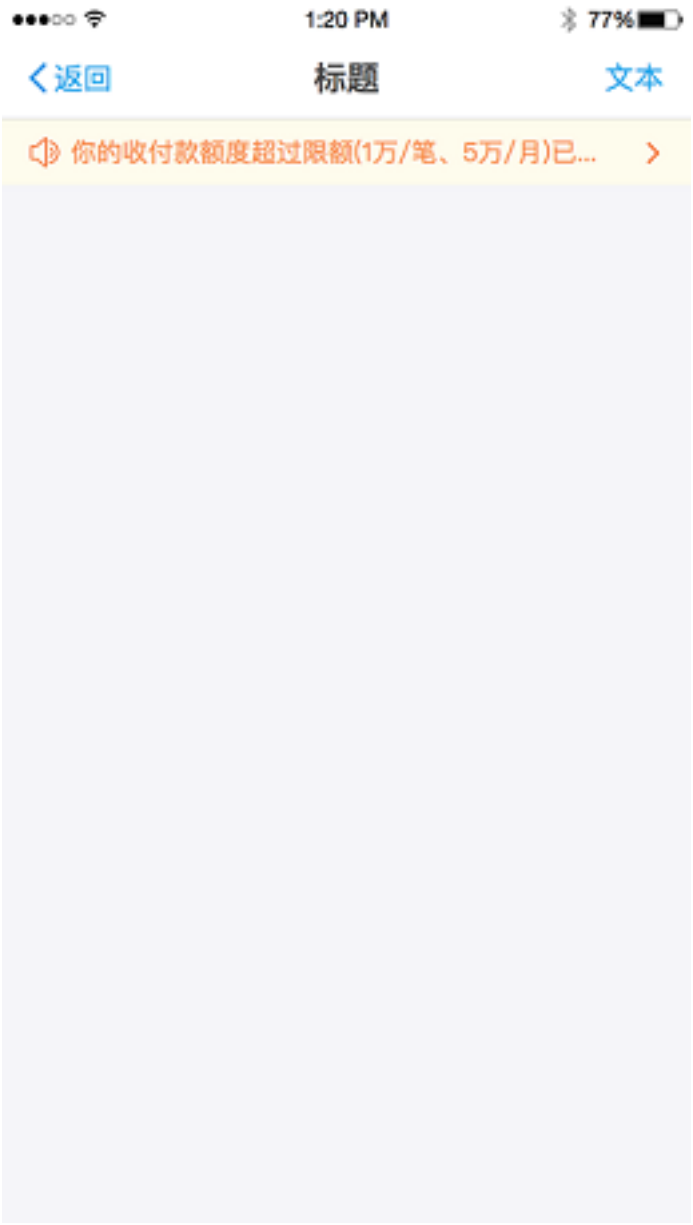
- 公告切勿用于展示产品运营类信息，否则会降低公告的公信力。
- 公告文案的长度最好不要超过屏幕的宽度，超过屏幕宽度只可用 “...” 省略，不可换行。
- 用户点击查看或者关闭公告后，不可再次出现同样的信息二次打扰用户。

