蚂蚁科技

蚂蚁动态卡片
使用指南

文档版本：20231226

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

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

商标声明

蚂蚁集团 ANT GROUP及其他蚂蚁集团相关的商标均为蚂蚁集团所有。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。蚂蚁集团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，并在蚂蚁集团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集团授权渠道下载、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。

通用约定

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

目录

1.蚂蚁动态卡片简介 08
2.接入客户端 10
2.1. 接入流程简介 10
2.2. 接入 Android 10
2.2.1. 快速开始 10
2.2.2. 进阶指南 13
2.2.2.1. 扩展初始化参数 13
2.2.2.2. 渲染卡片 14
2.2.2.3. 真机预览 16
2.2.2.4. 初始化引擎扩展 17
2.2.2.5. 预置卡片 21
2.2.2.6. 卡片调用客户端方法 22
2.2.2.7. 客户端调用卡片方法 23
2.2.2.8. 卡片自定义标签 24
2.2.3. 接口说明 26
2.2.3.1. CubeService 26
2.2.3.2. CubeEngine 27
2.2.3.3. CubeEngineConfig 30
2.2.3.4. CExceptionListener 32
2.2.3.5. CExceptionType 33
2.2.3.6. CExceptionInfo 33
2.2.3.7. CubeCard 35
2.2.3.8. CubeCardConfig 38
2.2.3.9. CCardType 44
2.2.3.10. CCardLayoutChangeListener 44
2.2.3.11. CCardCallback 45

2.2.3.12. CubeCardResultCode 45
2.2.3.13. CubeView 46
2.2.3.14. CubeModuleModel 46
2.2.3.15. CubeModule 47
2.2.3.16. CubeJSCallback 47
2.3. 接入 iOS 47
2.3.1. 快速开始 47
2.3.2. 进阶指南 50
2.3.2.1. 渲染卡片 50
2.3.2.2. 真机预览 51
2.3.2.3. 初始化引擎扩展 56
2.3.2.4. 预置卡片 58
2.3.2.5. 卡片调用客户端方法 60
2.3.2.6. 客户端调用卡片方法 61
2.3.2.7. 卡片自定义标签 62
2.3.3. 接口说明 64
2.3.3.1. CubeService 64
2.3.3.2. CubeEngine 65
2.3.3.3. CubeEngineConfig 67
2.3.3.4. CExceptionListener 67
2.3.3.5. CExceptionType 67
2.3.3.6. CExceptionInfo 68
2.3.3.7. CubeCard 68
2.3.3.8. CubeCardConfig 69
2.3.3.9. CCardType 70
2.3.3.10. CCardLayoutChangeListener 71
2.3.3.11. CCardCallback 71
2.3.3.12. CubeModuleProtocol 71

2.3.3.13. CubeCardResultCode 71
3.使用控制台 73
3.1. 卡片管理 73
3.1.1. 创建卡片 73
3.1.2. 添加卡片资源 73
3.1.3. 删除卡片 74
3.1.4. 创建发布任务 75
3.2. 卡片分析 76
3.3. 卡片性能 77
4.开发工具 80
4.1. 关于 AntCubeTool 80
4.2. 安装 AntCubeTool 80
4.3. 使用 AntCubeTool 80
4.4. 更新 AntCubeTool 84
4.5. 卸载 AntCubeTool 84
5.卡片语法 85
5.1. 卡片基础 85
5.1.1. 工程管理 85
5.1.2. 模板写法 86
5.1.3. 单位 86
5.1.4. 数据绑定 96
5.1.5. 事件绑定 96
5.1.6. 逻辑渲染 97
5.2. 卡片组件 98
5.2.1. div 98
5.2.2. text 99
5.2.3. image 105
5.2.4. richtext 106

5.2.5. slider 110
5.2.6. scroller 112
5.3. 卡片样式 115
5.3.1. 样式语法 115
5.3.2. 通用样式 119
5.3.2.1. 背景 119
5.3.2.2. filter 123
5.3.2.3. 盒子模型 125
5.3.2.4. 布局 131
5.3.2.5. hover 134
5.3.2.6. 动画 135
5.3.2.7. 无障碍 141
5.4. JS 能力 142
5.4.1. JS 能力 142
5.4.2. 生命周期 143
5.4.3. 定时器 144
5.4.4. JS API 145
5.4.4.1. dom 145
5.4.4.2. animation 147
5.4.5. keyframe 动画 148
6.常见问题 156

1.蚂蚁动态卡片简介

本文从定义和产品优势两方面对蚂蚁动态卡片（Ant Cube Card）进行了详细介绍，同时也提供了系列视频以深入了解蚂蚁动态卡片。

产品定义

蚂蚁动态卡片最初是支付宝自研的 Native 高性能渲染引擎，可以以不同产品形态在输入产物、JS 动态能力支持、渲染数据结构、样式支持能力等方面提供多种功能加持，并支持对增强功能进行组合以运用于不同场景。伴随着新业务场景对渲染能力以及外围能力的要求，蚂蚁动态卡片逐渐外扩为“一种以高性能渲染引擎为基础的应用开发技术栈”。而“兼顾用户体验和研发效率，追求极致的性能”，则成为了蚂蚁动态卡片的技术目标。

产品优势

- **提供动态内容展示，提高开发&运营效率**
  - 以卡片的形式嵌入到原生 Native 页面中
  - Android/iOS 双端具有一致性，开发效率高，发布即可见
  - 体积小、性能好、内存少
- **经过支付宝钱包业务深度打磨**
  - 多种前端开发语言（精简 Vue）
  - 完善的开发调试工具（编译、预览、调试、发布）
  - 客户端 SDK + 服务端卡片管理系统

介绍视频

基础介绍

您可以参照下面表格中提供的时间戳，快速定位到具体内容。

内容	时间戳
蚂蚁动态卡片的定义和特点	00:00:53~00:01:23
应用框架	00:01:24~00:03:01
系统架构	00:03:02~00:05:23
核心模块	00:05:24~00:07:03
线程模型	00:07:03~00:09:03
数据模型	00:09:04~00:11:26
高性能并发渲染	00:11:27~00:13:28
卡片生产/工作流程	00:13:29~00:14:15
卡片开发调试	00:14:16~00:15:09

操作演示

在以下视频中对卡片的创建、预览、运行效果进行了实操演示。

2.接入客户端

2.1. 接入流程简介

本文介绍了将蚂蚁动态卡片接入客户端的整体流程。

在使用蚂蚁动态卡片时，需要遵循以下流程：

1. 选择定制基线。
   i. 添加卡片基线。
   ii. 添加卡片组件。
2. 初始化卡片。
3. 构建卡片工程。
   i. 初始化工程。
   ii. 构建工程。
4. 发布卡片。
   i. 进入卡片后台。
   ii. 创建卡片。
   iii. 添加卡片资源。
   iv. 发布卡片。
5. 渲染卡片。

更多参考

- 接入 Android
- 接入 iOS

2.2. 接入 Android

2.2.1. 快速开始

本文介绍了接入蚂蚁动态卡片的流程和部分使用蚂蚁动态卡片的内容。

蚂蚁动态卡片已通过 mPaaS 10.2.3 基线版本 开始公测，但当前仅支持采用 mPaaS 原生 AAR 的接入方式。更多关于接入方式的信息，请参见 接入方式简介 。

> **说明**
>
> 如果有更多接入相关问题，欢迎搜索群号 32843812 加入钉钉群进行咨询交流。该钉钉群已添加 mPaaS 公有云答疑小助手，能够快速回答常见接入问题。更多关于使用公有云答疑小助手的信息，请参见 公有云答疑小助手 。

前置条件

- 已经开通并接入 mPaaS。
- 已经安装蚂蚁动态卡片 AntCubeTool 工具。更多详情请参见 关于 AntCubeTool。

操作步骤

1. 选择基线。
   i. 点击 **mPaaS** > **原生 AAR 接入**，在弹出的接入面板中，点击接入/升级基线下的 **开始配置**，添加 10.2.3 基线。
   ii. 添加 **魔方卡片** 和 **魔方卡片-网络图片加载库** 组件。
2. **初始化卡片。**

i. **在不同的基线版本中，需要进行的初始化处理不同，请根据您的基线版本，进行相应操作。**

a. **在 10.2.3 及以后的基线版本中，只需在 Application 中添加 `MP.init(this);` 初始化 mPaaS 即可。如您需要设置初始化参数，请参见 扩展初始化参数。**

b. **在 10.2.3 之前的基线版本中，需在 Application 中添加以下代码，初始化 mPaaS。**

```
public class MainApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        // 初始化 mPaaS
        MP.init(this);
    }
}
```

ii. **在 App 目录下 `AndroidManifest.xml` 中添加以下代码，以保证可以抓到 C 层闪退。**

```
<!-- 蚂蚁动态卡片抓取 C 层闪退必备 -->
<receiver
    android:name="com.alipay.mobile.common.logging.process.LogReceiverInToolsProcess"
    android:enabled="true"
    android:exported="false"
    android:process=":tools">
    <intent-filter>
        <action android:name="${applicationId}.monitor.command" />
    </intent-filter>
</receiver>
<receiver
    android:name="com.alipay.mobile.logmonitor.ClientMonitorWakeupReceiver"
    android:enabled="true"
    android:exported="false"
    android:process=":push">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <action android:name="${applicationId}.push.action.CHECK" />
        <action android:name="${applicationId}.monitor.command" />
    </intent-filter>
</receiver>
```

3. **构建卡片工程。**

i. **初始化工程。在终端里执行 `act init` 命令。**

- **请选择应用类型为 Cube ，选择模板卡片（VUE 格式）。**
- **请输入应用名称，输入您的项目名称，建议采用英文、数字和下划线的组合。**
- **请选择“需要额外创建工程源码文件夹”，会额外以应用名称建立一个文件夹。如果选择不需要，会在当前目录直接初始化。**

ii. **构建工程。使用 `cd` 命令打开刚创建的卡片工程，运行 `act build` 完成构建。构建完的产物会在您的工程的 `/dist/` 文件夹下。**

4. **发布卡片。**

i. **进入卡片后台。**

ii. **单击 新建卡片。**

卡片 ID 建议使用英文、数字和下划线的组合，长度不短于 8 个字符。客户端渲染卡片时会依赖卡片 ID。卡片名称可以取任意值，长度不超过 20 个字符。

iii. **添加卡片资源。**

- **建议使用 4 位版本号。**
- **选择刚才编译的 main.zip。**
- **客户端范围指的是可以拉取到该卡片的客户端版本。如果要覆盖所有客户端版本，在最低版本中填入 0.0.0.0 即可。**

iv. **发布卡片。**

a. **单击 创建发布。**

b. **选择 正式发布。**

卡片发布成功后，客户端就可以拉取到卡片。

5. **渲染卡片。**

```
// 创建卡片配置
CubeCardConfig cardConfig = new CubeCardConfig();
// 后台创建的卡片 ID
cardConfig.setTemplateId("hello_cube");
// 卡片版本
cardConfig.setVersion("1.0.0.0");
// 卡片宽度，这里选取屏幕宽度
cardConfig.setWidth(MFSystemInfo.getPortraitScreenWidth());
// 卡片数据（用于渲染卡片的数据，一般是 mock.json 里面的内容）
JSONObject obj = new JSONObject("xxxxx");
cardConfig.setData(obj);
// 创建卡片信息
CubeService.instance().getEngine().createCard(cardConfig, new CCardCallback() {
    @Override
    public void onLoaded(final CubeCard crystalCard, CCardType cardType, CubeCardConfig cubeCard
Config, CubeCardResultCode result
        if (resultCode == CubeCardResultCode.CubeCardResultSucc) {
            // 需要运行在主线程
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // 创建卡片 View
                    CubeView view =
CubeService.instance().getEngine().createView(FastActivity.this);
                    // 添加到外层 ViewGroup 里
                    mWrapperLl.addView(view);
                    // 渲染卡片
                    crystalCard.renderView(view);
                }
            });
        } else {
            MPLogger.info("cube", "fail " + cubeCardConfig.getTemplateId() + " style " +
cardType + " error " + resultCode);
        }
    }
});
```

销毁页面时，需要把卡片手动回收。

```
@Override
protected void onDestroy() {
    super.onDestroy();
    if (mCard != null) {
        mCard.recycle();
    }
    int chidrenCount = mWrapperLl.getChildCount();
    for (int i = 0; i < chidrenCount; i++) {
        if (mWrapperLl.getChildAt(i) instanceof CubeView) {
            ((CubeView) mWrapperLl.getChildAt(i)).destroy();
        }
    }
    mWrapperLl.removeAllViews();
}
```

6. 效果预览。

2.2.2. 进阶指南

2.2.2.1. 扩展初始化参数

在 `cp_change_23596.28` 及以后版本的基线中，使用卡片前无需手动初始化，只需要安装卡片组件即可。本文则介绍了在此种情况下设置初始化参数的实现方法。

如果您需要设置初始化参数，请参考以下代码：

```
// 设置 cube 初始化参数
CubeInitParam cubeInitParam
    = CubeInitParam.getDefault()
    // 引擎初始化配置
    .setCubeEngineConfig(generateCubeEngineConfig())
    // 注册卡片 to 客户端通道
    .setCubeModuleModels(generateModuleModel())
    // 注册自定义标签
    .setCubeWidgetInfos(generateWidget());
// 初始化 mPaaS
MP.init(this, MPInitParam.obtain().addComponentInitParam(cubeInitParam));
```

setAutoInitCube

```
/**
 * 设置是否框架自动初始化 cube，默认自动
 * @param autoInitCube
 * @return
 */
public CubeInitParam setAutoInitCube(boolean autoInitCube)
```

setCubeEngineConfig

```
/**
 * 设置 CubeEngineConfig，可参考 CubeEngineConfig 介绍
 * @param cubeEngineConfig
 * @return
 */
public CubeInitParam setCubeEngineConfig(CubeEngineConfig cubeEngineConfig)
```

setCubeModuleModels

```
/**
 * 注册 cube jsapi
 * @param cubeModuleModels
 */
public CubeInitParam setCubeModuleModels(Collection<CubeModuleModel> cubeModuleModels)
```

setCubeWidgetInfos

```
/**
 * 注册自定义 view（自定义标签）
 * @param cubeWidgetInfos
 */
public CubeInitParam setCubeWidgetInfos(Collection<CubeWidgetInfo> cubeWidgetInfos)
```

2.2.2.2. 渲染卡片

本文介绍了在 Android 客户端渲染卡片的整体实现流程。

渲染卡片的流程分为四部分，第一步，组装卡片配置信息；第二步，根据配置信息请求卡片，获取到卡片实例；第三步，通过卡片实例，使用卡片 View 去渲染；第四步，在整个业务完成后，在 `destroy` 声明周期中，释放卡片。具体流程如下：

1. **组装卡片配置信息。**

   创建配置信息，并设置各种参数。更多参数请参见 接口说明。

```
/**
 * 组装卡片配置信息
 * @return
 */
private CubeCardConfig assembleCubeCardConfig(){
    // 创建卡片配置
    CubeCardConfig cardConfig = new CubeCardConfig();
    // 后台创建的卡片ID
    cardConfig.setTemplateId("hello_cube");
    // 卡片版本
    cardConfig.setVersion("1.0.0.0");
    // 卡片宽度，这里选取屏幕宽度
    cardConfig.setWidth(MFSystemInfo.getPortraitScreenWidth());
    return cardConfig;
}
```

2. **请求卡片。**

   根据组装好的卡片配置信息，请求卡片。卡片引擎会去服务端获取卡片模板信息，可以使用 `createCard` 方法一次请求一个卡片，也可以使用 `createCards` 方法一次请求多个卡片。

```
/**
 * 请求卡片信息
 * @param cardConfig
 */
private void requestCubeCard(final CubeCardConfig cardConfig){
    // 创建卡片信息
    CubeService.instance().getEngine().createCard(cardConfig, new CCardCallback() {
        @Override
        public void onLoaded(final CubeCard cubeCard, CCardType cardType, CubeCardConfig cubeCar
dConfig, CubeCardResultCode resultCode) {
            renderCubeCard(cubeCard,cardType,cubeCardConfig,resultCode);
        }
    });
}
```

3. **渲染卡片。**

   获取到卡片信息后，生成卡片 View，在主线程中进行渲染。这一步也要做异常判断，防止未顺利获取到卡片信息的情况发生。

```
/**
 * 渲染卡片
 * @param cubeCard
 * @param cardType
 * @param cubeCardConfig
 * @param resultCode
 */
private void renderCubeCard(final CubeCard cubeCard, CCardType cardType, CubeCardConfig cubeCard
Config, CubeCardResultCode resultCode) {
    if (resultCode == CubeCardResultCode.CubeCardResultSucc) {
        // 需要运行在主线程
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                mCard = cubeCard;
                // 创建卡片View
                CubeView view =
CubeService.instance().getEngine().createView(FastActivity.this);
                // 添加到外层 ViewGroup 里
                mWrapperLl.addView(view);
                // 渲染卡片
                cubeCard.renderView(view);
            }
        });
        MPLogger.info(TAG, "succ " + cubeCardConfig.getTemplateId() + " style " + cardType);
    } else {
        MPLogger.info(TAG, "fail " + cubeCardConfig.getTemplateId() + " style " + cardType + " e
rror " + resultCode);
    }
}
```

4. **释放卡片。**

   卡片使用完成之后，释放卡片的内存资源，通常是在页面的 `onDestroy` 生命周期里调用。

```
/**
 * 释放卡片资源
 */
private void releaseCubeCard(){
    if (mCard != null) {
        mCard.recycle();
    }
    int chidrenCount = mWrapperLl.getChildCount();
    for (int i = 0; i < chidrenCount; i++) {
        if (mWrapperLl.getChildAt(i) instanceof CubeView) {
            ((CubeView) mWrapperLl.getChildAt(i)).destroy();
        }
    }
    mWrapperLl.removeAllViews();
}
```