如果有更详细的内容，可以通过点击公告进入详情页面。返回后公告消失。

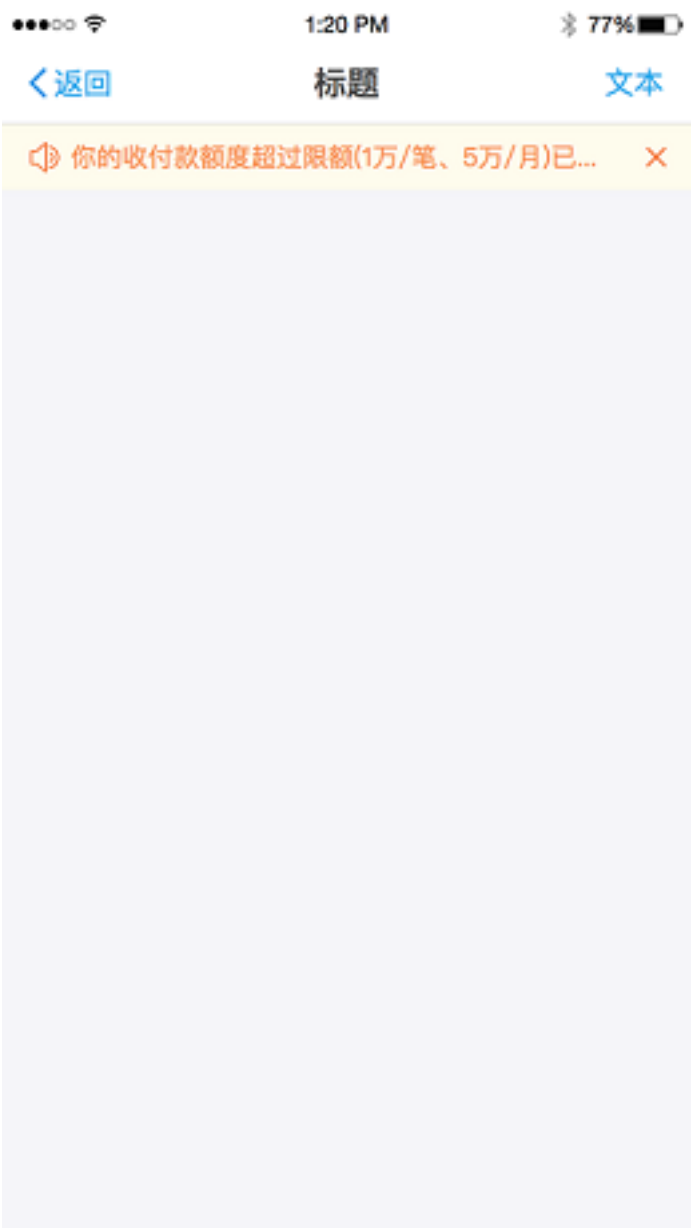
视觉样式

可配备查看详情的箭头或者关闭的按钮。

查看详情的箭头，用户点击并且进入详情页以后，公告消失。



关闭公告的按钮，用户点击 **关闭** 按钮以后公告消失。



步骤条

步骤条展示步骤的步数以及当前所处的进程。

定义及原则

步骤条向用户展示一个任务可以拆分为几个步骤，以及这些步骤的先后顺序。如果这些任务拆分为几个不同的页面来呈现，那步骤条还可以充当这几个页面的导航。

示例

以信用卡还款、余额宝转入为例，这两个任务在用户操作完成以后都需要等待一段时间，任务才算真正的完结。因此，需要在结果页通过任务的步骤条告知用户需要等待。

信用卡还款结果页：


中国电信 4G
下午9:15
10%
信用卡还款
完成
付款成功，银行处理中
1.00元
招商银行(尾号9696)
还款成功
今天到账(最晚24:00)
银行延后1-2日更新还款结果
蚂蚁会员专享
刮888元现金大奖
点我刮奖
详情查询>>
持续还款有助提升芝麻信用分

余额宝转入结果页：

12.3.4 交互反馈

人机交互过程中很重要的一点就是操作的反馈，我们要对用户的操作给出及时的反馈，即时的响应会给用户增加信赖感。

mPaaS 提供了一系列的反馈组件，需要在不同的场景下选择正确的反馈形式进行反馈。

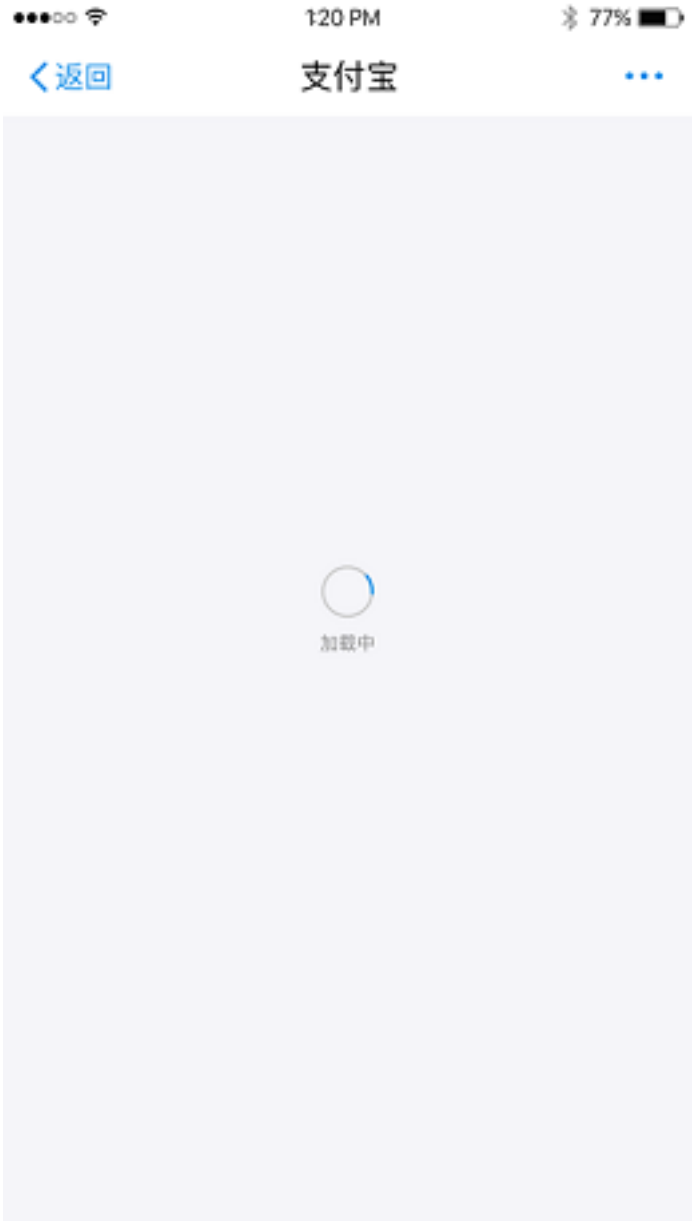
反馈原则：

1. 为用户在各个阶段的操作提供必要、积极、即时的反馈；
2. 避免过度反馈，以免给用户带来不必要的打扰，能够及时看到效果的、简单的操作，可以省略反馈提示。

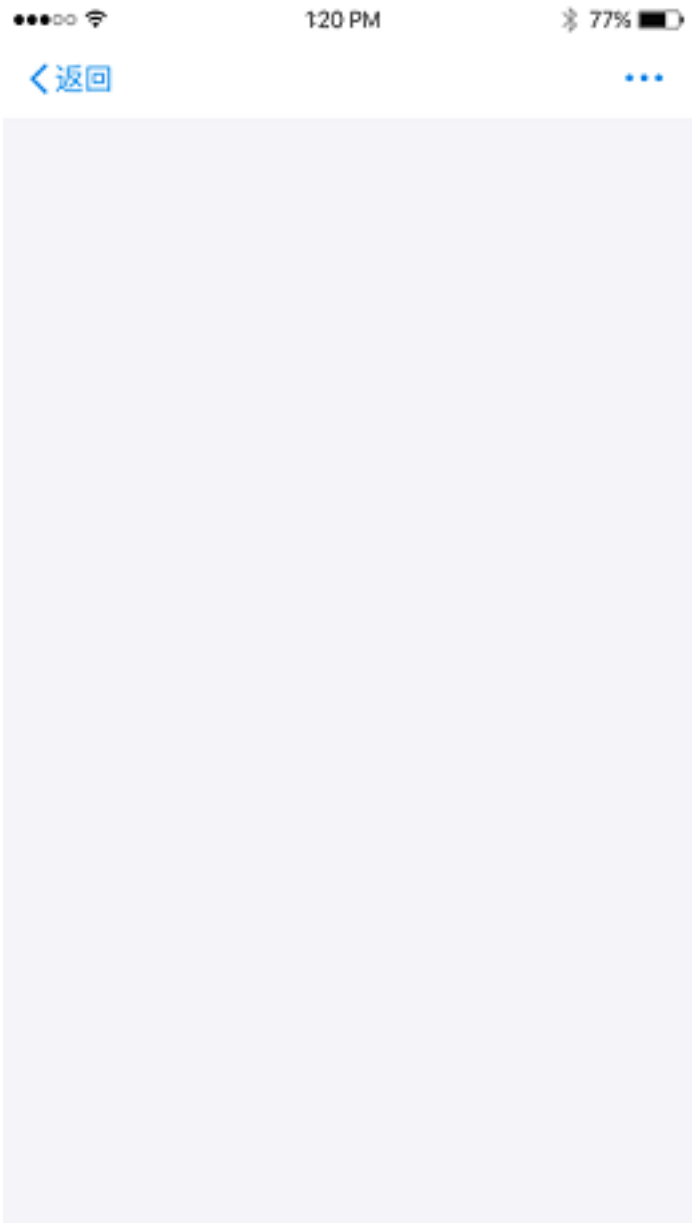
必要反馈

为用户在各个阶段的操作提供必要、积极、即时的反馈。

正确示例：打开空白页面明确告知用户需要等待



错误示例：打开空白页面，没有任何等待提示



避免过度反馈

过度反馈会给用户带来不必要的打扰，应避免。

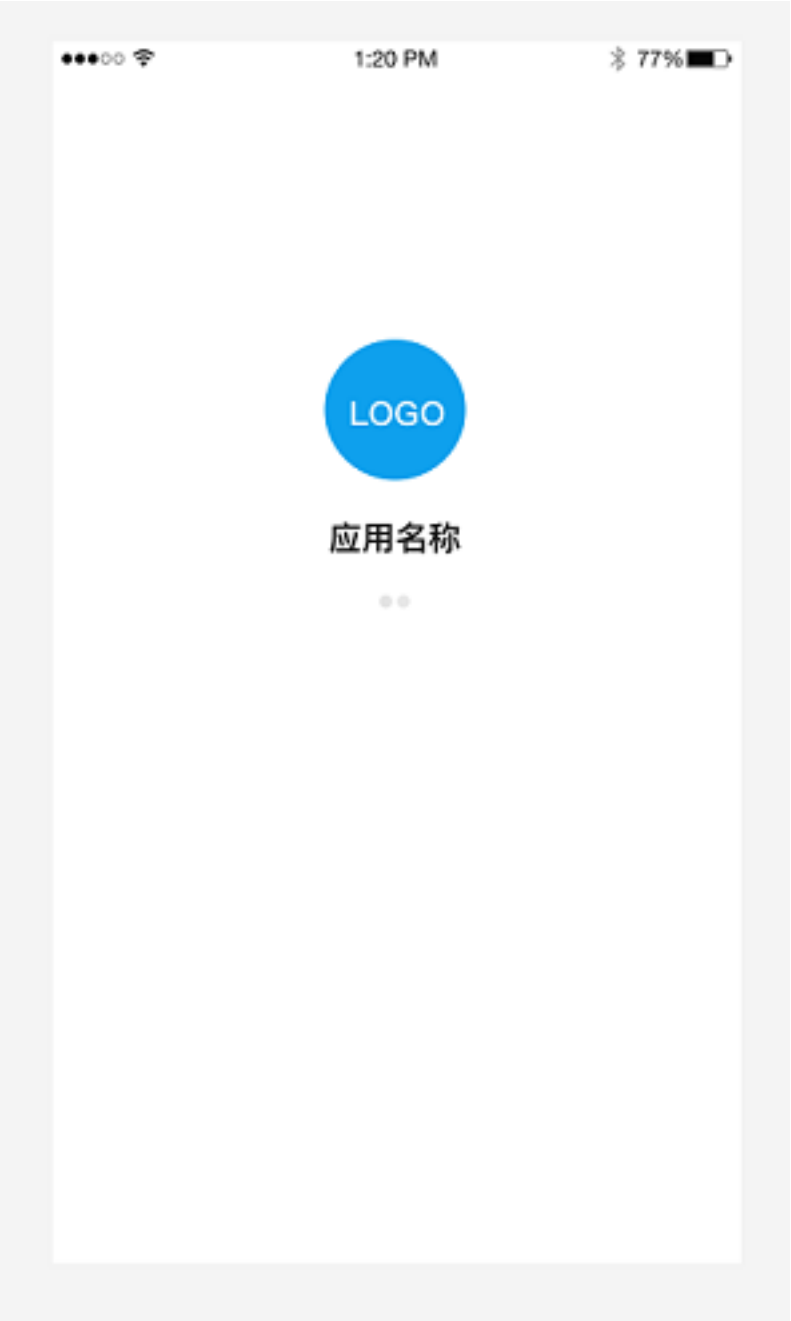
错误示例 1：用户主动关闭收银台，会弹出对话框让用户确认是否退出，显得十分多余。