2.2.2.3. 真机预览

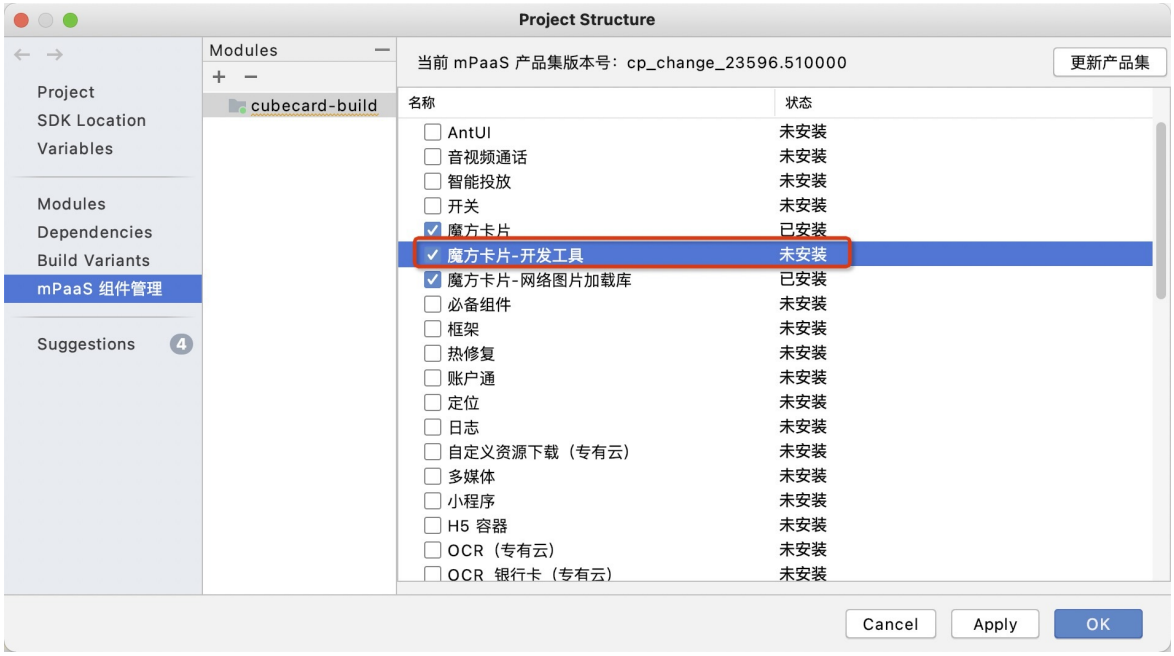
本文介绍了在 Andoid 客户端中进行真机预览卡片的操作流程。

前置条件

- 已经开通并接入 mPaaS。
- 已经安装蚂蚁动态卡片 AntCubeTool 工具。更多详情请参见 关于 AntCubeTool。
- 已经按照 快速开始 完成接入流程。

操作步骤

1. **添加真机预览依赖。**
   i. **添加 蚂蚁动态卡片-开发工具 组件。**



ii. 在项目主 module 的 `build.gradle` 中添加第三方依赖，如果有冲突可以以您的依赖版本为准。

```
dependencies {
    ......
        implementation "com.squareup.okhttp3:logging-interceptor:3.12.12"
    implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.72'
    implementation "com.squareup.okhttp3:okhttp:3.12.12"
    implementation 'com.squareup.picasso:picasso:2.5.2'
    implementation 'org.simple:androideventbus:1.0.5'
    .......
}
```

2. 通过命令行启动本地调试服务。在工程的路径下，运行指令开启服务。在 macOS 和 Windows 上开启服务的指令如下：
   - macOS: `act prepare && act server`
   - Windows: `act prepare | act server`

   执行指令后，在终端会生成二维码。
3. 启动客户端并扫码，建立连接。终端会提示设备已连接。

```
CubeCardDebug.openScanner(activity);
```

> **说明**
> 由于是建立内网连接，因此如果 targetVersion 大于 27，需要进行降级，或者在 manifest 里配置 `android:usesCleartextTraffic="true"android:networkSecurityConfig="@xml/network_security_config"` 以及对应的 `network_security_config.xml` 。
>
> ```
> <?xml version="1.0" encoding="utf-8"?>
> <network-security-config>
>     <domain-config cleartextTrafficPermitted="true">
>         <domain includeSubdomains="true">${本地 IP 地址}</domain>
>     </domain-config>
> </network-security-config>
> ```

扫码完成之后，在客户端上会有 toast 提醒：Cube Socket： 已连接

终端也会提示已连接。

4. **预览。**

修改卡片代码，然后调用 `act build` 完成编译。再调用 `act preview` ，将编译好的内容推送到客户端上。

2.2.2.4. 初始化引擎扩展

本文介绍了在 Android 客户端中初始化引擎扩展的操作步骤。

前置条件

- 已经开通并接入 mPaaS。
- 已经在控制台 发布蚂蚁动态卡片 。

操作步骤

1. **预置卡片本地路径。**
   - **设置卡片引擎类 `CubeEngineConfig` 的属性。**

```
/**
* 存储模版的本地资源包的路径
*
* @param resourcePath - 设置资源文件路径
*/
public void setResourcePath(String resourcePath) {
                this.resourcePath = resourcePath;
}
```

- 本地蚂蚁动态卡片 assets 路径。

```
//清除卡片数据
CubeService.instance().destroyEngine();
CubeEngineConfig config = new CubeEngineConfig();
//设置路径
config.setResourcePath("您设置的资源所在路径");
CubeService.instance().initEngine(config, MainApplication.getApplication());
```

示例：如果您设置 `setResourcePath("cube")` **，那么预置卡片路径就是** `assets/cube/` **。**

2. **设置异常监听。**

i. **设置异常监听接口** `CExceptionListener` **。**

```
public interface CExceptionListener {
    void onException(CExceptionInfo var1);
}
```

ii. **异常接口方法描述。**

限定符和类型	方法和说明
void	onException(CExceptionInfo info)

iii. **异常方法详细说明。**

```
/**
* 异常监听通知
*
* @param cExceptionInfo - 异常信息
*/
void onException(CExceptionInfo cExceptionInfo)
```

iv. **在** `Activity` **中接入** `CExceptionListener` **类，执行继承方法。**

```
public class MainActivity extends AppCompatActivity implements CExceptionListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onException(CExceptionInfo cExceptionInfo) {
                                //根据 CExceptionInfo 返回数据判断异常接口错误信息
    }
}
```

3. **设置网络图片下载 Handler。**

i. **网络图片下载接口 `ICKImageHandler` 。**

ii. **在 `ICKImageHandler` 内部定义 `onBitmapLoaded` 和 `onBitmapFailed` 。方法如下：**

```
public interface ICKImageHandler {

    String loadImage(String var1, int var2, int var3, Map<String, Object> var4,
ICKImageHandler.LoadImageListener var5);

    void cancel(String var1);

    public interface LoadImageListener {
        void onBitmapLoaded(Bitmap var1);
        void onBitmapFailed(Exception var1);
    }
}
```

iii. **图片下载方法描述。**

限定符和类型	方法和说明
String	loadImage(String var1, **int** var2, **int** var3, Map<String, Object> var4, ICKImageHandler.LoadImageListener var5)
void	cancel(String var1)
void	onBitmapLoaded(Bitmap var1)
void	onBitmapFailed(Exception var1)

iv. 图片下载方法详细描述。

```
/**
* 网络图片下载监听
*
* @param url - 图片下载网络地址 URL
* @param width - 图片宽度
* @param height - 图片高度
* @param params - 参数
* @param loadImageListener - 加载监听
*/
String loadImage(String url, int width, int height, Map<String, Object> params, ICKImageHandle
r.LoadImageListener loadImageListener)


/**
* 网络图片下载取消监听
*
* @param var1 - URL
*/
void cancel(String var1)


/**
* 图片下载成功监听
*
* @param var1
*/
void onBitmapLoaded(Bitmap var1)


/**
* 图片下载失败监听
*
* @param var1
*/
void onBitmapFailed(Exception var1)
```

v. 在 `Activity` 中接入 `ICKImageHandler` 类，执行继承方法，重写网络图片下载功能。

```
public class LocalCardsActivity extends AppCompatActivity implements ICKImageHandler {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_local_cards);

    }

    @Override
    public String loadImage(String s, int i, int i1, Map<String, Object> map,
LoadImageListener loadImageListener) {
        new Thread(){
            @Override
            public void run() {
                try {
                    final Bitmap bitmap = HttpUtil.get(url, null, null);
                    if (bitmap != null){
                        if (loadImageListener != null){
                            loadImageListener.onBitmapLoaded(bitmap);
                        }
                    }else {
                        if (loadImageListener != null){
                            loadImageListener.onBitmapFailed(new Exception("bitmap is null"));
                        }
                    }
                }catch (Exception e){
                    e.printStackTrace();
                    if (loadImageListener != null){
                        loadImageListener.onBitmapFailed(e);
                    }
                }
            }
        }.start();
        return url;
    }

    @Override
    public void cancel(String s) {

    }
}
```

2.2.2.5. 预置卡片

本文介绍了在 Android 客户端中预置蚂蚁动态卡片的操作步骤。

前置条件

- 已经开通并接入 mPaaS。
- 已经在控制台 发布蚂蚁动态卡片 。
- 已经在初始化引擎扩展中预置卡片本地路径。更多信息，请参见 初始化引擎扩展 。

操作步骤

1. **从控制台下载加密的 Bin 文件。**
2. **将下载的文件添加拷贝到资源文件中。**

   将下载的 Bin 文件重新命名为 卡片 ID@卡片版本，例如 main@1_0_0_0.zip。

> 说明
>
> **后期控制台将直接提供完整命名的 zip 包，无需用户手动修改命名，只需要添加到 assets 即可。**

3. 将 Assets 引入工程中。更多详情，请参见 初始化引擎扩展 。
4. 加载预置卡片。

```
// 创建卡片配置
CubeCardConfig cardConfig = new CubeCardConfig();
// 创建的卡片 ID
cardConfig.setTemplateId("main");
// 卡片版本
cardConfig.setVersion("1.0.0.0");
// 卡片宽度，这里选取屏幕宽度
cardConfig.setWidth(MFSystemInfo.getPortraitScreenWidth());
// 卡片数据．（必填）
JSONObject obj = new JSONObject("xxxxx");
cardConfig.setData(obj);
// 创建卡片信息
CubeService.instance().getEngine().createCard(cardConfig, new CCardCallback() {
    @Override
    public void onLoaded(CubeCard cubeCard, CCardType cCardType, CubeCardConfig cubeCardConfig,
CubeCardResultCode cubeCardResultCode) {
        if (cubeCardResultCode == CubeCardResultCode.CubeCardResultSucc) {
            // 需要运行在主线程
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // 创建卡片 View
                    CubeView view = CubeService.instance().getEngine().createView(您当前 Activity
所在);
                    // 添加到外层 ViewGroup 里
                    local_cards.addView(view);
                    // 渲染卡片
                    cubeCard.renderView(view);
                }
            });
        } else {
                        MPLogger.info("cube", "fail " + cubeCardConfig.getTemplateId() + " style
" + cCardType + " error " + cubeCardResultCode);
        }
    }
});
```

2.2.2.6. 卡片调用客户端方法

本文介绍了在蚂蚁动态卡片中调用 Android 客户端方法的实现路径。

操作步骤

1. **客户端注册 module。**

i. **自定义 module。**

```
public class CustomCubeModule extends CubeModule {
    private static final String TAG = CustomCubeModule.class.getSimpleName();

    // 注解，uiThread 表示是否在主进程回调
    @JsMethod(uiThread = true)
    public void cubeToClient(final CubeJSCallback callback) {
        // 向卡片发送回调
        if (callback != null) {
            callback.invoke("cubeToClient callback data: " + System.currentTimeMillis());
        }
    }
}
```

ii. **注册自定义 module。**

第一个参数是 type，第二个参数是自定义 module 的全路径，第三个参数是调用的方法名称。全路径和方法名称不需要混淆。

```
Collection<CubeModuleModel> cubeModuleModels = new LinkedList<>();
cubeModuleModels.add(new CubeModuleModel("custom", CustomCubeModel.class.getName(), new String
[]{"cubeToClient"}));
CubeService.instance().getEngine().registerModule(cubeModuleModels, null);
```

2. **卡片侧调用。**

i. **注册 module。type 需和客户端保持一致，此处以** `custom` **为例。**

ii. **调用 module，方法名和客户端注册方法名要保持一致。**

```
<script>
    // 注册 module
    const navigator = requireModule("custom");
    export default {
        data: {
            message: 'Hello Cube 1'
        },
        beforeCreate() {
            this.message = 'Hello Cube 2'
        },
        didAppear() {

        },
        methods: {
            onClick() {
                // 调用客户端方法
                navigator.cubeToClient(event=>{
                    this.message = event;
                })
            }
        }
    }
</script>
```

iii. **将卡片打包发布到后台，即可调用客户端方法。**

2.2.2.7. 客户端调用卡片方法

本文介绍了在 Android 客户端调用卡片方法的实现路径。

操作步骤

1. **卡片侧实现 JS 方法。**

```
<script>
    export default {
        data: {
            message: 'Hello Cube 1'
        },
        beforeCreate() {
            this.message = 'Hello Cube 2'
        },
        didAppear() {

        },
        methods: {
            onClick() {
                console.info('invoke on-click event');
            },
            // js 方法
            clientToCube(data) {
                this.message = data;
            }
        }
    }
</script>
```

2. **将卡片打包，发布到卡片后台。**
3. **客户端调用。**

   客户端获取卡片实例，调用对应的 JS 方法，并发送数据。

```
if (mCubeCard != null) {
    mCubeCard.callJsFunction("clientToCube", "client to cube: " + System.currentTimeMillis());
}
```

2.2.2.8. 卡片自定义标签

本文介绍了在卡片中自定义标签的实现步骤。

操作步骤

1. **客户端注册自定义标签（自定义 View）。**
   i. **实现自定义标签（自定义 View）。**

   继承 CCardWidget 并实现其抽象方法，主要方法为 `onCreateView` **，在这里返回对应的自定义 View。**

```
public class CustomCubeWidget extends CCardWidget {
    private static final String TAG = "CustomCubeWidget";

    public CustomCubeWidget(Context context) {
        super(context);
    }

    /**
     * UI创建接口，表示当前组件要上屏的视图UI
     *
     * @param params 组件在卡片上声明的数据，包括样式、事件和属性，对应的key为DATA_KEY_STYLES、
DATA_KEY_EVENTS、DATA_KEY_ATTRS;
     * @param width  组件的宽度
     * @param height 组件的高度
     * @return 返回嵌入的View
     */
    @Override
```

```
    @Override
    public View onCreateView(Map<String, Object> params, int width, int height) {
        MPLogger.debug(TAG, "onCreateView:" + width + "," + height);
        for (String key : params.keySet()) {
            MPLogger.debug(TAG, key + ":" + params.get(key));
        }
        TextView textView = new TextView(getContext());
        textView.setText("test");
        return textView;
    }

    /**
     * UI复用接口，如果组件支持复用（参见 canReuse 方法），当组件被复用重新上屏时调用。
     *
     * @param params 组件在卡片上声明的数据
     * @param width  组件的宽度
     * @param height 组件的高度
     */
    @Override
    public void onReuse(Map<String, Object> params, int width, int height) {
        MPLogger.debug(TAG, "onReuse");
    }

    /**
     * 组件数据更新接口
     *
     * @param params 组件在卡片上声明的数据
     */
    @Override
    public void onUpdateData(Map<String, Object> params) {
        MPLogger.debug(TAG, "onUpdateData");
    }

    /**
     * 组件复用清理接口。如果组件支持复用（canReuse 返回 true），则组件在进入复用池后会调用
onRecycleAndCached方法，表示当前组件已经离屏放入缓存内，需要清理资源。
     */
    @Override
    public void onRecycleAndCached() {
        MPLogger.debug(TAG, "onRecycleAndCached");
    }

    /**
     * 组件是否支持复用；为了提高效率，扩展组件可以支持复用。例如当某个自定义的标签组件由于数据更新被移除当前视
图，此时该组件如果支持复用，那么会放入复用池内，下次该组件显示时，会直接从复用池内获取；如果组件不支持复用，则直
接调用销毁接口(onDestroy)；
     *
     * @return true: 复用，false: 不复用
     */
    @Override
    public boolean canReuse() {
        MPLogger.debug(TAG, "canReuse");
        return false;
    }

    /**
     * 组件销毁回调接口，释放资源
     */
    @Override
    public void onDestroy() {
        MPLogger.debug(TAG, "onDestroy");
```

```
    }
}
```

ii. 注册自定义标签。

第一个参数是自定义的标签，可随意定制，在卡片侧会写在 <> 内。建议加上前缀，防止和动态卡片本身的标签冲突。第二个是自定义标签实现类的全路径，注意不要混淆。

```
Collection<CubeWidgetInfo> widgetInfos = new LinkedList<>();
widgetInfos.add(new CubeWidgetInfo("custom-widget", CustomCubeWidget.class.getName()));
CubeService.instance().getEngine().registerWidgets(widgetInfos);
```

2. 卡片侧调用。

在标签中，写入自定义的标签，此处为 `custom-widget` **。**

> **说明**
> **写入的自定义标签需要和注册自定义标签步骤中的第一个参数保持一致。**

```
<template>
    <div class="root">
        ...
        <custom-widget></custom-widget>
        ...
    </div>
</template>
```

3. 将卡片打包发布到后台或者预览，即可渲染客户端自定义的 view。

2.2.3. 接口说明

2.2.3.1. CubeService

本文介绍了卡片服务类的方法。

公共函数

函数	返回值类型	说明
`public static CubeCrystalService instance()`	static CubeCrystalService	获取 CubeCrystalService 实例。
`public void initEngine(CubeEngineConfig config, Application application)`	void	全局初始化。
`public CubeEngine getEngine()`	CubeEngine	获取引擎实例。
`public void destroyEngine()`	void	全局销毁。

instance

- **声明：** `public static CubeCrystalService instance()`

- **说明：**获取 CubeCrystalService 实例。
- **参数：**无。
- **返回值：**

返回值	类型
CubeCrystalService 实例	CubeCrystalService

initEngine

- **声明：** `public void initEngine(CubeEngineConfig config, Application application)`
- **说明：**全局初始化。
- **参数：**

参数	类型	说明
config	CubeEngineConfig	初始化配置
application	Application	-

- **返回值：**无。

getEngine

- **声明：** `public CubeEngine getEngine()`
- **说明：**获取引擎实例。
- **参数：**无。
- **返回值：**

返回值	类型
引擎实例	CubeEngine

destroyEngine

- **声明：** `public void destroyEngine()`
- **说明：**全局销毁。
- **参数：**无。
- **返回值：**无。

2.2.3.2. CubeEngine

本文介绍了卡片引擎核心类的方法。

公共函数

函数	返回值类型	说明
createCard(final CubeCardConfig config, final CCardCallback callback)	void	创建单个卡片。

createCards(List<CubeCardConfig> configs, final CCardCallback callback)	void	批量创建卡片。
createView(Context context)	CubeView	创建渲染视图。
setCustomUnit(String unitName, float unitRadio)	void	设置自定义单位。
registerModule(Collection<CubeModuleModel> models, Bundle options)	void	注册自定义 module。
registerWidgets(Collection<CubeWidgetInfo> widgets)	void	注册一组自定义扩展组件。
sendEvent(Map<String, Object> componentData, String eventName, @Nullable Map<String, Object> eventParams)	void	Native 向 JS 侧发送自定义事件通道。
destroy()	void	销毁卡片实例。

createCard

- **声明：** `public void createCard(final CubeCardConfig config, final CCardCallback callback)`
- **说明：** 创建单个卡片。
- **参数：**

参数	类型	说明
config	CubeCardConfig	卡片配置参数。
callback	CCardCallback	回调。

- **返回值：** 无。

createCards

- **声明：** `public void createCards(List<CubeCardConfig> configs, final CCardCallback callback)`
- **说明：** 批量创建卡片。
- **参数：**

参数	类型	说明
configs	List<CubeCardConfig>	批量配置。
callback	CCardCallback	回调，每个卡片结果回调一次。