错误示例 2：在服务窗中，删除某个服务窗的时候会展示删除成功的 toast。此时页面状态已经发生明显的变化，这个 toast 完全没有必要。

启动页加载

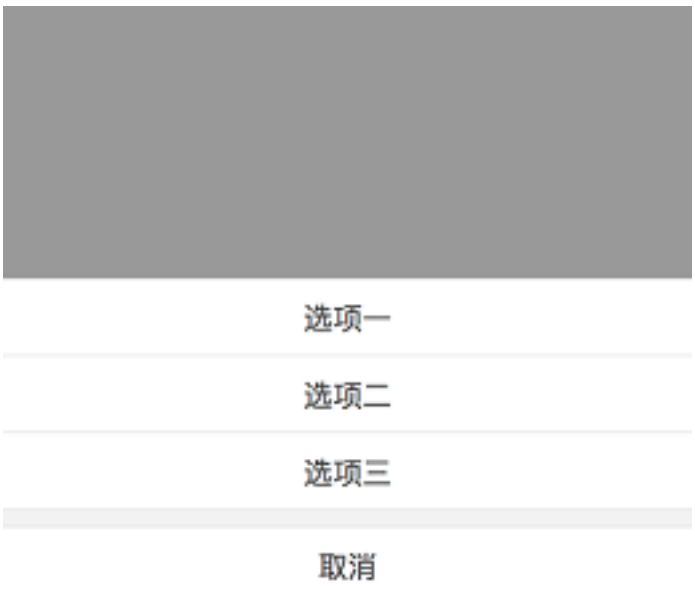
在应用程序中，启动页是小程序在一定程度上展现品牌特征的页面之一。

本页面将突出展示小程序品牌特征和加载状态。启动页除品牌标志（Logo）展示外，页面上的其他所有元素如加载进度指示，均由 mPaaS 统一提供且不能更改，无需开发者开发。



操作面板（ActionSheet）

操作面板是一种特殊类型的弹出框，集合展示多个操作选项。



定义及原则

- 操作面板给用户提供多个操作选项进行选择。
- 操作面板的最大高度是限定的，最多不能超过 5 个操作选项。
- 操作面板是一种特殊形式的弹出框，它出现的时候页面同时会盖上一层半透明的蒙层。

活动指示器（ActivityIndicator）

加载组件将页面内容的加载过程可视化，减轻用户的等待感。

定义及原则

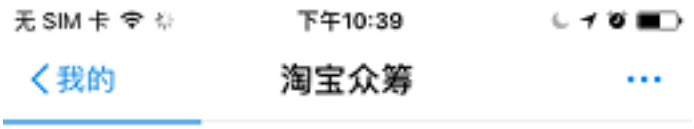
当用户进入一个新页面或提交了某个操作，页面的加载需要用户等待时，我们应该用进度条告知用户加载的进度。否则，用户的等待会变的茫然和漫长，从而愤然离开你的页面。