- **返回值：** 无。

createView

- **声明：** `public CubeView createView(Context context)`
- **说明：** 创建渲染视图。

- **参数：**

参数	类型	说明
context	Context	创建 view 的上下文。

- **返回值：**

返回值	类型	说明
createView	CubeView	Cube 渲染容器 view。

setCustomUnit

- **声明：** `public void setCustomUnit(String unitName, float unitRadio)`
- **说明：设置自定义单位。**
- **参数：**

参数	类型	说明
unitName	String	单位名称，例如 sip。
unitRadio	float	单位比例，例如 1.5。

- **返回值：无。**

registerModule

- **声明：** `public void registerModule(Collection<CubeModuleModel> models, Bundle options)`
- **说明：注册自定义 module。**
- **参数：**

参数	类型	说明
models	Collection<CubeModuleModel>	key 为 module 名称，例如 animation；value 为类名，例如 CKAnitmationModule。
options	Bundle	-

- **返回值：无。**

registerWidgets

- **声明：** `public void registerWidgets(Collection<CubeWidgetInfo> widgets)`
- **说明：注册一组自定义扩展组件。**
- **参数：**

参数	类型	说明

widgets	Collection<CubeWidgetInfo>	扩展组件信息。

- **返回值：无。**

sendEvent

- **声明：** `public void sendEvent(Map<String, Object> componentData, String eventName, @Nullable Map<String, Object> eventParams)`
- **说明：** Native 向 JS 侧发送自定义事件通道。
- **参数：**

参数	类型	说明
componentData	Map<String, Object>	组件数据，即 CCardWidget 中 onCreateView 创建组件时的入参 data。
eventName	String	自定义事件名称。
eventParams	Map<String, Object>	自定义事件参数。

- **返回值：无。**

destroy

- **声明：** `private void destroy()`
- **说明：** 销毁卡片实例。
- **参数：** 无。
- **返回值：** 无。

2.2.3.3. CubeEngineConfig

本文介绍了卡片引擎初始化配置类的方法。

公共函数

函数	返回值类型	说明
`public String getResourcePath()`	String	获取本地资源包路径。
`public void setResourcePath(String resourcePath)`	void	设置本地资源包路径。
`public CExceptionListener getExceptionListener()`	CExceptionListener	获取异常监听。

函数	返回值类型	说明
`public void setExceptionListener(CExceptionListener exceptionListener)`	void	设置异常监听。
`public ICKImageHandler getImageHandler()`	ICKImageHandler	获取 imagehandler。
`public void setImageHandler(ICKImageHandler imageHandler)`	void	设置 imageHandler。

getResourcePath

- **声明：** `public String getResourcePath()`
- **说明：获取本地资源包路径。**
- **参数：无。**
- **返回值：**

返回值	类型	说明
getResourcePath 对象	String	本地资源包路径。

setResourcePath

```
/**
 * 设置本地资源包路径
 * @param resourcePath
 */
public void setResourcePath(String resourcePath)
```

- **声明：** `public void setResourcePath(String resourcePath)`
- **说明：设置本地资源包路径。**
- **参数：**

参数	类型	说明
resourcePath	String	本地资源包路径。

- **返回值：无。**

getExceptionListener

- **声明：** `public CExceptionListener getExceptionListener()`
- **说明：获取异常监听。**
- **参数：无。**
- **返回值：**

返回值	类型	说明
getExceptionListener 对象	CExceptionListener	监听到的异常。

setExceptionListener

- **声明：** `public void setExceptionListener(CExceptionListener exceptionListener)`
- **说明：** 设置异常监听。
- **参数：**

参数	类型	说明
exceptionListener	CExceptionListener	卡片异常监听器。

- **返回值：** 无。

getImageHandler

- **声明：** `public ICKImageHandler getImageHandler()`
- **说明：** 获取 imagehandler。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
imagehandler 对象	ICKImageHandler	获取到的 imagehandler。

setImageHandler

- **声明：** `public void setImageHandler(ICKImageHandler imageHandler)`
- **说明：** 设置 imageHandler。如果不设置，会默认内部实现。建议通过自定义实现。
- **参数：**

参数	类型	说明
imageHandler	ICKImageHandler	要设置的 imageHandler。

- **返回值：** 无。

2.2.3.4. CExceptionListener

本文介绍了卡片异常监听器的方法。

公共方法

函数	返回值类型	说明
`public void onException(CExceptionInfo info);`	void	异常监听通知。

onException

- **声明：** `public void onException(CExceptionInfo info);`
- **说明：** 异常监听通知。
- **参数：**

参数	类型	说明
info	CExceptionInfo	异常信息

- **返回值：** 无。

2.2.3.5. CExceptionType

本文介绍了卡片的异常类型。

- **声明：** `public enum CExceptionType {JS_EXCEPTION, STYLE_EXCEPTION}`
- **说明：** 卡片异常类型。
- **枚举值**

枚举值	说明
JS_EXCEPTION	JS 异常
STYLE_EXCEPTION	样式异常

2.2.3.6. CExceptionInfo

本文介绍了卡片异常信息类的方法。

公共函数

函数	返回值类型	说明
`public String getCardUid()`	String	获取卡片实例 ID。
`public CExceptionType getErrCode()`	CExceptionType	获取卡片异常类型。

函数	返回值类型	说明
`public String getTitle()`	String	获取异常标题。
`public String getException()`	String	获取异常信息。
`public Map<String, Object> getExtParams()`	Map<String, Object>	获取异常扩展信息。

getCardUid

- **声明：** `public String getCardUid()`
- **说明：** 获取卡片实例 ID。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
卡片实例 ID 对象	String	卡片实例 ID

getErrCode

- **声明：** `public CExceptionType getErrCode()`
- **说明：** 获取卡片异常类型。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
卡片异常类型对象	CExceptionType	卡片异常类型

getTitle

- **声明：** `public String getTitle()`
- **说明：** 获取异常标题。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
异常标题对象	String	异常标题

getException

- **声明：** `public String getException()`
- **说明：** 获取异常信息。

- **参数：无。**
- **返回值：**

返回值	类型	说明
异常信息对象	String	获取到的异常信息

getExtParams

- **声明：** `public Map<String, Object> getExtParams()`
- **说明：获取异常扩展信息。**
- **参数：无。**
- **返回值：**

返回值	类型	说明
异常扩展信息对象	Map<String, Object>	获取到的异常扩展信息

2.2.3.7. CubeCard

本文介绍了卡片核心类的方法。

公共函数

函数	返回值类型	说明
renderView(CubeView view)	void	渲染视图，需要提供 CubeView。
getSize()	Rect	获取卡片宽高尺寸。
updateData(JSONObject jsonData)	void	更新渲染数据。
callJsFunction(final String methodName, final Object... params)	void	调用 JS 方法。
recycle()	void	销毁，回收资源。
getCardUid()	String	获取卡片实例 ID。
notifyState(CCardState state)	void	通知卡片状态。
getCubeCardConfig()	CubeCardConfig	获取创建卡片的 config 参数。
getBindView()	CubeView	获取卡片临时绑定的 view。

renderView

- **声明：** `public void renderView(CubeView view)`
- **说明：渲染视图，需要提供 CubeView。**
- **参数：**

参数	类型	说明
view	CubeView	由 CubeEngine 生成

- **返回值：无。**

getSize

- **声明：** `public Rect getSize()`
- **说明：获取卡片宽高尺寸。**
- **参数：无。**
- **返回值：**

类型	说明
Rect	卡片宽、高尺寸对象

updateData

- **声明：** `public void updateData(JSONObject jsonData)`
- **说明：更新渲染数据。**
- **参数：**

参数	类型	说明
jsonData	JSONObject	渲染卡片所需要的外部数据模型

- **返回值：无。**

callJsFunction

- **声明：** `public void callJsFunction(final String methodName, final Object... params)`
- **说明：调用 JS 方法。**
- **参数：**

参数	类型	说明
methodName	String	方法名
params	Object	调用参数

- **返回值：无。**

recycle

- **声明：** `public void recycle()`
- **说明：销毁，回收资源。**

- **参数：无。**
- **返回值：无。**

getCardUid

- **声明：** `public String getCardUid()`
- **说明：获取卡片实例 ID。**
- **参数：无。**
- **返回值：**

类型	说明
String	卡片实例 ID 对象

notifyState

- **声明：** `public void notifyState(CCardState state)`
- **说明：在卡片出屏、上屏、前后台时通知变更卡片状态。**
- **参数：**

参数	类型	说明
state	CCardState	卡片状态

- **返回值：无。**

getCubeCardConfig

- **声明：** `public CubeCardConfig getCubeCardConfig()`
- **说明：获取创建卡片的 config 参数。**
- **参数：无。**
- **返回值：**

类型	说明
CubeCardConfig	创建卡片的 config 对象。

getBindView

- **声明：** `public CubeView getBindView()`
- **说明：获取卡片临时绑定的 view。**

> **重要**
> view 可能被其他卡片复用。

- **参数：无。**
- **返回值：**

类型	说明

CubeView	绑定的 view 对象。

2.2.3.8. CubeCardConfig

本文介绍了卡片配置类的方法。

公共函数

函数	返回值类型	说明
`public String getTemplateId()`	String	获取模版唯一 ID。
`public CubeCardConfig setTemplateId(String templateId)`	CubeCardConfig	设置模版唯一 ID。
`public String getVersion()`	String	获取模版版本号。
`public CubeCardConfig setVersion(String version)`	CubeCardConfig	设置模版版本号。
`public String getCardUid()`	String	获取卡片唯一 ID。
`public JSONObject getData()`	JSONObject	获取卡片数据。
`public CubeCardConfig setData(JSONObject data)`	CubeCardConfig	设置卡片数据。
`public int getWidth()`	int	获取卡片预设的宽度。
`public CubeCardConfig setWidth(int width)`	CubeCardConfig	置卡片预设的宽度。
`public int getHeight()`	int	获取卡片预设的高度。
`public CubeCardConfig setHeight(int height)`	CubeCardConfig	设置卡片预设的高度。
`public JSONObject getExtOption()`	JSONObject	获取卡片扩展参数。

函数	返回值类型	说明
`public CubeCardConfig setExtOption(JSONObject extOption)`	CubeCardConfig	设置卡片扩展参数。
`public JSONObject getEnvData()`	JSONObject	获取卡片环境变量数据。
`public CubeCardConfig setEnvData(JSONObject envData)`	CubeCardConfig	设置卡片环境变量数据。
`public CCardLayoutChangeListener getLayoutChangeListener()`	CCardLayoutChangeListener	获取布局变更监听。
`public void setLayoutChangeListener(CCardLayoutChangeListener layoutChangeListener)`	void	设置布局变更监听。

getTemplateId

- **声明：** `public String getTemplateId()`
- **说明：** 获取模版唯一 ID。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
模版 ID 对象	String	模版唯一 ID

setTemplateId

- **声明：** `public CubeCardConfig setTemplateId(String templateId)`
- **说明：** 设置模版唯一 ID。
- **参数：**

参数	类型	说明
templateId	String	要设置的模版 ID

- **返回值：**

返回值	类型	说明
卡片配置参数对象	CubeCardConfig	卡片配置参数

getVersion

- **声明：** `public String getVersion()`
- **说明：** 获取模版版本号。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
模板版本号对象	String	模版版本号

setVersion

- **声明：** `public CubeCardConfig setVersion(String version)`
- **说明：** 设置模版版本号。
- **参数：**

参数	类型	说明
version	String	要设置的模板版本号

- **返回值：**

返回值	类型	说明
卡片配置参数对象	CubeCardConfig	卡片配置参数

getCardUid

- **声明：** `public String getCardUid()`
- **说明：** 获取卡片唯一 ID。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
卡片 ID 对象	String	卡片 ID

getData

- **声明：** `public JSONObject getData()`
- **说明：** 获取卡片数据。
- **参数：** 无。

- **返回值：**

返回值	类型	说明
卡片数据对象	JSONObject	卡片数据

setData

- **声明：** `public CubeCardConfig setData(JSONObject data)`
- **说明：设置卡片数据。**
- **参数：**

参数	类型	说明
data	JSONObject	卡片数据

- **返回值：**

返回值	类型	说明
卡片配置对象	CubeCardConfig	卡片配置

getWidth

- **声明：** `public int getWidth()`
- **说明：获取卡片预设的宽度。**
- **参数：无。**
- **返回值：**

返回值	类型	说明
卡片的宽度对象	int	卡片预设的宽度

setWidth

- **声明：** `public CubeCardConfig setWidth(int width)`
- **说明：置卡片预设的宽度。**
- **参数：**

参数	类型	说明
width	int	卡片宽度

- **返回值：**

返回值	类型	说明
卡片配置参数对象	CubeCardConfig	卡片配置参数

getHeight

- **声明：** `public int getHeight()`
- **说明：** 获取卡片预设的高度。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
卡片的高度对象	int	卡片预设的高度

setHeight

- **声明：** `public CubeCardConfig setHeight(int height)`
- **说明：** 设置卡片预设的高度。
- **参数：**

参数	类型	说明
height	int	卡片高度

- **返回值：**

返回值	类型	说明
卡片配置参数对象	CubeCardConfig	卡片配置参数

getExtOption

- **声明：** `public JSONObject getExtOption()`
- **说明：** 获取卡片扩展参数。
- **参数：** 无。
- **返回值：**

返回值	类型	说明
卡片扩展参数对象	JSONObject	卡片扩展参数

setExtOption

- **声明：** `public CubeCardConfig setExtOption(JSONObject extOption)`
- **说明：** 设置卡片扩展参数。
- **参数：**

参数	类型	说明
extOption	JSONObject	卡片扩展参数

- **返回值：**

返回值	类型	说明
卡片配置参数对象	CubeCardConfig	卡片配置参数

getEnvData

- **声明：** `public JSONObject getEnvData()`
- **说明：获取卡片环境变量数据。**
- **参数：无。**
- **返回值：**

返回值	类型	说明
卡片环境变量数据	JSONObject	卡片环境变量数据

setEnvData

- **声明：** `public CubeCardConfig setEnvData(JSONObject envData)`
- **说明：设置卡片环境变量数据。**
- **参数：**

参数	类型	说明
envData	JSONObject	卡片环境变量数据

- **返回值：**

返回值	类型	说明
卡片配置参数对象	CubeCardConfig	卡片配置参数

getLayoutChangeListener

- **声明：** `public CCardLayoutChangeListener getLayoutChangeListener()`
- **说明：获取布局变更监听。**
- **参数：无。**
- **返回值：**

返回值	类型	说明

返回值	类型	说明
布局变更监听对象	CCardLayoutChangeListener	布局变更监听器

setLayoutChangeListener

- **声明：** `public void setLayoutChangeListener(CCardLayoutChangeListener layoutChangeListener)`
- **说明：设置布局变更监听。**
- **参数：**

参数	类型	说明
layoutChangeListener	CCardLayoutChangeListener	布局变更监听器

- **返回值：无。**

2.2.3.9. CCardType

本文介绍了卡片模版来源的类型。

- **声明：** `public enum CCardType {C_CARD_TYPE_NONE, C_CARD_TYPE_CACHE, C_CARD_TYPE_BUNDLE, C_CARD_TYPE_FILE, C_CARD_TYPE_CLOUD}`
- **说明：卡片模版来源类型。**
- **枚举值**

枚举值	说明
C_CARD_TYPE_CACHE	样式异常
C_CARD_TYPE_BUNDLE	本地资源包中的模版
C_CARD_TYPE_FILE	本地 SD 卡中的模版
C_CARD_TYPE_CLOUD	云端模版

2.2.3.10. CCardLayoutChangeListener

本文介绍了布局变更通知监听器的方法。

公共函数

函数	返回值类型	说明
`void onLayout(Rect size)`	void	卡片创建回调类。

onLayout

- **声明：** `void onLayout(Rect size)`

- **说明：布局变更通知监听。**
- **参数：**

参数	类型	说明
size	Rect	变更后的卡片尺寸。

- **返回值：无。**

2.2.3.11. CCardCallback

本文介绍了卡片创建回调类的方法。

公共函数

函数	返回值类型	说明
`public void onLoaded(final CubeCard card, CCardType cardType, CubeCardConfig config, CubeCardResultCode errorCode)`	void	卡片创建回调类。

onLoaded

- **声明：** `public void onLoaded(final CubeCard card, CCardType cardType, CubeCardConfig config, CubeCardResultCode errorCode)`
- **说明：卡片创建回调类。**
- **参数：**

参数	类型	说明
card	CubeCard	卡片实例
cardType	CCardType	卡片模版来源类型
config	CubeCardConfig	卡片配置参数
errorCode	CubeCardResultCode	错误码

- **返回值：无。**

2.2.3.12. CubeCardResultCode

本文介绍了卡片结果码。

- **声明：** `public enum CubeCardResultCode {CubeCardResultSucc, CubeCardResultNetworkError, CubeCardResultParamError, CubeCardResultNotExist, CubeCardResultContentInvalid}`
- **说明：卡片模版来源类型。**
- **枚举值**

枚举值	说明
CubeCardResultSucc	卡片请求成功
CubeCardResultNetworkError	网络异常
CubeCardResultParamError	请求参数异常
CubeCardResultNotExist	卡片不存在
CubeCardResultContentInvalid	卡片内容无效

2.2.3.13. CubeView

本文介绍了卡片渲染承载类的方法。

公共函数

函数	返回值类型	说明
`public void destroy()`	void	释放资源，需要显式调用。

destroy

- **声明：** `public void destroy()`
- **说明：** 释放资源，需要显式调用。
- **参数：** 无。
- **返回值：** 无。

2.2.3.14. CubeModuleModel

本文介绍了 module 的描述信息。

公共方法

函数	返回值类型	说明
`public CubeModuleModel(String type, String fullClsName, String[] methods)`	无。	构造函数。

CubeModuleModel

- **声明：** `public CubeModuleModel(String type, String fullClsName, String[] methods)`
- **说明：** 构造函数。
- **参数：**

参数	类型	说明
type	String	module 名称，例如 animation。
fullClsName	String	module 类完整包路径。
methods	String[]	module 中声明的 JS methods。

- **返回值：无。**

2.2.3.15. CubeModule

自定义 module 必须继承该类。

2.2.3.16. CubeJSCallback

本文介绍了 CubeJSCallback 类的方法。

公共方法

函数	返回值类型	说明
`public void invoke(Object data)`	void	发送给卡片的 callback 数据

invoke

当卡片调用 native 时，通过此类，发送 callback 返回给卡片。

- **声明：** `public void invoke(Object data)`
- **说明：向卡片发送 callback 数据。**
- **参数：**

参数	类型	说明
data	Object	发送给卡片的 callback 数据

- **返回值：无。**

2.3. 接入 iOS

2.3.1. 快速开始

蚂蚁动态卡片目前支持 基于 mPaaS 框架接入、基于已有工程且使用 mPaaS 插件接入 和 基于已有工程且使用 CocoaPods 接入三种接入方式，本文介绍了采用上述方式接入蚂蚁动态卡片和使用蚂蚁动态卡片的过程。