我们有直线进度条和圆形进度条两种样式，并且分别对样式进行了统一的定义。

示例

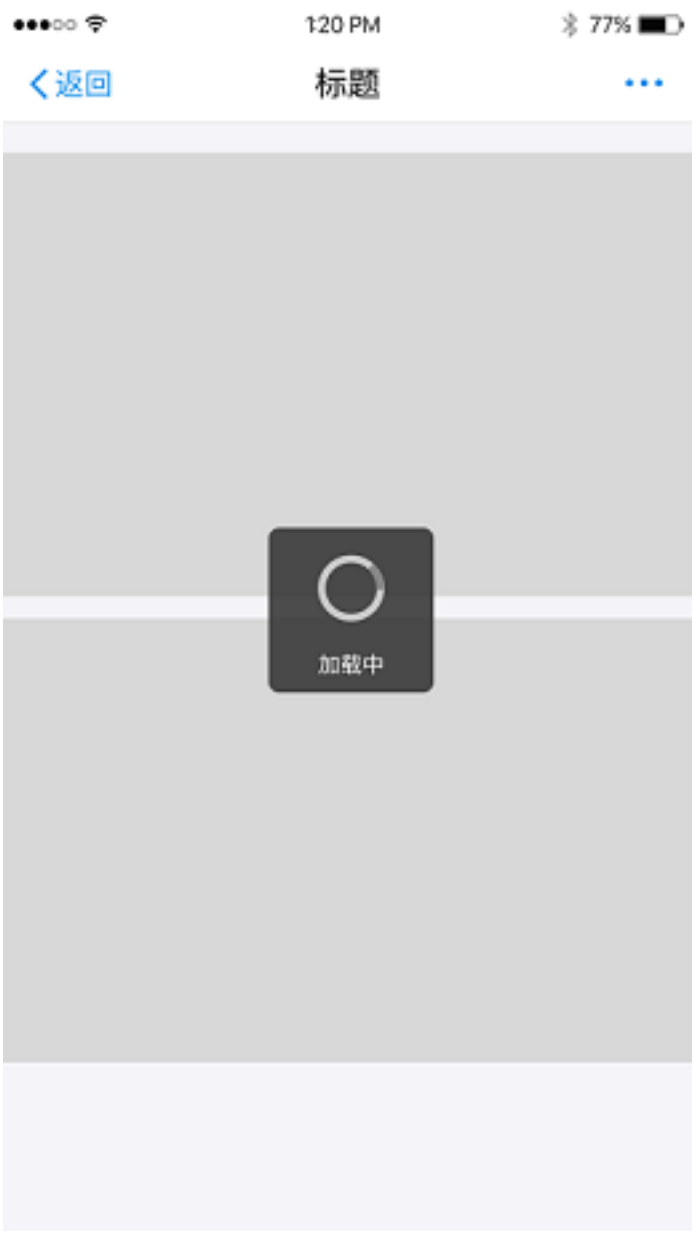
页面启动加载进度条：

用户使用 mPaaS 框架加载一个全新的在线页面的时候，导航栏的下方会出现一个直线形页面加载的进度条，展示当前页面内容加载的进度。加载进度条由 mPaaS 统一提供且不能更改，无需开发者开发。



模态加载：

当页面提交内容以后需要等待时使用模态加载，模态的加载样式将覆盖整个页面。由于无法明确告知具体加载的位置或内容，模态加载可能引起用户的焦虑感，因此应谨慎使用。除了在某些全局性操作下以外，尽量不要使用模态的加载。

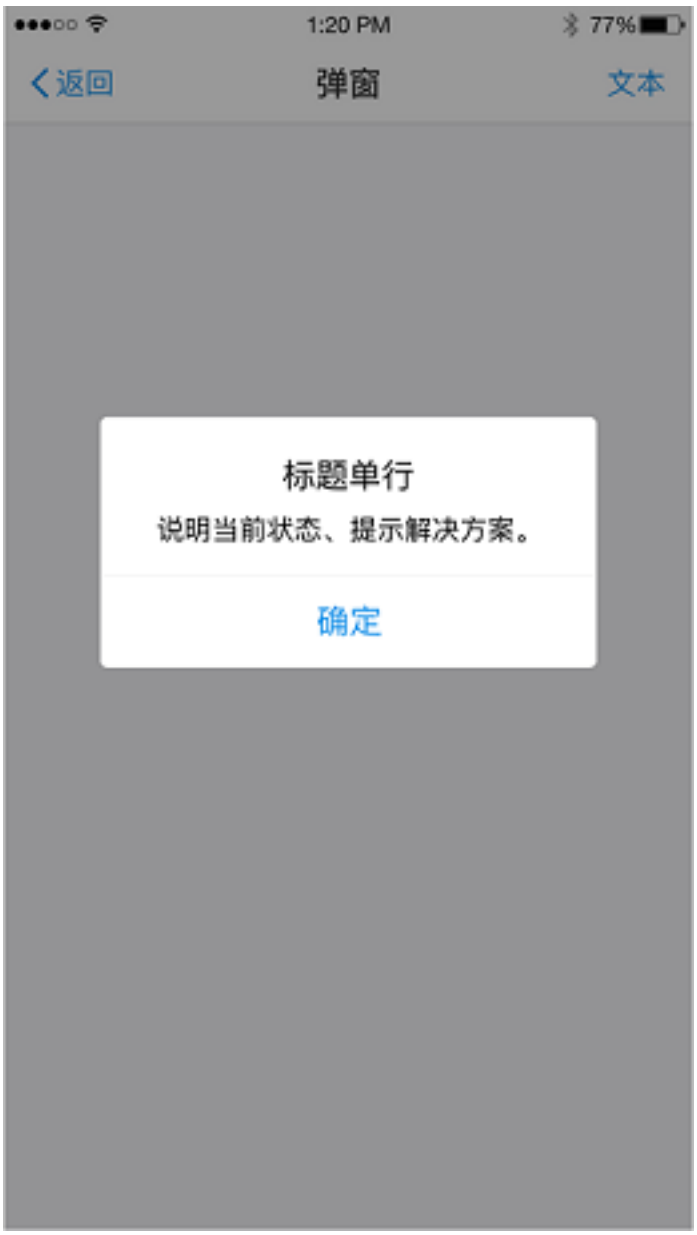


局部加载：

局部加载反馈即只在触发加载的页面局部进行反馈，这样的反馈机制更加有针对性，页面跳动小，是mPaaS 推荐的反馈方式。

对话框（Modal）

对话框出现的目的应该是告知用户必须知道的重要信息，或者让用户做出必要的选择。



定义及原则

对话框由一两句说明文字和操作按钮组成，有两种用法：

1. 确认和取消重要操作（如是否删除内容）；
2. 告知用户非常重要的信息（如出现严重错误）。通常会用加粗的颜色，突出显示可能造成用户损失的操作项（比如，“删除”、“不保存”、“取消”）。