前置条件

- 您已经接入工程到 mPaaS。更多信息，请参见以下内容：
  - 基于 mPaaS 框架接入
  - 基于已有工程且使用 mPaaS 插件接入

- 基于已有工程且使用 CocoaPods 接入

- 已经安装蚂蚁动态卡片 AntCubeTool 工具。更多详情请参见 关于 AntCubeTool。

> **说明**
>
> 如果有更多接入相关问题，欢迎搜索群号 32843812 加入钉钉群进行咨询交流。该钉钉群已添加 mPaaS 公有云答疑小助手，能够快速回答常见接入问题。更多关于使用公有云答疑小助手的信息，请参见 公有云答疑小助手 。

操作步骤

1. 选择定制基线。

   蚂蚁动态卡片目前在自定义基线 `cp_change_29273` 版本中提供。根据您采用的接入方式，请选择相应的添加方式。

   - 使用 mPaaSPlugin 插件。

     此方式适用于采用了 基于 **mPaaS** 框架接入或 基于已有工程且使用 **mPaaS** 插件接入 的接入方式。

     a. 单击 Xcode 菜单项 **Editor**> **mPaaS**> 编辑工程 > 升级基线，切换工程到 `cp_change_29273` 定制基线。如果 升级基线不可点，请确保工程配置已经导入，参考前置条件打开编辑工程页面。

     b. 升级基线后，单击 编辑模块> 修改模块，选择 **Cube** 卡片，保存后单击 开始编辑，即可完成添加。

   - 使用 cocoapods-mPaaS 插件。

     此方式适用于采用了 基于已有工程且使用 **CocoaPods** 接入 的接入方式。

     a. 在 Podfile 文件中，使用 **mPaaS_pod "mPaaS_Cube"**添加 Cube 卡片组件依赖。

     b. 执行 `pod install` 即可完成接入。

2. 初始化卡片。

   i. 编译 Xcode 工程。如果遇到头文件找不到的情况，请打开选项，将 `Allow Non-modular Includes In Framework Modules` 置为 `Yes` 。

   ii. 引入 Cube 卡片需要的库。

   ```
   #import <CubeCrystal/CubeEngine.h>
   #import <AntCube/CubeService.h>
   ```

   iii. 使用单例初始化卡片引擎。

   ```
   - (void)initEngie{
       static dispatch_once_t onceToken;
       NSString *mockBundlePath = [NSString stringWithFormat:@"%@/%@/crystal", [[NSBundle mainBun
   dle] resourcePath], @"MPCubeDemo.bundle"];
       dispatch_once(&onceToken, ^{
           CubeEngineConfig* config = [[CubeEngineConfig alloc] init];
           [config setBundlePath:mockBundlePath];
           [[CubeService sharedInstance] initWithConfig:config];
       });
   }
   ```

3. 构建一个卡片工程。

   i. 初始化工程。在终端里执行 `act init` 命令。

   - 请选择应用类型为 Cube ，选择模板卡片 (VUE 格式）。
   - 请输入应用名称，输入您的项目名称，建议采用英文、数字和下划线的组合。
   - 请不要选择“应用在支付宝客户端卡片业务场景”。
   - 请选择“需要额外创建工程源码文件夹”，会额外以应用名称建立一个文件夹。如果选择不需要，会在当前目录直接初始化。

   ii. 构建工程。使用 `cd` 命令打开刚创建的卡片工程，运行 `act build` 完成构建。构建完的产物会在您的工程的 `/dist/` 文件夹下。

4. 发布卡片。

i. **进入卡片后台。**

ii. **单击 新建卡片。**

卡片 ID 建议使用英文、数字和下划线的组合，客户端渲染卡片时会依赖卡片 ID。卡片名称可以取任意值。

iii. **添加卡片资源。**

- **建议使用 4 位版本号。**
- **选择刚才编译的 main.zip。**
- **客户端范围指的是可以拉取到该卡片的客户端版本。如果要覆盖所有客户端版本，在最低版本中填入 0.0.0.0 即可。**

iv. **发布卡片。**

a. **单击 创建发布。**

b. **选择 正式发布。**

卡片发布成功后，客户端就可以拉取到卡片。

5. **渲染卡片。**

```
//创建卡片对象
CubeCardConfig *cardConfig = [[CubeCardConfig alloc] init];
//配置卡片版本（必填），从控制台复制
[cardConfig setVersion:@"1.0.0.0"];
//配置卡片 ID（必填），从控制台复制
[cardConfig setTemplteId:@"20211118"];
//预设卡片宽度
[cardConfig setWidth:[UIScreen mainScreen].bounds.size.width];
//预设卡片高度
[cardConfig setHeight:350];
//设置卡片数据（必填）参数为业务 JSON 数据。
[cardConfig setData:@{}];
//加载卡片
[[[CubeService sharedInstance] getEngine] createCard:cardConfig callback:self];
```

卡片取 key :"iosImage"，会获取到 key :"iosImage" 对应的 value 值，如下代码所示：

```
 [self.cardConfig
setData:@{@"iosImage":@"https://img.zcool.cn/community/013edb5c7b5b1ca801213f269fc887.jpg@1280w_11
_100sh.jpg"}];
```

创建卡片后的代理方法需要遵循代理 `CCardCallback`，并且需要实现 `onLoaded:cardType:config:erroCode` 协议方法。

```
- (void)onLoaded:(CubeCard *)card cardType:(CCardType)cardType config:(CubeCardConfig *)config e
rroCode:(CubeCardResultCode)erroCode {
    //创建卡片失败
    if (!card) {
        NSLog(@"load card fail %@ style %d error %d", [config templteId], cardType, erroCode);
        return;
    }
    //创建成功
    self.card = card; //需要持有卡片，才可以进行操作

    NSLog(@"load card success %@ style %d error %d", [config templteId], cardType, erroCode);
    dispatch_async(dispatch_get_main_queue(), ^{
        CubeView *view = [[[CubeService sharedInstance] getEngine] createView];
        CGSize size = [card getSize];
        [view setFrame:CGRectMake(0, 320, size.width, size.height) ];
        [self.view addSubview:view];
        [self.card renderView:view];

    });
}
```

6. 效果预览。

2.3.2. 进阶指南

2.3.2.1. 渲染卡片

本文介绍了在 iOS 客户端渲染卡片的整体实现流程。

渲染卡片的流程分为四部分。第一步，组装卡片配置信息；第二步，根据配置信息请求卡片，获取卡片实例；第三步，通过卡片实例，使用卡片 `View` 去渲染；第四步，在整个业务完成后，在 `destroy` 声明周期中，释放卡片。具体流程如下：

1. 组装卡片配置信息。

   创建配置信息，并设置各种参数。

```
- (void)createCubeConfig {
                // 设置卡片参数
    CubeCardConfig *cardConfig = [[CubeCardConfig alloc] init];
    // 卡片版本
    [cardConfig setVersion:@"1.0.0.0"];
    // 后台创建的卡片 ID
    [cardConfig setTempleteId:@"987654321"];
    // 预设宽度
    [cardConfig setWidth:MP_Screen_Width - 40];
    // 预设高度
    [cardConfig setHeight:CGFLOAT_MAX];
}
```

2. 请求卡片。

   根据组装好的卡片配置信息请求卡片，卡片引擎会去服务端获取卡片模板信息。可以使用 `createCard` 方法一次请求一个卡片，也可以使用 `createCards` 方法一次请求多个卡片。

```
- (void)requestCard {
    // 加载单个卡片
    [[[CubeService sharedInstance] getEngine] createCard:cardConfig callback:self];

    NSMutableArray *cardArray = [NSMutableArray array];
    [cardArray addObject:cardConfig];
    [cardArray addObject:cardConfig];

    // 加载多个卡片
    [[[CubeService sharedInstance] getEngine] createCards:cardArray callback:self];
}
```

3. 渲染卡片。

   拿到卡片信息之后，生成卡片 View，然后在主线程中进行渲染。在此环节中需要进行异常判断，防止未顺利获取到卡片信息的情况发生。

```
#pragma mark---CrystalCardCallback
- (void)onLoaded:(CubeCard *)card cardType:(CCardType)cardType config:(CubeCardConfig *)config e
rroCode:(CubeCardResultCode)erroCode {
    if (!card) {
        NSString *errMsg = [NSString stringWithFormat:@"创建失败: templteId=%@,style=%d, error=%d"
, [config templteId], cardType, erroCode];
        NSLog(@"错误信息:%@", errMsg);
        return;
    }

    NSLog(@"创建成功 succ %@ style %d error %d", [config templteId], cardType, erroCode);

    dispatch_async(dispatch_get_main_queue(), ^{
        //主线程刷新UI操作
        CubeView *view = [[[CubeService sharedInstance] getEngine] createView];
        CGSize size = [card getSize];

        if (![view isEqual:[NSNull null]]) {
            [view setFrame:CGRectMake(0, 0, MP_Screen_Width - 40, size.height)];
            [card renderView:view];
        }

        [self.view addSubView:view];
    });
}
```

4. 释放卡片。

   卡片使用完成之后，需要释放卡片的内存资源，通常是在页面的 `dealloc` 生命周期里调用或者主动销毁。

```
- (void)destoryCubeService {
    //销毁卡片引擎
    if (![[[CubeService sharedInstance] getEngine] isEqual:[NSNull null]]) {
        [[CubeService sharedInstance] destroyEngine];
    }

    //删除卡片视图
    [self.view removeAllSubviews];
}
```

2.3.2.2. 真机预览

本文介绍了在 iOS 客户端中进行真机预览卡片的操作流程。

前置条件

- 已经开通并接入 mPaaS。
- 已经安装蚂蚁动态卡片 AntCubeTool 工具。更多详情请参见 关于 AntCubeTool。
- 已经按照 快速开始 完成接入流程。

操作步骤

1. **添加真机预览依赖。以 Pod 为例。**

   i. 添加 Pod 模块。

   ```
   mPaaS_pod "mPaaS_Cube"
   mPaaS_pod "mPaaS_Cube_Dev"
   ```

   > **重要**
   > - mPaaS_Cube 和 mPaaS_Cube_Dev 必须配对使用。
   > - 发布之前，务必删掉 mPaaS_Cube_Dev。

   ii. 执行 pod 更新。

   ```
   pod mpaas update cp_change_29273
   ```

   iii. 执行 `pod install` 。

   ```
   pod mpaas update cp_change_29273
   ```


   iv. 添加扫码预览需要的依赖库。

   ```
   #import <APCubePlayground/APCubePlayground.h>
   #import <AudioToolbox/AudioToolbox.h>
   #import <MPCubeAdapter/MPCubeAdapter.h>
   ```

v. 设置预览页面。

> **重要**
> - 使用预览功能时需要确保 engine 已经初始化成功。
> - 如果使用了自定义扩展 module，需确保自定义 module 已注册完成，否则无效。

- 如果使用默认预览页面可以直接调用以下代码：

```
[MPCubeAdapter setDefaultDebugPreview:YES];
```

- 如果自己定义预览页面则需要主动注册 `listener` 且实现 `CubeCardPreviewListener Protocol`：

```
[APCubePlayground registPreViewListener:id<CubeCardPreviewListener>];

//  CubeCardPreviewListener
- (void)onPreViewFinish:(BOOL)rst cubeCard:(CubeCard*)cubeCard {

}
```

vi. 调用扫码预览方法。

```
- (void)scan {
    [APCubePlayground launch];
    [APCubePlayground scan];
}
```

> **重要**
> 当前 App 需要有 `NavigationController`，否则调用 `scan` 后会得不到响应。

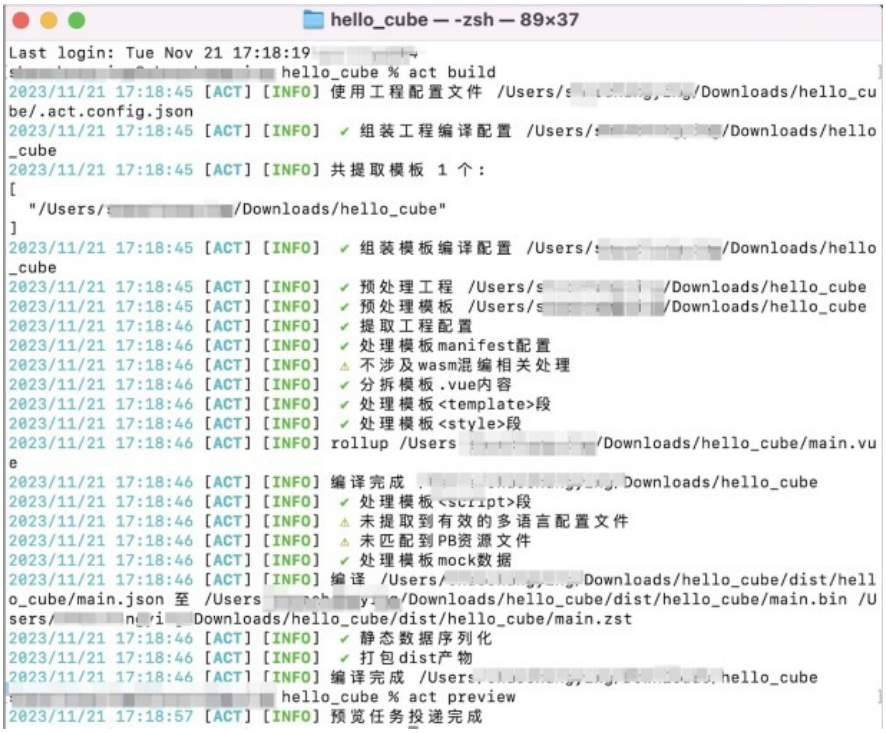
2. 使用命令行启动本地调试服务。在工程的路径下，运行指令开启服务。在 macOS 和 Windows 上开启服务的指令如下：
   - macOS: `act prepare && act server`
   - Windows: `act prepare | act server`

   执行指令后，在终端上会生成二维码。

3. 启动客户端并扫码，建立连接。终端会提示设备已连接。

4. 卡片预览。

   终端新增视窗，打开另一个窗口，使用 cd 命令切换到工程目录下执行 `act build` 完成编译。再调用 `act preview` 将编译好的内容推送到客户端上。



- 如果设置使用默认预览页面，客户端即可自动跳转展示预览的卡片。
- 如果自己定义了预览界面，则会触发 `CubeCardPreviewListener Protocol` 回调，此时业务侧拿到 `CubeCard` 实例自行渲染即可。

2.3.2.3. 初始化引擎扩展

本文介绍了在 iOS 客户端中初始化引擎扩展的操作步骤。

前置条件

- 您已经接入工程到 mPaaS。更多信息，请参见 接入方式介绍 。
- 已经在控制台 发布蚂蚁动态卡片 。

操作步骤

1. **设置预置卡片本地路径。**
   - **设置卡片引擎类 `CubeEngineConfig` 的属性。**

     ```
     // 卡片引擎类 CubeEngineConfig 的属性

     /// 存储模版的本地资源包的路径
     @property (nonatomic, strong) NSString *bundlePath;
     ```

   - 设置本地蚂蚁动态卡片的 assets 路径。

     ```
     - (void)initEngine {
         //本地模板所在的 Bundle 路径
         NSString *bundlePath = [NSString stringWithFormat:@"%@/%@", [[NSBundle mainBundle] resourc
     ePath], @"MPCubeBundle.bundle"];
         CubeEngineConfig *config = [[CubeEngineConfig alloc] init];
         //设置读取资源路径
         [config setBundlePath:bundlePath];
         //配置卡片引擎
         [[CubeService sharedInstance] initWithConfig:config];
     }
     ```

2. 设置异常监听。

   `CubeEngineConfig` 支持异常监听，捕获前端代码异常。客户端需要遵循 `CExceptionListener` 代理，实现监听方法。

   ```
   // 卡片引擎类 CubeEngineConfig 的属性

   /// 异常监听
   @property (nonatomic, strong) id<CExceptionListener> exceptionListener;
   ```

```
// CExceptionListener 代理类

//  CrystalExceptionProtocol.h
//  CubeCrystal
//  Created by hejin on 2021/9/10.

#import <Foundation/Foundation.h>
#import "CExceptionInfo.h"
#ifndef CExceptionListener_h
#define CExceptionListener_h

@protocol CExceptionListener <NSObject>

@required

/**
 异常监听
 @param info 异常信息
 */
- (void)onException:(CExceptionInfo *)info;

@end
```

```
@interface MPCubeManager () <CExceptionListener>

- (void)initEngine {
    CubeEngineConfig *engineConfig = [[CubeEngineConfig alloc] init];
    //设置 delegate
    engineConfig.exceptionListener = self;
    [[CubeService sharedInstance] initWithConfig:engineConfig];
}

//实现监听方法，打印监听信息
- (void)onException:(CExceptionInfo *)info {
    NSLog(@"异常类型: %lu", (unsigned long)info.type);
    NSLog(@"异常信息: %@", info.message);
    NSLog(@"异常卡片 id: %@", info.cardUid);
    NSLog(@"异常信息扩展参数: %@", info.extraInfo);
}
```

3. 设置网络图片下载 Handler。

`CubeEngineConfig` 支持拦截 Image 下载，定制图片参数；客户端需要遵循 `CKImageHandler` 代理，实现监听方法。

```
// 卡片引擎类 CubeEngineConfig 的属性

/// 图片 handler，如果为空，则内部默认实现，建议自定义 handler
@property (nonatomic, strong) id<CKImageHandler> imageHandler;
```

```
//  CKImageHandler.h
//  CubePlatform
//  Created by Joe on 2018/8/15.
//  Copyright © 2018年 mQuick. All rights reserved.

#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>

#ifndef CKImageHandler_h
#define CKImageHandler_h

extern NSString *const CKImageAppInstanceIDKey; // 触发加载图片对应的 AppInstanceID
extern NSString *const CKImagePageInstanceIDKey; // 触发加载图片对应的 PageInstanceID
extern NSString *const CKImageInstanceOptionKey; // instance option. 针对 falcon 链路添加

typedef void(^CKImageHandlerCallback)(UIImage *image, NSError *error);

@protocol CKImageHandler <NSObject>

@required
/**
 @param url 图片下载 URL
 @param size 图片尺寸
 @param option 扩展参数，视图片下载库需求而定
 @param callback 下载回调，下载完成后通过此接口回调
 @return 返回表示本次下载任务的唯一 ID
 */
- (NSString *)fetchImage:(NSString *)url size:(CGSize)size option:(NSDictionary *)option callbac
k:(CKImageHandlerCallback)callback;

@optional
/**
 @param fetchID 任务 ID, fetchImage 的返回值
 */
- (void)cancel:(NSString*)fetchID;

@end
```

```
@interface MPCubeManager () <CKImageHandler>

- (void)initEngine {
    CubeEngineConfig *engineConfig = [[CubeEngineConfig alloc] init];
    //设置 delegate
    engineConfig.imageHandler = self;
    [[CubeService sharedInstance] initWithConfig:engineConfig];
}

- (NSString *)fetchImage:(NSString *)url size:(CGSize)size option:(NSDictionary *)option callbac
k:(CKImageHandlerCallback)callback {
    NSLog(@"图片地址: %@", url);
    NSLog(@"图片扩展参数: %@", option);

    return @"20211202";
}
```

2.3.2.4. 预置卡片

本文介绍了在 iOS 客户端中预置蚂蚁动态卡片的操作步骤。

前置条件

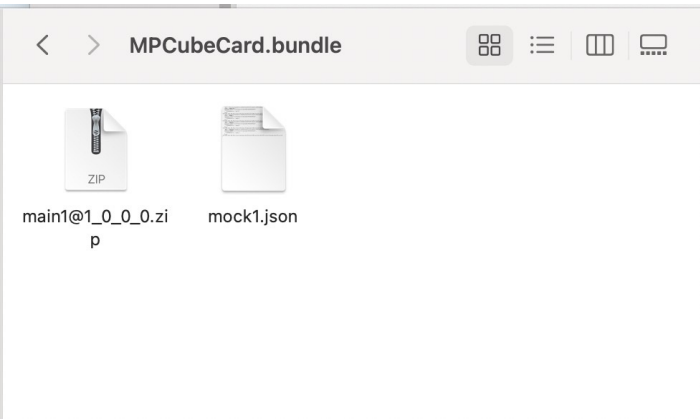
- 您已经接入工程到 mPaaS。更多信息，请参见 接入方式介绍。
- 已经在控制台 发布卡片。

操作步骤

1. **从控制台下载 Bin 文件。**

2. **将下载的文件添加到 Bundle 中。**

   新建 Bundle 文件，将下载的 Bin 文件压缩成 Zip 包，命名为 `卡片 ID@卡片版本`，例如 `main1@1_0_0_0.zip`。此处的 JSON 文件为卡片所需要的数据，可以使用文件内置，也可以通过调用接口动态请求，根据自身需要选择数据来源方式。



> **说明**
>
> **后期控制台将直接提供完整命名的 zip 包，无需用户手动修改命名，只需要添加到 Bundle 中即可。**

3. **将 Bundle 引入工程中，引擎初始化时传入 Bundle 路径。**

```
- (void)initEngine {
    //本地资源模板的 Bundle 路径
    NSString *bundlePath = [NSString stringWithFormat:@"%@/%@", [[NSBundle mainBundle] resourceP
ath], @"MPCubeBundle.bundle"];
    CubeEngineConfig *config = [[CubeEngineConfig alloc] init];
    [config setBundlePath:bundlePath];
    [[CubeService sharedInstance] initWithConfig:config];
}
```

4. **加载预置卡片。**

```
CubeCardConfig *cardConfig = [[CubeCardConfig alloc] init];
//设置卡片版本
[cardConfig setVersion:@"1.0.0.0"];
//设置卡片 ID
[cardConfig setTempleteId:@"main1"];
//预设卡片宽度
[cardConfig setWidth:MP_Screen_Width];
//预设卡片高度
[cardConfig setHeight:CGFLOAT_MAX];

NSString *mockBundlePath = [NSString stringWithFormat:@"%@/%@", [[NSBundle mainBundle] resourceP
ath], @"MPCubeBundle.bundle"];
NSString *fullTplPath = [NSString stringWithFormat:@"%@/%@", mockBundlePath, @"mock1.json"];
NSString *strData = [NSString stringWithContentsOfFile:fullTplPath encoding:NSUTF8StringEncoding
error:nil];
//设置卡片数据，此处为 Bundle 内置 Json 读取数据；也可以请求自身服务端接口获取数据，但要转换为 Json 字符串
[cardConfig setData:[self dictionaryWithJsonString:strData]];
//创建卡片
[[CubeEngine sharedInstance] createCard:cardConfig callback:self];
```

2.3.2.5. 卡片调用客户端方法

本文介绍了在蚂蚁动态卡片中调用 iOS 客户端方法的实现路径。

操作步骤

1. **在工程中新建 `NSObject` 对象并引入协议 `<CubeModuleProtocol>` 。**

```
#import <Foundation/Foundation.h>
#import <CubeCrystal/CubeModuleProtocol.h>

NS_ASSUME_NONNULL_BEGIN

@interface MPCubeModule : NSObject <CubeModuleProtocol>

@end

NS_ASSUME_NONNULL_END
```

2. **实现与模板约定的方法。**

   与模板约定的方法名为 `push` ，效果是点击卡片时调用 `Native` 实现 `Push Viewcontroller` 操作。

```
#import "MPCubeModule.h"
#import <CubeBridge/CKJSDefine.h>

@implementation MPCubeModule

//cardUid 是 module 的属性，在调用 module api 时可以通过 cardUid 判断是哪个卡片产生的调用，cardUid 即 Cub
eCard 中的 cardUid。
@synthesize cardUid;

//配置约定的方法
CK_EXPORT_METHOD(@selector(push:))

- (void)push:(CubeModuleMethodCallback)callback {
    if (callback) {
                //此处执行Push ViewController操作

        //回调给模板的参数
        callback(@"Push Page");
    }
}

@end
```

3. **注册自定义 Module。**

```
- (void)registerModules {
    [MPCubeManager shareManager];

    //注册自定义Module，key为module名称（与服务端约定），value为客户端类名
    NSDictionary *dic = @{@"crystalnavigator": @"MPCubeModule"};
    [[[CubeService sharedInstance] getEngine] registerModules:dic];
}
```

4. **卡片侧调用。**

```
<script>
                //约定的自定义Module标识
    const navigator = requireModule("crystalnavigator");
    export default {
        methods: {
            onClick() {
                navigator.push(event => {console.info(event);});
                console.info('invoke on-click event');
            }
        }
    }
</script>
```

5. 将卡片打包发布到后台，即可调用客户端方法。

2.3.2.6. 客户端调用卡片方法

本文介绍了在 iOS 客户端调用卡片方法的实现路径。

操作步骤

1. **在卡片侧实现调用的 JS 方法。**

```
<script>
    const navigator = requireModule("crystalnavigator");
    export default {
        data: {
            message: 'Hello Cube',
            text:"Cube引擎是一套简单易用的跨平台开发方案，能以Web的开发体验构建高性能",
            string: "defaultString",
            temp:"https://gw.alicdn.com/tfs/TB1dZ4WowoQMeJjy0FnXXb8gFXa-950-1267.jpg"
        },
        methods: {
                        //客户端调用的JS方法
            jsTestMethod(string) {
                    //更新页面参数
              this.string = string;
           }
        }
    }
</script>
```

2. 将卡片打包，发布到卡片后台。
3. 客户端调用。

   在卡片加载的代理方法中获取卡片实例，调用对应的 JS 方法，并发送数据。

```
#pragma mark---CrystalCardCallback
- (void)onLoaded:(CubeCard *)card cardType:(CCardType)cardType config:(CubeCardConfig *)config e
rroCode:(CubeCardResultCode)erroCode {
    if (!card) {
        NSString *errMsg = [NSString stringWithFormat:@"创建失败: templteId=%@,style=%d, error=%d"
, [config templteId], cardType, erroCode];
        NSLog(@"错误信息:%@", errMsg);
        return;
    }

    NSLog(@"创建成功 succ %@ style %d error %d", [config templteId], cardType, erroCode);

    dispatch_async(dispatch_get_main_queue(), ^{
        NSString *text = @"客户端调用卡片JS方法传参: Hello World";
        if (![text isEqualToString:@""]) {
            NSArray *valueArray = @[text];
            //调用与卡片约定的JS方法，传入参数
            [card callJsFunction:@"jsTestMethod" arguments:valueArray];
        }
    });
}
```

2.3.2.7. 卡片自定义标签

本文介绍了在卡片中自定义标签的实现步骤。

1. **客户端注册自定义标签（自定义 View）。**
   i. **实现自定义标签（自定义 View）。**

   继承 `CCardWidget` 并实现其协议方法，其中 `onCreateView` 和 `updateData` 为必须实现的方法，其他方法为可选。

```
#import "CustomCardView.h"

@interface CustomCardView ()
```

```
@property (nonatomic, strong) UILabel *titleLabel;

@end

@implementation CustomCardView

//必须实现
/**
 * UI 创建接口，表示当前组件要上屏的视图 UI
 *
 * @param data  map 类型，组件在卡片上声明的数据，包括样式、事件和属性，对应的 key 为DATA_KEY_STYLES、DATA_KEY_EVENTS、DATA_KEY_ATTRS;
 * @param size  View 的大小，宽高
 * @return 可用的 UIView
 */
- (UIView *)onCreateView:(NSDictionary *)data size:(CGSize)size {
    NSLog(@"数据打印:%@", data);

    UIView *customView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, size.width, size.height)];
    customView.backgroundColor = [UIColor redColor];

    self.titleLabel = [[UILabel alloc] initWithFrame:CGRectMake(0, 0, size.width, 30)];
    //数据来源是 data 提供
    self.titleLabel.text = @"A";
    [customView addSubview:self.titleLabel];

    return customView;
}

//必须实现
/**
 * 组件数据更新接口
 * @param data
 */
- (void)onUpdateData:(NSDictionary *)data {
    NSLog(@"数据打印:%@", data);
}

/**
 *  从复用池中获取的已存在控件，被使用前的数据准备。
 *  @param data 初始化数据
 *  @param size 组件复用时的大小
 */
- (void)onReuse:(NSDictionary *)data size:(CGSize)size {
    NSLog(@"数据打印:%@", data);
    NSLog(@"Size打印:%@", size);

    //数据来源是 data 提供
    self.titleLabel.text = @"B";

}

/**
 * 组件是否支持复用;
 * 为了提高效率，扩展组件可以支持复用。例如当某个自定义的标签组件由于数据更新被移除当前视图，此时该组件如果支持复用，那么会放入复用池内，下次该组件显示时，会直接从复用池内获取;
 * @return YES：复用；NO：不复用。
 */
```

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

/**
 * 组件复用清理接口。如果组件支持复用（canReuse 返回 true），则组件在进入复用池后会调用
 * onRecycleAndCached 方法，表示当前组件已经离屏放入缓存内，需要清理资源。
 */
- (void)onRecycleAndCache {
    //清理自定义 View 上的 SubView 内容
    self.titleLabel.text = @"";
}
```

ii. **注册自定义标签。**

第一个参数是自定义的标签，需要与卡片侧约定好，在卡片侧会写到 <> 内。建议加上前缀，防止和动态卡片本身的标签冲突。第二个是自定义标签实现类的类名。

```
CubeWidgetInfo *widgetInfo = [CubeWidgetInfo new];
widgetInfo.tag = @"custom-widget";
widgetInfo.className = [CustomCardView class];

NSMutableArray *widgetArray = [NSMutableArray array];
[widgetArray addObject:widgetInfo];
[[[CubeService sharedInstance] getEngine] registerWidgets:[NSArray
arrayWithArray:widgetArray]];
```

2. **卡片侧调用。**

在标签中，写入自定义的标签。此处为 `custom-widget`，对应客户端注册的标签。

> **说明**
> 写入的自定义标签需要和注册自定义标签步骤中的第一个参数保持一致。

```
<template>
    <div class="root">
        ...
        <custom-widget></custom-widget>
                                ...
    </div>
</template>
```

3. 将卡片打包发布到后台或者预览，即可渲染客户端自定义的 View。

2.3.3. 接口说明

2.3.3.1. CubeService

本文介绍了蚂蚁动态卡片服务核心类的方法。

方法

sharedInstance

```
/**
 获取 service 实例

 @param 无
 @return CubeCrystalService
*/

+ (instancetype)sharedInstance;
```

initWithConfig

```
/**
 初始化引擎

 @param config 配置参数
 @return 无
*/

- (void)initWithConfig:(CubeEngineConfig *)config;
```

getEngine

```
/**
 获取引擎实例

 @param 无
 @return CubeEngine 单例引擎实例
*/

- (CubeEngine *)getEngine;
```

destroyEngine

```
/**
 释放引擎资源，当保证不再调用 crystal 相关接口时可调用释放资源
*/

- (void)destroyEngine;
```

2.3.3.2. CubeEngine

本文介绍了蚂蚁动态卡片引擎核心类的方法。

createCard

```
/**
 创建单个卡片

 @param config 卡片配置参数
 @param callback 卡片创建回调
*/

- (void)createCard:(CubeCardConfig *)config callback:(id<CCardCallback>)callback;
```

createCards

```
/**
 创建批量卡片，每个卡片结果回调一次

 @param configs 批量卡片配置参数
 @param callback 卡片创建回调
*/

- (void)createCards:(NSArray<CubeCardConfig *> *)configs callback:(id<CCardCallback>)callback;
```

createView

```
/**
 创建渲染 view

 @return CubeView
*/

- (CubeView *)createView;
```

setCustomUnit

```
/**
 设置自定义单位

 @param unitName 单位名称，例如 sip
 @param unitRadio 单位比例，例如 1.5
*/

- (void)setCustomUnit:(NSString *)unitName unitRadio:(float)unitRadio;
```

registerModules

```
/**
 设置自定义扩展 module

 @param modules 字典，key 为 module 名称，例如 animation，value 为类名，例如 CKAnitmationModule
*/

- (void)registerModules:(NSDictionary<NSString *, NSString *> *)modules;
```

registerWidgets

```
/**
 注册组件
 @param widgets 组件配置信息
 */
- (void)registerWidgets:(NSArray<CubeWidgetInfo *> *)widgets;
```

sendEvent

```
/**
 Native 向 JS 侧发送自定义事件通道
 @param widgetData 组件数据，即 CCardWidget 中 onCreateView 创建组件时的入参 data，在这里透传即可
 @param eventName 自定义事件名称
 @param eventParams 自定义事件参数
 */
- (void)sendEvent:(NSDictionary *)widgetData eventName:(NSString *)eventName eventParams:(NSDictionary*)eventParams;
```

2.3.3.3. CubeEngineConfig

本文介绍了蚂蚁动态卡片引擎初始化配置类的方法。

属性

bundlePath

```
/// 存储模板的本地资源包的路径
@property (nonatomic, strong) NSString *bundlePath;
```

exceptionListener

```
/// 异常监听
@property (nonatomic, strong) id<CExceptionListener> exceptionListener;
```

imageHandler

```
/// 图片 handler。如果为空，则内部默认实现，建议自定义 handler。
@property (nonatomic, strong) id<CKImageHandler> imageHandler;
```

2.3.3.4. CExceptionListener

本文介绍了蚂蚁动态卡片异常监听器的方法。

方法

onException

```
/**
 异常监听

 @param info 异常信息
*/

- (void)onException:(CExceptionInfo *)info;
```

2.3.3.5. CExceptionType

本文介绍了蚂蚁动态卡片的异常类型。

枚举

```
/// 异常类型
@property (nonatomic, assign) CExceptionType type;

typedef NS_ENUM(NSUInteger, CExceptionType) {
                //JS 异常
    CExceptionTypeJavaScript = 0,
    //卡片样式异常
    CExceptionTypeStyle
};
```

2.3.3.6. CExceptionInfo

本文介绍了蚂蚁动态卡片异常信息类的属性。

属性

exceptionMessage

```
/// 异常信息
@property (nonatomic, copy) NSString *message;
```

exceptionCardUid

```
/// 异常卡片id
@property (nonatomic, copy) NSString *cardUid;
```

exceptionExtraInfo

```
/// 异常信息扩展参数
@property (nonatomic, copy) NSDictionary *extraInfo;
```

2.3.3.7. CubeCard

本文介绍了蚂蚁动态卡片核心类的属性和方法。

属性

cardUid

```
/// 卡片唯一 id
@property (nonatomic, readonly) NSString *cardUid;
```

方法

renderView

```
/**
 渲染 view
 @param view
*/
- (void)renderView:(CubeView *)view;
```

getSize

```
/**
 获取卡片大小
 @return 卡片 size
*/
- (CGSize)getSize;
```

updateData

```
/**
 更新数据渲染
 @param data 模板数据，json 格式
*/
- (void)updateData:(NSDictionary *)data;
```

callJsFunction

```
/**
 调用 JS 方法
 @param function 方法名
 @param arguments 调用参数
*/
- (void)callJsFunction:(NSString *)function arguments:(NSArray *)arguments;
```

notifyState

```
/**
 通知卡片状态
 @param state 卡片状态，在卡片出屏，上屏、前后台时通知变更状态
 */
-(void)notifyState:(CCardState)state;
```

getBindView

```
/**
 获取卡片绑定的 view（view 可能被其他卡片复用）
 @return 绑定 view
 */
-(CubeView*)getBindView;
```

2.3.3.8. CubeCardConfig

本文介绍了蚂蚁动态卡片配置类的属性。

属性

templateId

```
/// 模版唯一 Id，必填
@property (nonatomic, strong) NSString *templateId;
```

version

```
/// 模版版本号，必填
@property (nonatomic, strong) NSString *version;
```

data

```
/// 卡片数据
@property (nonatomic, strong) NSDictionary *data;
```

width

```
/// 卡片预设宽度
@property (nonatomic, assign) NSInteger width;
```

height

```
/// 卡片预设高度
@property (nonatomic, assign) NSInteger height;
```

extOption

```
/// 卡片扩展数据
@property (nonatomic, strong) NSDictionary *extOption;
```

envData

```
/// 卡片环境变量
@property (nonatomic, strong) NSDictionary *envData;
```

layoutChangeListener

```
/// 卡片布局大小变更监听
@property (nonatomic, weak) id<CCardLayoutChangeListener> listener;
```

2.3.3.9. CCardType

本文介绍了蚂蚁动态卡片的模板来源类型。

- **声明：** `typedef enum{C_CARD_TYPE_NONE, C_CARD_TYPE_CACHE, C_CARD_TYPE_BUNDLE, C_CARD_TYPE_FILE, C_CARD_TYPE_CLOUD} CCardType;`
- **说明：卡片结果码。**
- **枚举值**

枚举值	说明
C_CARD_TYPE_NONE	无效模板
C_CARD_TYPE_CACHE	内存中模板
C_CARD_TYPE_BUNDLE	本地资源包中模板
C_CARD_TYPE_FILE	本地物理存储模板
C_CARD_TYPE_CLOUD	云端模板

2.3.3.10. CCardLayoutChangeListener

本文介绍了蚂蚁动态卡片的布局变更通知监听类的方法。

方法

onLayout

```
/**
 卡片大小变更

 @param size 卡片新 size
*/

- (void)onLayout:(CGSize)size;
```

2.3.3.11. CCardCallback

本文介绍了蚂蚁动态卡片创建的回调类的方法。

方法

onLoaded

```
/**
 卡片回调接口

 @param card 卡片实例，当创建失败时为nil
 @param cardType 卡片模版来源
 @param config 创建卡片的配置参数
 @param erroCode 错误码
*/

- (void)onLoaded:(CubeCard *)card cardType:(CCardType)cardType config:(CubeCardConfig *)config err
oCode:(CubeCardResultCode)erroCode;
```

2.3.3.12. CubeModuleProtocol

本文介绍了蚂蚁动态卡片实现协议 `CubeModuleProtocol` 的属性。

属性

cardUid