对话框一定要慎用。必须是用户非知道不可，或者是用户必须亲自做决策的内容才使用对话框。否则，我们应选用 toast 等其他较弱的反馈方式。

视觉样式

标准的对话框由标题、正文、行动选项三部分组成。特殊类型的对话框可以加入图片和插画，更好的向用户传递信息。


标题单行
说明当前状态、提示解决方案。
确定
标题单行
说明当前状态、提示用户解决方案，最好不要超过两行。
确定
无标题弹框
描述信息，采用标题大小
取消
确定

示例

警告和报错

当用户的操作会产生不可挽回的损失时，应该出现二次确认的对话框，提示用户注意。

删除二次确认：

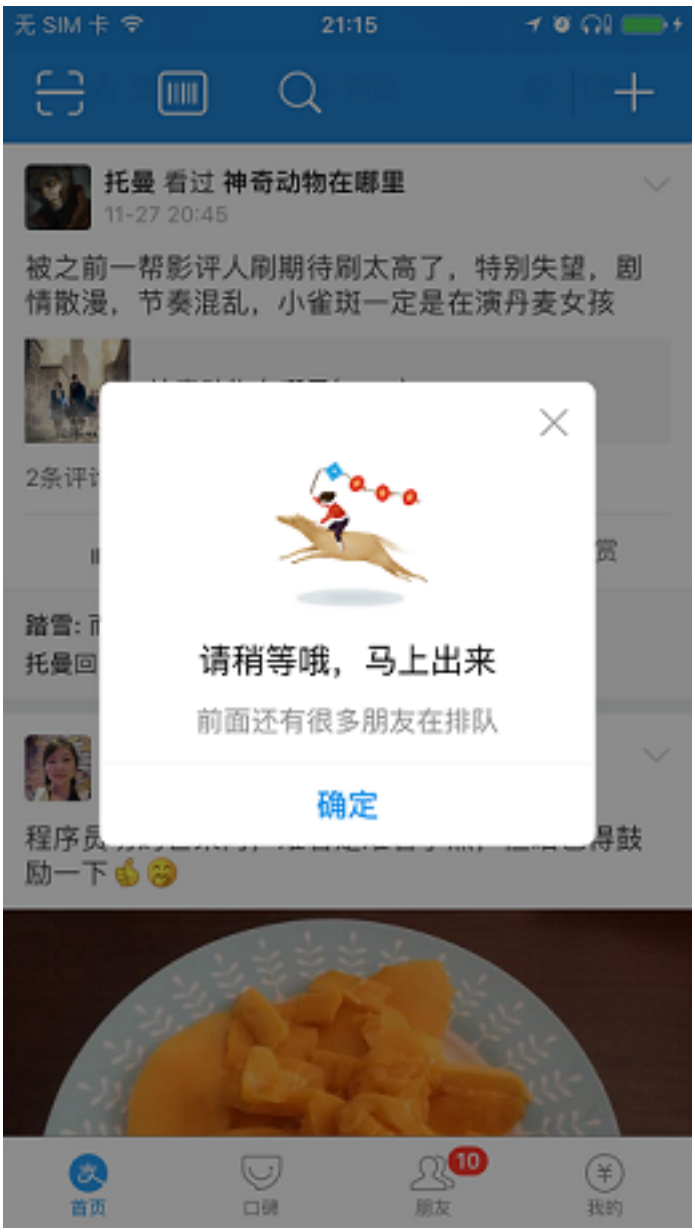
当用户的操作发生错误时，对话框告知用户错误的原因以及接下来应该怎么做。

表单提交出错：

带插图的特殊对话框

某些场景下你可能希望对话框加上图形的传达更佳的生动活泼，此时可使用带插图的特殊对话框。


中国联通
下午2:50
100%
杭州
必胜客
美食
快餐小吃
火锅
水果/零食
休闲娱乐
外卖
电影
开启地址位置权限
为你推荐身边精选商家
有优有惠，口碑带你吃喝玩乐。
去开启
大牌快抢
每天10点限量疯抢
火热进行中
旅游版
纽约国际儿童俱乐部 一节早教课程（...
首页
口碑
朋友
我的



轻提示（Toast）

Toast 是一种比较轻量的操作反馈或者提示信息，它其实是一种弱化版的对话框。它就像气泡一样，在界面上展示短暂的时间（ 1.5 秒或 3 秒），然后自动消失。它不强制用户做任何操作和确认，所以对用户的打扰比对话框小。