```
/// 调用卡片的唯一 ID
@property (nonatomic, copy) NSString *cardUid;
```

CubeModuleMethodCallback

```
/// callback
typedef void (^CubeModuleMethodCallback) (id result);
```

2.3.3.13. CubeCardResultCode

本文介绍了蚂蚁动态卡片的结果码。

- **声明：** `typedef enum{CubeCardResultSucc = 0, CubeCardResultNetworkError, CubeCardResultParamError,`

```
CubeCardResultNotExist, CubeCardResultContentInvalid}CubeCardResultCode;
```

- **说明：卡片结果码。**
- **枚举值**

枚举值	说明
CubeCardResultSucc = 0	卡片请求成功
CubeCardResultNetworkError	网络异常
CubeCardResultParamError	请求参数异常
CubeCardResultNotExist	卡片不存在
CubeCardResultContentInvalid	卡片内容无效

3.使用控制台

3.1. 卡片管理

3.1.1. 创建卡片

本文介绍了在 mPaaS 控制台创建蚂蚁动态卡片的操作流程。

操作步骤

1. 进入 mPaaS 控制台，在应用列表中选择目标应用。
2. 单击左侧导航栏中的 **实时发布** > **蚂蚁动态卡片** > **卡片管理**，进入 **卡片管理** 页。
3. 在 **卡片管理** 页，单击 **新建卡片**，在 **创建卡片** 表单中填写以下信息：
   - **卡片 ID**：必填。卡片 ID 建议使用英文、数字和下划线的组合，长度不得少于 8 位字符。
   - **卡片名称**：必填。可以取任意值。
   - **描述**：选填。

4. 填写完成后，单击 **确定**。

至此，您已完成了创建卡片，创建后的卡片会出现在卡片列表中。

后续步骤

创建卡片后，还需要上传卡片资源才可以发布卡片。更多信息，请参见 添加卡片资源。

3.1.2. 添加卡片资源

本文介绍了在 mPaaS 控制台为已创建的蚂蚁动态卡片添加卡片资源的操作流程。

操作步骤

1. 进入 mPaaS 控制台，在应用列表中选择目标应用。
2. 单击左侧导航栏中的 **实时发布** > **蚂蚁动态卡片** > **卡片管理**，进入 **卡片管理** 页。
3. 在 **卡片管理** 页的 **卡片列表** 中，单击需要添加资源的卡片。
4. 单击 **添加卡片资源** 按钮，进入 **新增卡片版本** 页面。

5. 在 **新增卡片版本** 页面，填写以下配置信息并上传文件（资源文件）。完成填写后，单击 **提交**。
   - **卡片版本号**：必填。版本号应为四位，格式参考 `0.0.0.1` 。
   - **文件**：卡片模板的 zip 文件，zip 文件应小于 10MB。卡片模板的开发请参考快速开始文档（ Android、iOS）。
   - **客户端生效范围**：是可以拉取到该卡片的客户端版本范围，包含最高版本和最低版本两个信息。客户端生效范围需按照平台进行区分。
   - **最低版本**：卡片将在输入的最低客户端版本及以上版本生效。如果要覆盖所有客户端版本，在最低版本中填入 `0.0.0.0` 即可。
   - **最高版本**：卡片将在输入的最高客户端版本及以下版本生效。如不填则最低版本以上全部生效。

> **说明**
> 如填写了最高版本，可能会导致后续客户端升级到更高版本时离线包不生效的问题。

卡片资源成功添加后，卡片信息如下图所示。

3.1.3. 删除卡片

本文介绍了在 mPaaS 控制台删除蚂蚁动态卡片的操作流程。

操作步骤

1. 进入 mPaaS 控制台，在应用列表中选择目标应用。
2. 单击左侧导航栏中的 **实时发布** > **蚂蚁动态卡片** > **卡片管理**，进入 **卡片管理** 页。
3. 在 **卡片管理** 页的 **卡片列表** 中，将鼠标指针指向目标卡片，单击 按钮。
4. 在弹出的对话框中单击 **确认删除**。

至此，您已完成删除卡片。

3.1.4. 创建发布任务

在创建了蚂蚁动态卡片并添加了卡片资源后，还需要发布蚂蚁动态卡片，客户端才能够拉取到卡片。本文介绍了在 mPaaS 控制台创建蚂蚁动态卡片发布任务的操作流程。

前提条件

如果要采用白名单的发布模式，需要在发布卡片前创建好白名单。关于如何创建白名单，请参见 白名单管理。

操作步骤

1. 进入 mPaaS 控制台，在应用列表中选择目标应用。
2. 单击左侧导航栏中的 **实时发布** > **蚂蚁动态卡片** > **卡片管理**，进入 **卡片管理** 页。
3. 在 **卡片管理** 页的 **卡片列表** 中，单击需要发布的卡片。
4. 单击 **创建发布** 按钮，进入 **新建发布** 页面。

5. 在 **新建发布** 页面，进行以下配置，完成配置后，单击 **确定**。
   - **发布类型**：必选。灰度发布、正式发布。选择灰度发布时，可以进行发布模型、白名单、高级规则等项进行配置；选择正式发布时，则不需进行这些配置。
   - **发布模型**：必选。可选择白名单或时间窗的发布模型。
     - **白名单**：根据白名单进行定向发布，即只有白名单限定的用户才能够拉取到卡片。
       - **白名单配置**：如果选择了发布模型为白名单，需要在此处选择已经创建好的白名单。

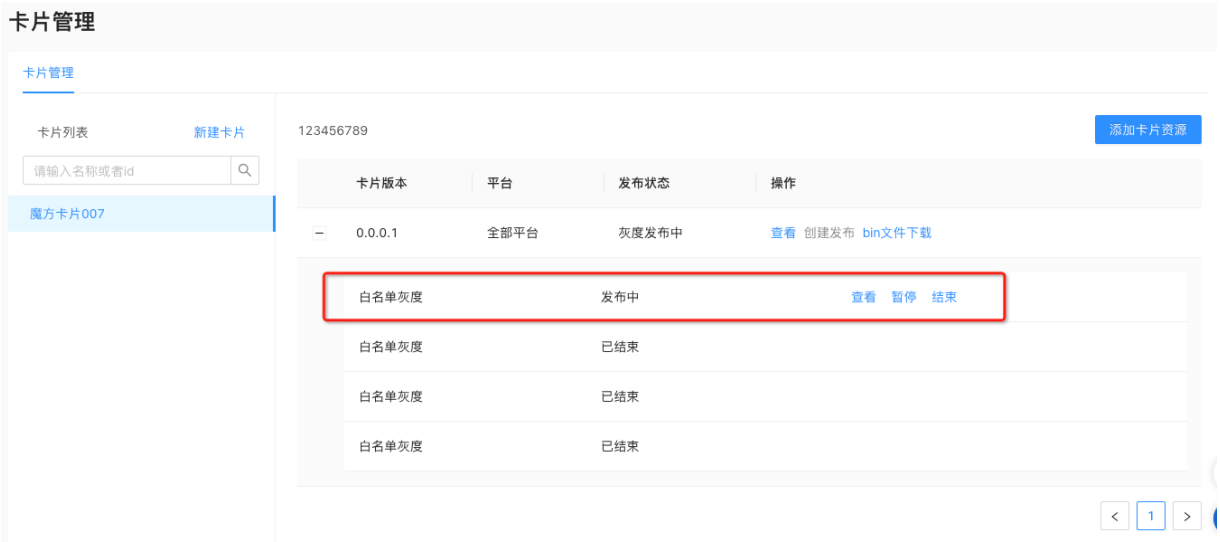
         > **说明**
         > 当选中的某个白名单用户数超过 10 万时，仅取前 10 万个用户。

     - **时间窗**：在规定时间范围内，以灰度人数为上限进行灰度发布。
       - 选择时间窗模型的灰度发布时，需要填写时间窗关闭时间和灰度人数。
   - **发布描述**：选填。对当前发布任务进行备注说明。
   - **高级规则**：可选。可以在特定规则范围内发布，支持选择平台、类型、操作类型、资源值，如下图所示。

     - **平台**：必填。iOS、Android、所有。

- **类型**：必填。城市、机型、网络、设备系统版本。
- **操作类型**：必填。包含、不包含。在设备系统版本类型下，操作类型额外有范围内、范围外选项。
- **资源值**：必填。可直接输入资源值；也可提前在发布规则中配置资源映射关系，在本处调用。更多关于添加高级规则资源值的信息，请参见 发布规则管理 。

至此，您已完成卡片的发布。卡片发布后，任务状态为 **发布中**，如下图所示。对于“发布中”的任务，可以查看、暂停或结束任务。当卡片有版本处于“发布中”任务时，不可再创建发布任务，需先等待当前发布任务结束，或结束当前的“发布中”任务后再创建新任务。



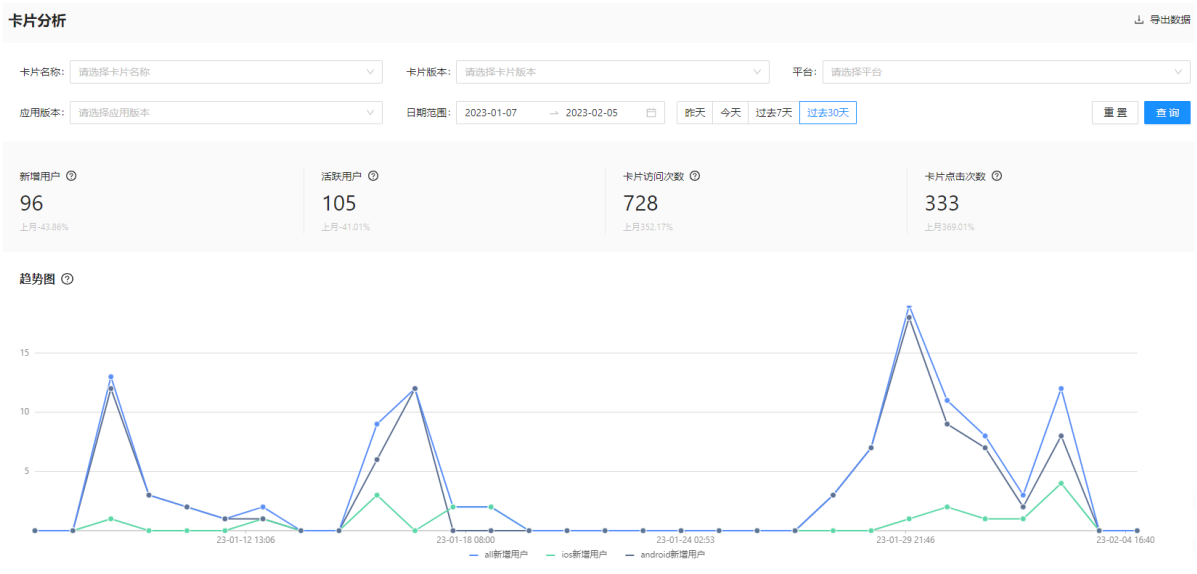
3.2. 卡片分析

本文介绍了在 mPaaS 控制台查看蚂蚁动态卡片分析的操作流程。

查看卡片分析

1. 进入 mPaaS 控制台，在应用列表中选择目标应用。
2. 单击左侧导航栏中的 **实时发布** > **蚂蚁动态卡片** > **卡片分析**，进入 **卡片分析** 页。

   该页面展示了蚂蚁动态卡片的新增用户数以及新增用户趋势图、活跃用户数以及活跃用户趋势图、卡片访问次数以及卡片访问次数趋势图、卡片点击次数以及卡片点击次数趋势图。可以根据卡片名称、卡片版本、平台、应用版本、日期范围 查询卡片分析 。



- 新增用户数：访问卡片的新增用户（设备 ID）数量。
- 活跃用户数：访问卡片的总用户（设备 ID）数，同一用户多次访问不重复计算。
- 卡片访问次数：卡片的总访问次数。
- 卡片点击次数：卡片的总点击次数。

查询卡片分析

1. 在 **卡片分析** 页面，根据需要设置卡片名称、卡片版本、平台、应用版本、日期范围或选择时间段。
2. 单击 **查询**，查看卡片分析详情。

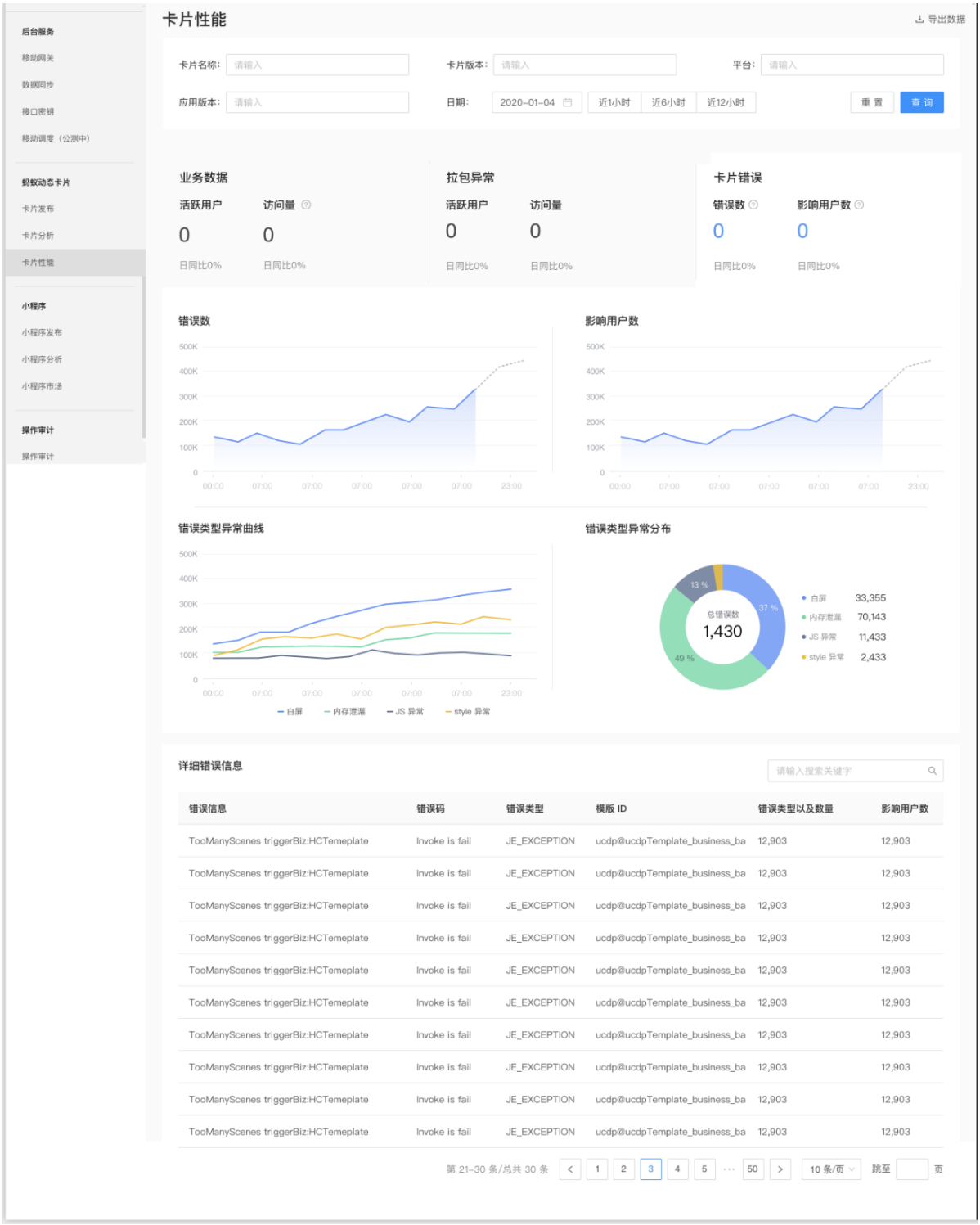
3.3. 卡片性能

本文介绍了在 mPaaS 控制台查看蚂蚁动态卡片性能的操作流程。

查看卡片性能

1. 进入 mPaaS 控制台，在应用列表中选择目标应用。
2. 单击左侧导航栏中的 **实时发布** > **蚂蚁动态卡片** > **卡片性能**，进入 **卡片性能** 页。

   该页面展示了蚂蚁动态卡片的业务数据、拉包异常和卡片错误。可以根据卡片名称、卡片版本、平台、应用版本、日期范围 查询卡片性能。



- 业务数据
  - 活跃用户：访问卡片的总用户（设备 ID）数，同一用户多次访问不重复计算。
  - 访问量：卡片的总访问次数。
- 拉包异常
  - 拉包异常次数：卡片的拉包请求异常数。
  - 拉包异常率：拉包请求异常率 = 卡片的拉包请求异常数 /卡片的拉包请求总量
- 卡片错误，展示了卡片错误数、影响用户数、错误类型异常曲线、错误类型异常分布和详细错误信息列表。

- 错误数：卡片的错误次数（JS 异常次数+ style 异常次数）。
- 影响用户数：卡片错误影响的用户（设备 ID）数。

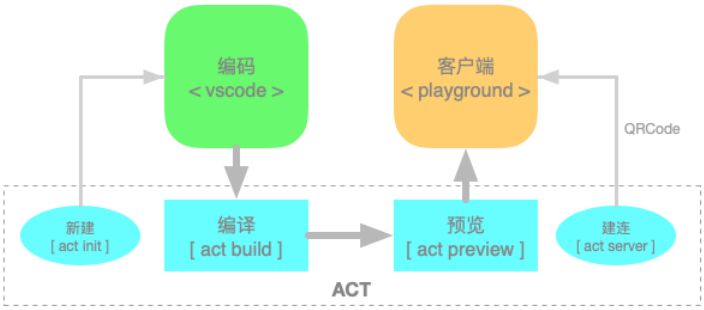
查询卡片性能

1. 在 **卡片性能** 页面，根据需要设置卡片名称、卡片版本、平台、应用版本、日期范围或选择时间段。
2. 单击 **查询**，查看卡片性能详情。

4.开发工具

4.1. 关于 AntCubeTool

AntCubeTool（简称 ACT）是用于辅助蚂蚁动态卡片进行应用场景的开发调试、效果预览的命令行工具，本文提供了对该工具的基本介绍。



ACT 适用于 macOS 和 Windows 平台。对这两个平台的具体要求如下：

- macOS
  - 系统版本不低于 11.0。
  - 命令行：可以使用 macOS 自带的终端，也可以使用其他命令行工具。
- Windows
  - 系统版本不低于 Windows 10，且需要为 64 位系统。
  - 命令行：请使用 Windows 自带的 PowerShell 命令行工具。

4.2. 安装 AntCubeTool

本文介绍了安装蚂蚁动态卡片开发工具 AntCubeTool 时需要的准备工作和安装过程。

准备

- 安装和配置 Node.js 环境。

  ACT 基于 Node.js 进行设计开发，在使用 ACT 之前请确保已正确安装和配置 Node.js 环境。Node.js 版本不低于 LTS v14，可使用命令 `node -v` 查看 NodeJS 版本。更多关于使用 Node.js 的信息请参见 Node.js 官网。

- 安装 cnpm。

  为了加快国内用户的安装速度，建议使用 npm 国内镜像源 cnpm 工具进行安装。在命令行执行以下命令即可安装。更多关于 cnpm 的使用请参见 cnpm 主页。

```
npm install -g cnpm --registry=http://registry.npmmirror.com
```

安装

使用 `cnpm install xcube -g` 即可安装 ACT。成功安装 xcube 后就可以在命令行中使用 `act` 命令。

```
cnpm install xcube -g
```

> **重要**
>
> 安装时，请勿使用 sudo 安装。

4.3. 使用 AntCubeTool

本文介绍了一些使用 AntCubeTool 的常见场景中需要用到的 `act` 命令。

新建工程

`act init`：新建工程。

> 说明
> 配置中的 name 类字段会作为创建文件或文件夹的依据，仅支持字母、数字、下划线和中折线。

```
➜  ~ act init -h
Usage: act init [options]

在当前目录下初始化一个Cube应用

Options:
  -h, --help        display help for command
```

启动服务

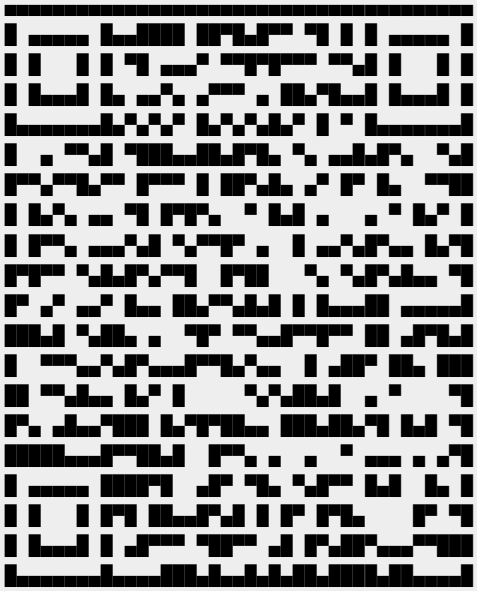
`act prepare`：清理历史端口。

`act server`：开启连接监听。

```
➜  ~ act prepare --help
Usage: act prepare [options]

准备环境

Options:
  -h, --help  display help for command

➜  ~ act server --help
Usage: act server [options]

启动通信服务器

Options:
  -h, --help  display help for command
```

在 macOS 中一般会组合使用这两条命令。在终端中执行了这两条命令后，在启动成功前不可以关闭终端窗口。启动成功后，在终端可以看到以下信息。期间如需执行其他 `act` 命令，可打开新的终端窗口执行。

```
➜  ~ act prepare && act server
2021-1-7 16:38:25 [ACT] [INFO] cmd : kill -9 $(lsof -t -i:9001)
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
2021-1-7 16:38:25 [ACT] [INFO] 端口清理完毕
2021-1-7 16:38:26 [ACT] [INFO] 启动服务 [30.30.200.193:9001]
2021-1-7 16:38:26 [ACT] [INFO] 通信通道已就绪，请扫描二维码或输入端口号进行连接
2021-1-7 16:38:26 [ACT] [INFO]
```



```
2021-1-7 16:38:26 [ACT] [INFO] 路由模块启动完毕
2021-1-7 16:38:26 [ACT] [INFO] 服务启动完毕，请勿关闭终端窗口
```

```
➜  ~ act prepare && act server
2021-1-7 16:38:25 [ACT] [INFO] cmd : kill -9 $(lsof -t -i:9001)
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -
2021-1-7 16:38:25 [ACT] [INFO] 端口清理完毕
2021-1-7 16:38:26 [ACT] [INFO] 启动服务 [30.30.200.193:9001]
2021-1-7 16:38:26 [ACT] [INFO] 通信通道已就绪，请扫描二维码或输入端口号进行连接
2021-1-7 16:38:26 [ACT] [INFO]
```

```
2021-1-7 16:38:26 [ACT] [INFO] 路由模块启动完毕
2021-1-7 16:38:26 [ACT] [INFO] 服务启动完毕，请勿关闭终端窗口
```

建立连接

通过 蚂蚁动态卡片-开发工具 可以建立连接。更多详情，请参见：

- Android 真机预览
- iOS 真机预览

编译工程

`act build`：编译工程。

```
➜  ~ act build --help
Usage: act build [options] [path]

编译指定路径下的卡片应用，未指定路径时使用当前路径

Options:
  -p, --path [v]   待编译路径，兼容老版本工具【已废弃：请直接使用 build [path] 】
  -a, --all <v>    批量编译，兼容老版本工具【已废弃：请使用 build [path] --batch 】
  --batch          是否为批量处理模式 (default: false)
  --watch          是否打开实时编译，不支持批量处理模式 (default: false)
  -h, --help       display help for command
```

配置了 `--watch` 参数时，即打开了实时编译时，ACT 将监听整个工程目录内的文件变更，并在文件保存时自动触发编译。

> **重要**
> 开启实时编译时请勿关闭终端窗口。

预览工程

通过 蚂蚁动态卡片-开发工具 可以进行工程预览。更多详情，请参见：

- Android 真机预览
- iOS 真机预览

实时预览

通过 蚂蚁动态卡片-开发工具 可以进行实时预览。更多详情，请参见：

- Android 真机预览
- iOS 真机预览

查看当前连接二维码

`act qrcode`：查看当前连接二维码。该命令用于需要重新连接 Playground 设备的场景。act server 长时间运行后，日志过多，无法看到 act server 之前输出的二维码内容，此时就可以通过 `qrcode` 命令再次查看二维码。

```
➜  ~ act qrcode --help
Usage: act qrcode [options]

显示可供连接的二维码

Options:
  -o, --onlyInfo  仅输出二维码内容，默认关闭，启用时不会输出二维码模拟图像内容 (default: false)
  -p, --pure      仅输出内容，不包含格式化信息（时间、前缀等） (default: false)
  -h, --help      display help for command
```

通过配置参数，可以仅输出二维码内容字符串。其他基于 ACT 的工具可以考虑通过这种方式获取到二维码内容。

```
act qrcode -o -p
```

查看当前连接设备

`act device`：查看当前连接设备。该命令主要用于查看当前的 server 上是否有已连接的 Playground 设备，设备未连接时 preview、alive 类预览功能无法工作。

```
➜  ~ act device --help
Usage: act device [options]

显示当前连接的设备列表

Options:
  -h, --help  display help for command
```

查看帮助

- 查看支持的指令集

```
act -h
```

- 查看指令相关参数

```
act [command] -h
```

查看信息

```
act info
```

4.4. 更新 AntCubeTool

本文介绍了更新 AntCubeTool 需要使用到的命令。

使用 `act update` 命令即可更新 ACT。

```
➜  ~ act update --help
Usage: act update [options]

检查并更新到最新版本。

Options:
  -h, --help  display help for command
```

您也可以通过 `cnpm install xcube -g` 命令重新安装以获取最新版本。

4.5. 卸载 AntCubeTool

使用 `cnpm uninstall xcube -g` 命令即可卸载 ACT。

```
➜  ~ cnpm uninstall xcube -g
```

5.卡片语法

5.1. 卡片基础

5.1.1. 工程管理

本文介绍了蚂蚁动态卡片的工程配置信息。

工程目录结构

一个有效的卡片工程由位于工程根目录下的一个配置文件 `.act.config.json` 和一组卡片描述文件 `.vue` 、 `.css` 、 `.json` 等组成，目录结构如下所示：

```
.
├── dist                    // 编译结果文件夹（执行"编译"操作时自动生成）
│   ├── app.manifest        // 应用配置信息（命名格式固定）
│   ├── test_cube
|   |   ├── main.bin        // 编译产物的二进制文件
|   |   ├── main.json       // 编译产物的 JSON 格式文件
|   |   ├── main.mock       // mock.json 的编译产物
|   |   ├── main.js         // 编译产物的JS逻辑段，便于运行时排查JS段异常
|   |   └── main.zip        // 针对卡片所有产物的整体打包文件
├── test_cube
|   ├── main.vue            //【必需】卡片源码文件，文件名不可改变
|   ├── mock.json           //【可选】卡片 mock 数据
|   ├── manifest.json       //【必需】卡片编译配置文件，文件名不可改变
|   └── main.css            //【可选】卡片样式文件
└── .act.config.json        //【必需】工程配置文件，文件名不可改变
```

.act.config.json

`.act.config.json` 是卡片工程的配置文件，当前由蚂蚁动态卡片命令行工具生成，无需修改，不需关注。

> 说明
>
> `.act.config.json` 必须位于工程的根目录下。

```
//.act.config.json

{
  "type": "templates",        // 必填，工程类型，取值支持 templates(普通卡片)
}
```

manifest.json

`manifest.json` 是对应卡片的编译配置文件，当前由蚂蚁动态卡片命令行工具生成，无需修改，不需关注。

> 说明
>
> `manifest.json` 必须同对应的卡片 `main.vue` 位于同一路径下。

```
//        manifest.json
{
  "name": "my-card",        // 选填，卡片名称，卡片发布后以卡片后台 ID 为准。
  "version": "x.x.x",          // 选填，卡片版本，卡片发布后以卡片后台版本为准。
  "compilerType": 1,         // 选填，卡片编译模式，0（静态卡片） | 1（动态卡片，支持 JS，推荐使用），默认 0。
  "jsformat": 1,             // 选填，卡片 JS 编译格式，0（表达式导出） | 1（IIFE 导出，支持 JS import，推荐
使用），默认 0。
}
```

工程示例

test_cube.zip

5.1.2. 模板写法

模板的书写语法借鉴 VUE，精简后仅包含 <template>、<style>、和 <script> 三个字段。

<template> 字段

模板模式页面布局的相关逻辑应放置于 <template></template> 段内。

模板模式 <template> 段内支持的元素由卡片组件决定。

<style> 字段

样式相关的支持请参考 样式语法。

<script> 字段

JS 相关的逻辑放在 script 里。JS 相关的能力支持请参考 JS 能力。

5.1.3. 单位

本文介绍了蚂蚁动态卡片在数值、时间、长度和颜色等方面的定义。

数值单位

卡片中有一些属性需执行纯数值单位，比如 flex，lines 等，此时数值后不加 px、vw 等单位。

纯数值的取值范围和取值类型，请参见 通用样式 中各组件的属性定义。如果遇到非纯数值，则默认按照 0 处理。

示例：

```
lines:5;
flex:1;
```

时间单位

卡片有部分属性需要用时间描述，例如：animation-duration。卡片支持以秒、毫秒为时间单位。

示例：

```
animation-duration:2s;
```

长度单位

卡片有几种不同的单位表示长度，长度由一个数字和单位组成，数字和单位之间不能出现空格，长度单位分为内置单位和自定义单位。

内置单位

相对长度

相对长度单位指定了一个长度相对于另一个长度的属性。对于不同的设备相对长度更适用

- vw：viewpoint width，视窗宽度，1vw=视窗宽度的 1%。

- %：视窗高/宽度的百分比。

示例：

```
font-size: 20vw;
background-size:50%;
```

绝对长度

绝对长度单位是一个固定的值，它反应一个真实的物理尺寸。绝对长度单位视输出介质而定，不依赖于环境（显示器、分辨率、操作系统等）

- px：像素 (1px = 1/96th of 1in)
- rpx：以 750 为基准计算，2rpx = 1px

示例：

```
font-size: 17px;
line-height:40rpx;
```

单击此处 detailLength.zip 获取完整示例代码。

颜色单位

卡片中的颜色可以通过以下方法进行指定：

- 十六进制
- RGB 颜色
- 颜色名称

十六进制颜色

指定一个十六进制的颜色的组成部分有几种方式：

- #RRGGBB，其中 RR（红色），GG（绿色）和 BB（蓝色）。所有值必须介于 0 和 FF 之间。

例如，#0000FF 值呈现为蓝色，因为蓝色的组成设置为最高值 FF 而其他设置为 0。

- #RGB，其中 R（红色），G（绿色）和 B（蓝色）。所有值必须介于 0 和 F 之间。

例如，#00F 值呈现为蓝色，因为蓝色的组成设置为最高值 F 而其他设置为 0。

- 0xrrggbb，其中 rr（红色），gg（绿色）和 bb（蓝色）。所有值必须介于 0 和 ff 之间。

例如，0x0000ff 值呈现为蓝色，因为蓝色的组成设置为最高值 ff 而其他设置为 0。

示例：

```
background-color:#ff0000;
border-top-color:0x0000ff;
border-bottom-color:#00f;
```

RGB 颜色

指定一个 RGB 的颜色其组成部分有几种方式：

- RGB（红、绿、蓝）。每个参数（红色，绿色和蓝色）定义颜色的亮度，为 0 和 255 之间的整数。例如 RGB（0,0,255）值呈现为蓝色，因为蓝色的参数设置为最高值 255 而其他设置为 0。
- RGBA（红、绿、蓝，alpha）。Alpha 参数是一个介于 0.0（完全透明）和 1.0（完全不透明）之间的参数。例如 RGB（0,0,255,0.5）值呈现为半透明的蓝色。

示例：

```
background-color:rgb(0,0,255);
border-color:rgba(255,0,0,0.5);
```

颜色名称

卡片支持 147 种颜色名称定义，包括 17 个标准色和 130 个其他颜色。下表列出了所有颜色的名称和对应的十六进制值。

标准色

名称	十六进制
red	0xffff0000
blue	0xff0000ff
black	0xff000000
.....	.....

其他

名称	十六进制
red	0xffff0000
blue	0xff0000ff
black	0xff000000
.....	.....

示例

```
background-color:red;
```

> **说明**
> - 只有上面列出的颜色格式被支持，其他的颜色格式均不被支持。
> - 如果遇到不被支持的颜色格式，卡片将按照默认透明色处理（部分样式有独立的默认色规则除外）。

单击此处 detailColor.zip 获取完整示例代码。

附录-色值定义

名称	色值
red	0xffff0000
darkred	0xff8b0000
tan	0xffd2b48c

linen	0xfffaf0e6
sienna	0xffa0522d
indianred	0xffcd5c5c
teal	0xff008080
grey	0xff808080
green	0xff008000
gray	0xff808080
darkgrey	0xffa9a9a9
darkgreen	0xff006400
beige	0xfff5f5dc
orange	0xffffa500
darkgray	0xffa9a9a9
orangered	0xffff4500
khaki	0xfff0e68c
seagreen	0xff2e8b57
gold	0xffffd700
darkorange	0xffff8c00
darkkhaki	0xffbdb76b
indigo	0xff4b0082
goldenrod	0xffdaa520
maroon	0xff800000

gainsboro	0xffdcdcdc
lime	0xff00ff00
greenyellow	0xffadff2f
darkgoldenrod	0xffb8860b
slategrey	0xff708090
slategray	0xff708090
salmon	0xfffa8072
darkseagreen	0xff8fbc8f
seaShell	0xfffff5ee
darksalmon	0xffe9967a
tomato	0xffff6347
thistle	0xffd8bfd8
darkslategrey	0xff2f4f4f
cyan	0xff00ffff
forestgreen	0xff228b22
dimgrey	0xff696969
darkslategray	0xff2f4f4f
mistyrose	0xffffe4e1
dimgray	0xff696969
darkcyan	0xff008b8b
black	0xff000000
magenta	0xffff00ff

limegreen	0xff32cd32
coral	0xffff7f50
darkmagenta	0xff8b008b
azure	0xfff0ffff
blue	0xff0000ff
oldlace	0xfffdf5e6
cornsilk	0xfffff8dc
darkblue	0xff00008b
skyblue	0xff87ceeb
firebrick	0xffb22222
orchid	0xffda70d6
lightgrey	0xffd3d3d3
lightgreen	0xff90ee90
lightyellow	0xfffffffe0
lightgray	0xffd3d3d3
darkorchid	0xff9932cc
royalblue	0xff4169e1
aqua	0xff00ffff
steelblue	0xff4682b4
bisque	0xffffe4c4
crimson	0xffdc143c

slateblue	0xff6a5acd
dodgerblue	0xff1e90ff
blanchedalmond	0xffffebcd
lightseagreen	0xff20b2aa
lightslategrey	0xff778899
lightslategray	0xff778899
brown	0xffa52a2a
lightsalmon	0xffffa07a
snow	0xfffffafa
lightcyan	0xffe0ffff
rosybrown	0xffbc8f8f
sandybrown	0xfff4a460
darkslateblue	0xff483d8b
yellow	0xffffff00
lightcoral	0xfff08080
mintcream	0xfff5fffa
aquamarine	0xff7fffd4
saddlebrown	0xff8b4513
honeydew	0xfff0fff0
pink	0xffffc0cb
lightblue	0xffadd8e6

cadetblue	0xff5f9ea0
wheat	0xfff5deb3
lawngreen	0xff7cfc00
white	0xffffffff
aliceblue	0xfff0f8ff
chocolate	0xffd2691e
yellowgreen	0xff9acd32
moccasin	0xffffe4b5
navy	0xff000080
chartreuse	0xff7fff00
ivory	0xfffffff0
palegreen	0xff98fb98
lavender	0xffe6e6fa
hotpink	0xffff69b4
olive	0xff808000
fuchsia	0xffff00ff
mediumseagreen	0xff3cb371
silver	0xffc0c0c0
olivedrab	0xff6b8e23
darkturquoise	0xff00ced1
turquoise	0xff40e0d0

violet	0xffee82ee
violetred	0xffd02090
darkviolet	0xff9400d3
palegoldenrod	0xffeee8aa
whitesmoke	0xfff5f5f5
springgreen	0xff00ff7f
burlywood	0xffdeb887
peru	0xffcd853f
floralwhite	0xfffffaf0
lightpink	0xffffb6c1
darkolivegreen	0xff556b2f
ghostwhite	0xfff8f8ff
mediumblue	0xff0000cd
mediumorchid	0xffba55d3
lightsteelblue	0xffb0c4de
lightslateblue	0xff8470ff
transparent	0x00000000
deepskyblue	0xff00bfff
lightskyblue	0xff87cefa
lightgoldenrodyellow	0xfffafad2
plum	0xffdda0dd

mediumaquamarine	0xff66cdaa
mediumslateblue	0xff7b68ee
blueviolet	0xff8a2be2
midnightblue	0xff191970
deeppink	0xffff1493
lemonchiffon	0xfffffacd
antiquewhite	0xfffaebd7
paleturquoise	0xffafeeee
powderblue	0xffb0e0e6
navajowhite	0xffffdead
mediumspringgreen	0xff00fa9a
cornflowerblue	0xff6495ed
palevioletred	0xffdb7093
mediumvioletred	0xffc71585
purple	0xff800080
rebeccapurple	0xff663399
lavenderblush	0xfffff0f5
mediumturquoise	0xff48d1cc
peachpuff	0xffffdab9
mediumpurple	0xff9370db
papayawhip	0xffffefd5

5.1.4. 数据绑定

本文介绍了在模板模式中进行数据绑定的形式。

单数据源绑定

参照 Vue 格式，模板模式下单数据绑定支持插值和指令两种形式。

- 插值格式仅支持单独使用，不支持混写，如 <text>ab{{var1}}cd</text>。
- 指令形式支持简写格式。

使用时，可绑定的数据字段（即下表中的 name）由当前组件决定，被绑定的数据变量（即下表中的 variable）支持采用表达式的方式进行定义。

类型	格式	简写	级联	示例
插值	{{variable}}	无	. 或 []	<text>{{var1.var2}}</text>
指令	v-bind:name="variable"	:name="variable"	. 或 []	<text :value="var1[num1]"></text>

文本类组件（如 text）填充内容时支持的格式包括：

- [1] <text:value="var"></text> （其中 var="hello_1"）
- [2] <text>{{var}}</text> （其中 var="hello_2"）
- [3] <text>hello_3</text>

解析优先级为 [1] < [2] < [3]，即最终将显示 `hello_3` 。

双数据源绑定

模板支持同时代入两组数据作为待绑定数据源，主要解决实际业务开发过程中 native 侧向模板内注入额外的控制数据，遇到此类需求时，便以启用该功能。

模板注入的数据会和服务器下发的 mock 数据进行合并。注入的数据优先级更高，如有相同字段会覆盖 mock 数据中的对应字段，使用方法和 mock 数据中的字段相同。