Toast 一般用来确认用户执行的任务状态或者操作结果，也可以向用户提示一些轻量的信息，如：网络异常、加载失败等。

定义及原则

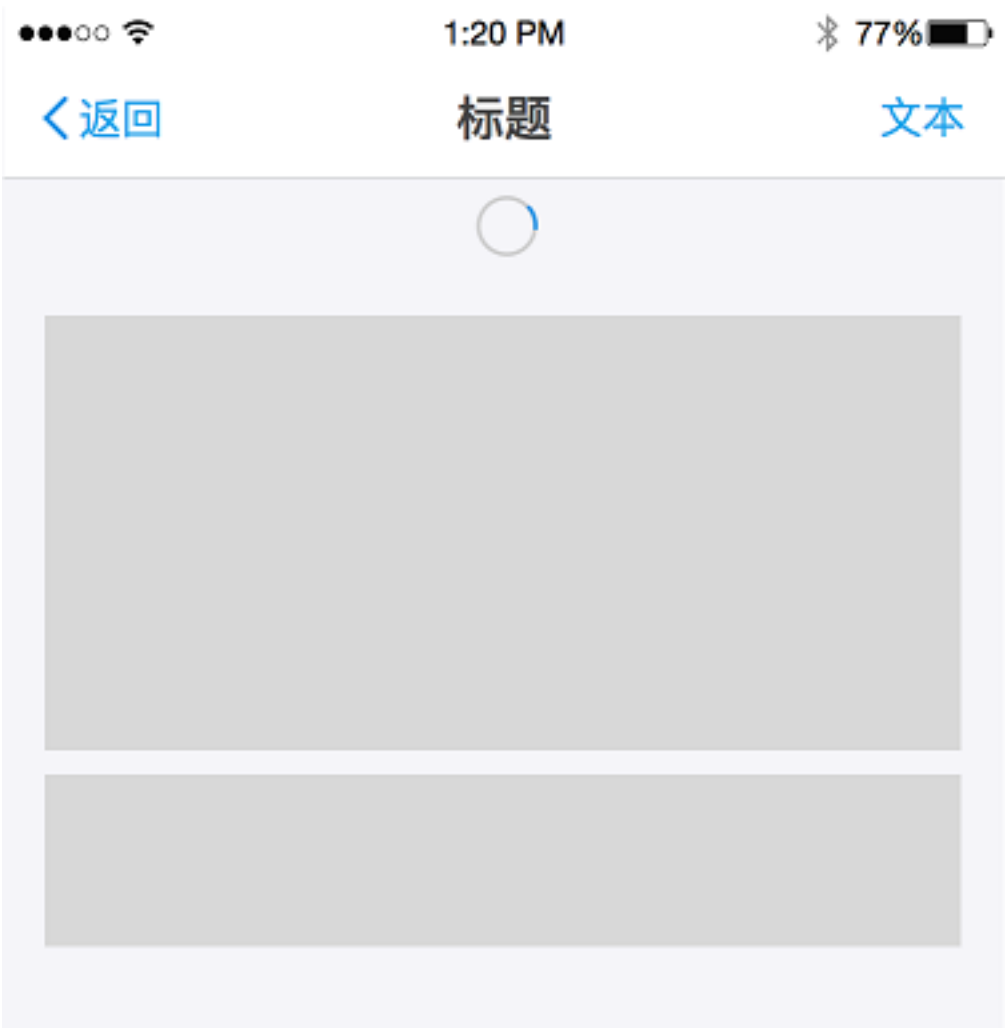
- 页面能及时看到状态变化的，不能再使用 Toast 提示，以免造成过度反馈。
- Toast 加载属于阻断式加载，也要少用，尽量用其他方式代替。
- Toast 显示时间较短，文案最多只能展示 15 个字符。

12.3.5 手势

手势作为隐藏快捷操作可以提升用户的操作效率。使用手势操作可以极大的提升用户的使用体验。iOS 和 Android 系统已经向用户普及了哪些手势代表什么操作，而您只要遵循和支持系统已有的操作就能提升用户的体验。

下拉刷新

通过下拉手势触发页面数据的刷新。



用户通过下拉页面的手势触发客户端向服务器请求数据更新，服务器在接到请求后，反馈给客户端最新的页面数据。

mPaaS 小程序框架提供标准的页面下拉刷新加载能力和样式。您可自定义需要通过下拉交互完成页面刷新。对于此类交互，mPaaS 小程序将提供标准能力和样式。

12.3.6 平台差异性设计

在 iOS 和 Android 两个平台上，人机交互的差异较大，我们应该遵循平台特性，做差异化的设计。

搜索

搜索分为 **凸显式** 和 **隐蔽式** 两种。

- 凸显式搜索用在支付宝首页、行业平台首页等，有强搜索需求，或者有营销引导需求的页面；
- 隐蔽式搜索用在搜索需求不是特别强烈的页面，省出搜索栏的空间，可以展示更多内容。

在 iOS 平台中，凸显式与隐蔽式搜索的设计如下：

在 Android 平台中，凸显式与隐蔽式搜索的设计如下：

操作列表

iOS 系统的操作列表从页面底部滑出，并且在列表底部有 **取消** 按钮，用来关闭列表；

Android 系统的操作列表从页面中间弹出，由于 Android 设备都有物理返回按钮，因此无需在列表中设计取消或关闭按钮。点击列表之外的空白区域，或者点击物理返回键可关闭列表。

弹出框

iOS、Android 平台的弹出框样式有区别，但交互方式与使用原则均相同：

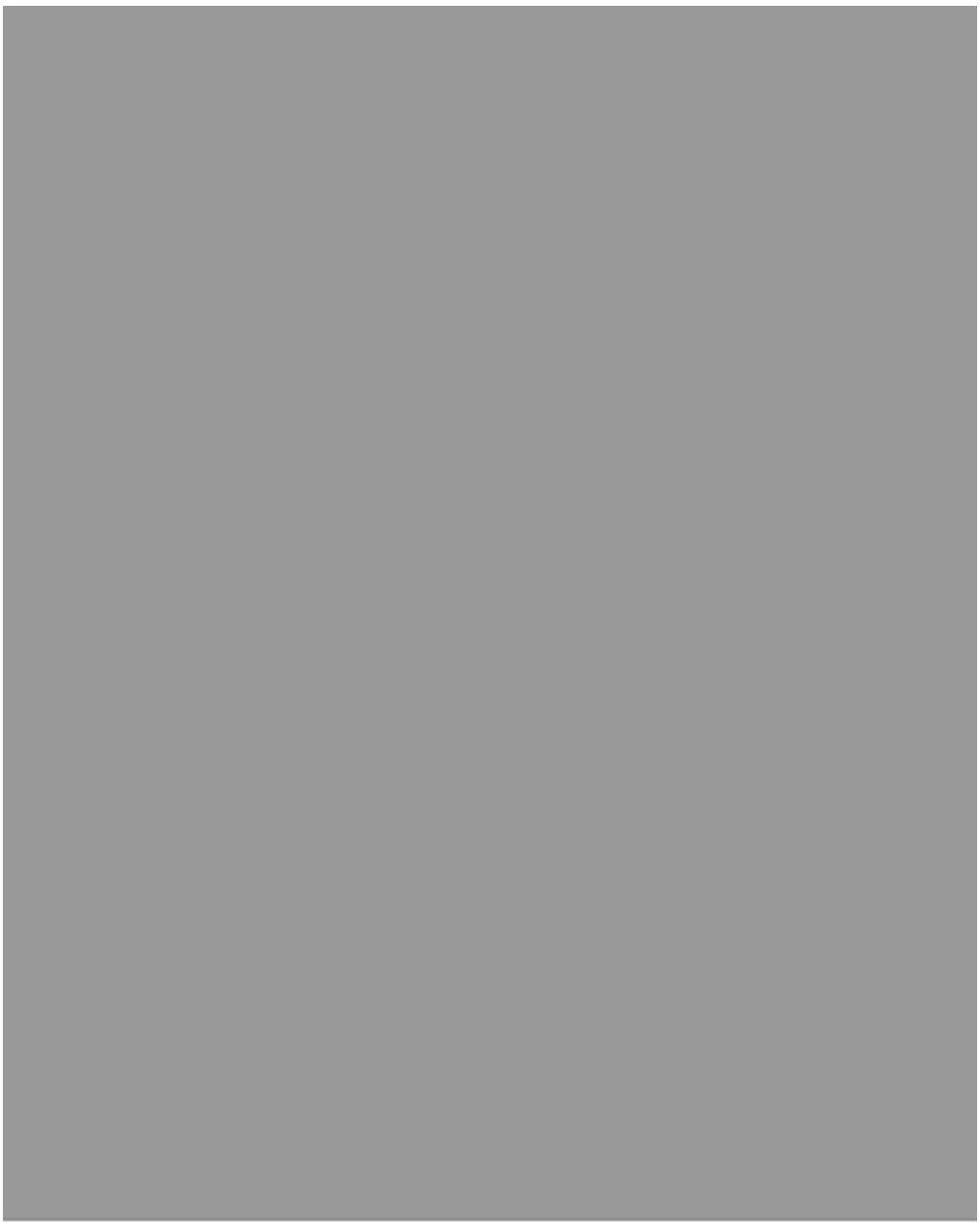
原则：

1. 在标题中询问是否执行当前操作；
2. 如果有必要，在正文解释当前操作造成的后果；
3. 确定执行的按钮上面重申操作动作。

禁忌：

1. 不要使用模糊不清的描述，如：“你确定？”；
2. 不要对操作后果进行解释和阐述；
3. 不要使用指意不明的行动按钮，如：按钮上只有“取消”和“确定”，“取消”按钮有时候会造成歧义。

iOS 弹出框如下：



选项一

选项一

选项一

取消

Android 弹出框如下：


正文正文正文正文正文正文正文正文正文正文正文
选项
正文正文正文正文正文正文正文正文正文正文正文
选项
选项
标题文字文字文字文字文字
正文正文正文正文正文正文正文正文正文正文正文
选项
选项
正文正文正文正文正文正文正文正文正文正文正文
选项一
点击选项二
选项三

12.3.7 组件组合

通过不同组件的组合，您可以拼接出一些通用模式的页面。

结果页


1:20 PM
77%
标题
完成
验证成功
所提交内容已成功完成验证
主要操作引导

定义及原则

完成某个任务后，需要明确告知用户任务的当前的状态。所以页面中可以展示操作成功的状态，或者任务当前处于什么状态、还需要等待多久。

后续操作

结果页除了展现任务最终的状态，还可以在下面引导用户进行相关的其他操作。

示例

结果页分为两种样式：

任务成功：


●●●●○ 中国电信 4G
下午9:15
10%
结果详情
完成
转出成功
成功转出1.00元至支付宝账户余额。
余月宝
钱要闲得更有价值
10元起投，稳逐收益！

任务某阶段成功，下一阶段还需要等待：


●●●●○ 中国电信 4G
下午9:15
10%
信用卡还款
完成
付款成功，银行处理中
1.00元
招商银行(尾号9695) 郭一贤
还款成功
今天到账(最晚24:00)
银行延后1-2日更新还款结果
蚂蚁会员专享
刮888元现金大奖
点我刮奖
详情查询>>
持续还款有助提升芝麻信用分

13 代码示例

请参考 代码示例 。

14 发布小程序

完成小程序开发与上传后，您可以到 **mPaaS 控制台** > **实时发布** > **小程序包管理** 发布小程序。更多信息参见小程序包管理 。


蚂蚁金服
ANT FINANCIAL
金融科技
FINANCIAL TECHNOLOGY