```
//注入的数据
{
title: "title"
}

// 数据提取方式
<text :value=title></text>
```

示例代码

单击此处 detailBindData.zip 获取完整示例代码。

5.1.5. 事件绑定

本文介绍了在蚂蚁动态卡片中可以绑定的事件种类和绑定方法。

通用事件

在卡片中，可以绑定在模板中的事件有以下两种：

- 点击：click
- 长按：longpress

模板用法

参照 VUE 格式，模板模式下事件绑定仅支持指令形式。指令形式支持简写格式。使用时，可绑定的事件名（即下文示例代码中的 @click）由卡片注册的组件决定，更多关于卡片支持注册的组件的信息，请参见 卡片组件。

- 被绑定的处理函数参数（即下文示例代码中的 param）支持表达式。
- 可在 <script> 段中定义对应的业务 JS 方法，对方法的参数无限制。

```
<template>
...
        <text @click="click(param)"></text>
...
</template>

<script>
    export default {
        data: {
        },
        beforeCreate() {
        },
        methods: {
            click(p) {
                console.info("onclick");
            }
        }
    }
</script>
```

示例代码

单击此处 detailBindEvent.zip 获取完整示例代码。

5.1.6. 逻辑渲染

本文介绍了逻辑渲染的种类和相应的指令。

条件渲染

v-show

`v-show` 指令用于条件性地渲染一个节点内容。不管 `v-show` 指令的结果是 `true` 还是 `false`，该节点都会进行渲染。

- 当 `v-show` 结果为 `true` 时，相当于给该节点的 CSS 样式增加一个 `display: flex`。
- 当 `v-show` 结果为 `false` 时，相当于给该节点的 CSS 样式增加一个 `display: none`。

```
<div class="div" v-show="exist(exist4)"></div>
```

v-if

`v-if` 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 `true` 的时候被渲染。

- 可以使用 `v-else-if` 充当 `v-if` 的“else-if 块”，可以连续使用。
- 可以使用 `v-else` 充当 `v-if` 的“else 块”。

```
<div class="div" v-if="exist(a)">
  ...
</div>
<div class="div" v-else-if="exist(b)">
  ...
</div>
<div class="div" v-else>
  ...
</div>
```

重要

- `v-else-if` 必须紧跟在 `v-if` 块后面，否则将不被识别。
- `v-else` 必须紧跟在 `v-if` 或 `v-else-if` 块后面，否则将不被识别。
- `v-if` 不能写在模板的根节点上。

列表渲染

vfor

`v-for` 指令可以基于一个数组 `/number/` 对象来渲染一组数据。目前支持的写法有 6 种：

- v-for="index in 10"
- v-for="index in number"
- v-for="item in array"
- v-for="(item,index) in array"
- v-for="item in object"
- v-for="(item,key) in object"

```
<div class="vforStyle" v-for="(value, index) in obj">
    <text class="content">{{index + value}}</text>
</div>
```

说明

- `v-for` 指令不支持嵌套使用，即 `v-for` 下面不能再使用 `v-for` 进行渲染。
- 当基于对象来渲染数据时，对象显示的先后顺序不可控。如需按照确定的顺序展示，请遍历 `array/number` 数据。
- `v-for` 不能写在模版的根节点上。

示例代码

单击此处 detailLogicalRender.zip 获取完整示例代码。

5.2. 卡片组件

5.2.1. div

 可定义文档中的分区（division）或节（section）。 标签可以把文档分割为独立的、不同的部分。它可以用作严格的组织工具，并且不使用任何格式与其关联。

嵌入组件支持

可以嵌套任何其他组件。

样式

 组件支持通用样式中的所有样式。

属性

无。

事件

 组件支持 通用事件 中的所有事件。

示例

```
<div>
    <div class="box"></div>
</div>


.box {
    border-width: 2px;
    border-style: solid;
    border-color: #BBB;
    width: 250px;
    height: 250px;
    margin-top: 250px;
    margin-left: 250px;
    background-color: #EEE
  }
```

示例代码

单击此处 detailDiv.zip 获取完整示例代码。

5.2.2. text

<text> 是蚂蚁动态卡片引擎内置的组件，用来将文本按照指定的样式渲染出来。<text> 只能包含文本值，可以使用 {{}} 标记插入变量值作为文本内容。

嵌入组件支持

不可嵌套任何其他组件。

样式

<text> 组件支持 通用样式 中的所有样式，此外，还支持其他特殊样式：

字体相关

属性	描述	值类型	默认值	可选值	写法	备注
font-size	文字大小	长度单位	16px		font-size: 10px;	

font-weight	字体粗细程度	string	normal	normal，bold，100，200，300，400，500，600，700，800，900； normal 等于 400。 iOS 支持 9 种值，Android 支持 normal、bold 和介于（normal，bold）之间的三种，对应的值为 400 为 normal 效果、700 及以上为 bold 效果、 400 和 700 之间统一为 FakeBold 效果。	font-weight: bold; font-weight: 700;	
font-style	字体样式	string	normal	normal，italic	font-style: normal;	
font-family	设置字体	string	平台默认字体		font-family: PingFangSC-Regular;	不保证该设置在不同平台、设备间的一致性，如所设置的字体在平台上不可用，将会降级到平台的默认字体。

排版相关

属性	描述	值类型	默认值	可选值	写法	备注
lines	文本行数	int	0，表示不限制行数		lines: 10;	
text-align	字体对齐方式	string	left	left, center, right	text-align: center;	
text-overflow	设置内容超长时的省略样式	string	clip	clip, ellipsis	text-overflow: clip;	

line-height	设置文本行高	长度单位+数值	0		line-height: 12px;	• 为数值时为 fontSize *value。 • 为长度单位 px 后缀时为 value 本身。 • 为长度单位的百分比时为 fontSize *value。
white-space	控制文本中的换行和空白策略	string	pre-wrap	• normal：空白字符折叠，文本自动换行，不根据内容中换行符或段落换行。 • nowrap：空白字符折叠，文本不换行，一行显示，可超出背景框。 • pre：空白字符不折叠，根据文本内容段落换行，每段内不自动换行（每段一行，可超出背景框）。 • pre-wrap：空白字符不折叠，根据内容段落换行，且段落内自动换行。pre-line：空白字符折叠，根据内容段落换行，且段落内自动换行。	white-space: nowrap;	-

word-wrap	控制折行方式（单词拆分与否）	string	break-word	• normal：单词结束处折行，可超出背景框 • break-word：单词结束处折行，但是长度仍不够时，可以在单词中间折行 • anywhere：可在任意处折行。	word-wrap: beak-word:	-
word-break	控制折行时单词如何拆分	string	无	• normal：单词不拆分折行，可超出背景框。 • break-all：单词可拆分折行。 • keep-all：与 normal 相同。	word-break;	不区分中英文，和中英混排的情况。
letter-spacing	控制字符间隔，可正，可负	string	0	normal：无额外空间。单值长度单位：可正可负。	letter-spacing:5px;	-
text-indent	首行文本的缩进，可正，可负，可以是百分比（父元素宽度的百分比）	string	0	单值长度单位+百分比：可正可负。	text-indent:30%;	-
				baseline|sub|super|长度|百分比|top|bottom|middle|top|bottom 重要 基线和字体相关，请谨慎使用。		

vertical-align	文本垂直方向对齐样式	string	baseline	• `middle` 使元素的中部与父元素的基线加上父元素 x-height 的一半对齐。 • `baseline` 使元素的基线与父元素的基线对齐。 HTML 规范没有详细说明部分可替换元素的基线，如 `<textarea>`，意味着这些元素使用此值的表现因浏览器而异。 • `sub` 使元素的基线与父元素的下标基线对齐。 • `super` 使元素的基线与父元素的上标基线对齐。 • `top` 使元素及其后代元素的顶部与整行的顶部对齐。 • `bottom` 使元素及其后代元素的底部与整行的底部对齐。没有基线的元素，使用外边距的下边缘替代。	vertical-align:middle	-

效果相关

属性	描述	值类型	默认值	可选值	写法	备注
color	字体颜色	色彩单位	0x000000		color:red; color:#333;	

					color:rgb(255,0,255);	
text-decoration	字体装饰	string	none	underline，none，line-through，overline	text-decoration: underline;	-
text-shadow	字体阴影	长度单位&色彩单位		支持格式 ${x}${y}${size}${color}，x，y，size 满足长度单位，color 满足色彩单位。	text-shadow: 2px 2px 3px gray;	颜色默认字体颜色 color。 x、y 是必需参数，其他可选。
text-shadow-color	字体阴影颜色	色彩单位	与 color 相同		text-shadow-color: blue;	可选参数
text-shadow-offset	字体阴影偏移	长度单位		-	text-shadow-offset: 2px 2px;	必需参数
text-shadow-radius	字体阴影半径	长度单位	0		text-shadow-radius: 3px;	可选参数

属性

属性	描述	值类型	默认值	写法	备注
value	组件的值，文本内容	string		<text value="文本内容字符串"></text>	
line-space	行间距，如 4px	长度单位		<text line-space="4px"></text>	

事件

<text>组件支持 通用事件 中的所有事件。

示例

```
<text class="text">
                Cube 引擎是一套简单易用的跨平台开发方案，能以 Web 的开发体验构建高性能、可扩展的原生应用。Vue 是一个轻量并且功能强大的渐进式前端框架。
</text>


.text {
    lines: 3;
    color: #666666;
    font-size: 32px
  }
```

与 Web 端差异

蚂蚁动态卡片与 Web 端差异主要是以下两点，更多细节请参见下表。

- **word-break 是否区分中英文处理（蚂蚁动态卡片不支持）。**
- **长词超出背景框时不区分中英混排。**

Web	蚂蚁动态卡片
word-wrap 不写的情况与 word-wrap:normal 相同。	word-wrap 不写的情况与 word-wrap:break-word 相同。
word-wrap:normal 单词不折行，超出背景框显示（识别单词的原则为连续英文字符为单词）。	word-wrap:normal 单词不折行，超出背景框显示（识别单词原则为连续英文字符，包括中间混排中文）。
word-break:normal 中文拆分折行，英文单词不拆分折行，可超出背景框显示。	word-break:normal 中文拆分折行，英文单词不拆分折行，可超出背景框显示。
word-break:keep-all 中文不拆分折行，英文单词不拆分折行，可超出背景框显示。	word-break:keep-all 与 normal 相同。
-	letter-spacing 在 Android 5.0 以上系统支持。

示例代码

- 单击此处 detailFontSize.zip获取关于字体大小的完整示例代码。
- 单击此处 detailFontWeight.zip获取关于字体粗细的完整示例代码。
- 单击此处 detailTextAlign.zip获取关于字体对齐方式的完整示例代码。

5.2.3. image

<image> 是蚂蚁动态卡片引擎内置的组件，用来将单张图片按照指定的样式渲染出来。

嵌入组件支持

不可嵌套任何其他组件。

样式

<image> 组件支持 通用样式 中的所有样式。

属性

属性	描述	值类型	默认值	可选值	写法
src	组件图片在线地址或本地地址，或 Base64 编码的数据	string		• URL(“https:/xxx”) CDN 地址。 • URL(“./xxx”) 离线包相对地址。 • URL(“data:”) Base64 编码。	src="https://gw-office.alipayobjects.com/basement_prod/e047f6c8-dc14-481f-a22c-8dd9012b01a3.png"

resize	图片位置	string	stretch	stretch|cover|contain|top|bottom|center|left|right|top left|top right|bottom left|bottom right，超出此范围为默认 cover。	resize: contain
placeholder	组件占位图的在线地址或 Base64 编码的数据	string		• URL(“https:/xxx”) CDN 地址。 • URL(“data:”) Base64 编码。	placeholder="https://gw-office.alipayobjects.com/basement_prod/e047f6c8-dc14-481f-a22c-8dd9012b01a3.png"

事件

<image> 组件支持 通用事件 中的所有事件。

示例

```
<image class="image" resize="contain" src="https://gw.alicdn.com/tfs/TB1dZ4WowoQMeJjy0FnXXb8gFXa-950-1267.jpg">
</image>

.image {
  width: 600px;
  height: 400px
}
```

> **说明**
> - <image>组件的图片不支持 SVG 格式。
> - 以下几种情况下载的是原图。
>   - resize 不为 cover/contain/stretch。
>   - 节点不写 width 或者 height。

单击此处 detailImage.zip 获取完整示例代码。

5.2.4. richtext

<external-richtext> 是蚂蚁动态卡片引擎内置的组件，用来渲染富文本。

嵌入组件支持

不可以嵌套任何其他组件。

样式

<external-richtext>组件支持 通用样式 中的所有样式，并且支持部分特殊样式。

格式

支持 HTML 标签格式，目前包含以下与字体相关的标签。

标签	描述	写法	备注

br	换行	 To break lines in a paragraph, use the br tag. 	-
span	用来组合文档中的行内元素	`<span>some text.</span>` some other text.	-
div	把文档分割为独立的、不同的部分	`<div style="color:#00FF00">` This is a header This is a paragraph. `</div>`	-
b	加粗文本	这是普通文本 - 这是粗体文本。	-
del	删除文本	a dozen is `<del>20</del>` 12 pieces	-
h1	标题 1	这是标题 1	-
h2	标题 2	这是标题 2	-
h3	标题 3	这是标题 3	-
h4	标题 4	这是标题 4	-
h5	标题 5	这是标题 5	-
h6	标题 6	这是标题 6	-
i	斜体文本	邮箱斜体 cn42ducn4***@163.comt 3@42du.online	-
p	定义段落	This is some text in a very short paragraph	-

img	定义图片	src：下载链接 width/height：图片绘制宽/高 	宽高默认值与字体高度相同。 设置宽高大于行高时，绘制效果上不支持把行高撑开。
a	定义链接	href：链接跳转地址 <a href='https://www.alipay.com'>	-

字体相关

属性	描述	值类型	写法	备注
font-size	文字大小	长度单位	<span style=\"font-size:30px;\">内联30px	-

效果相关

属性	描述	值类型	写法	备注
color	字体颜色	色彩单位	<span style=\"color:#F00\">内联#F00	-
font-weight	字重	string	<span style=\"font-weight:100\">内联#F00	取值和 text 标签中基础样式的 样式 相同。
font-family	字体	string	<span style=\"font-family:thinFont\">字体瘦体	取值和 text 标签中基础样式的 样式 相同。

属性

属性	描述	值类型	默认值	写法	备注
text	组件的值，文本内容	string		<external-richtext text="richtextContent"></external-richtext>	-

line-space	行间距，如 4px	长度单位		<external-richtext line-space="4px"> </external-richtext>	-
detectEmotionEmoji	是否检测自定义 Emoji 表情	1/0	0	<external-richtext text="richtextContent" detectEmotionEmoji="1" > </external-richtext>	-
linkColor	设置链接字体颜色（a 标签）	色彩单位	0xff108ee9	<external-richtext text="richtextContent" linkColor="#FF0000"> </external-richtext>	-
highlightedColor	设置链接点击高亮颜色	色彩单位	0xffa9a9a9	<external-richtext text="richtextContent" highlightedColor="#0000FF" > </external-richtext>	-

说明

- 为 img 标签设置的 height 大于行高时，绘制效果不会把行高撑开，默认与字体高度相同。
- 为 img 标签设置 width 和 height，会把图片压缩填满。
- a 标签链接字体颜色 span 设置优先级高于属性 linkcolor 设置。
- highlightedColor 控制高亮颜色，包括 a 标签链接。如果不设置，默认灰色。
- 表情大小与字体高度相同。

事件

<richtext> 组件支持 通用事件 中的所有事件。

示例

```
<external-richtext
:text="richTextContent"
:line-space="4px"
></external-richtext>

data: {
    richTextContent:
      '<span style=\"font-size:30px;\">内联30px</span><span style=\"font-size:30rpx;\">内联30rpx</span><span style=\"color:#F00\">内联#F00</span>
      <span>全局30px,color:#F00</span>
      <span>全局30rpx,color:#0F0</span>
      <span style=\"font-size:30px;color:#00F\">内联30px,color#0F0</span><span>全局20px,color#0F0</span>
      <div style=\"color:#F00\"><div><div>最外部#F00</div></div></div>
      <div><div><div style=\"color:#F00\">最内部#F00</div></div></div>
      <b>b</b><del>del</del><div>div</div><h1>h1</h1><h2>h2</h2><h3>h3</h3><h4>h4</h4><h5>h5</h5><h6>h6</h6><i>i</i><p>p</p><span>span</span>'
  }
```

示例代码

单击此处 detailRichtext.zip 获取完整示例代码。

5.2.5. slider

<slider> 组件用于在一个视图中交替展示多个图片。

嵌入组件支持

<slider> 组件支持更多高级功能，只能嵌套 <cell> 子组件使用。<cell> 用于定义 slider 中的子列表项。蚂蚁动态卡片引擎会对 <cell> 进行高效的内存回收以达到更好的性能。

样式

通用样式 中部分能力不支持，包括盒子模型中的 padding、布局中的 flex 容器、flex 成员、背景中的 background-image 相关、hover、动画。

属性

属性	描述	值类型	默认值	写法	备注
auto-play	设置是否自动轮播几张图片	boolean	true	auto-play="true"	无
interval	设置自动轮播图片的时间间隔，单位毫秒，当 auto-play 为 true 时生效	number	500ms	interval="500"	设置小于 500ms 的值，表现为 500ms
infinite	设置是否循环	boolean	true	infinite="true"	无
show-indicators	设置是否显示 indicator	boolean	false	show-indicators="true"	无
scrollable	设置是否可以通过手势操作滑动切换页面	boolean	true	scrollable="true"	无
index	设置显示 slider 的第几个页面	number	0	index="2"	无
previous-margin	设置透出前一个页面的距离	长度单位	0	previous-margin="200px"	不能与 `infinite="true"` 同时使用
next-margin	设置透出后一个页面的距离	长度单位	0	next-margin="200px"	不能与 `infinite="true"` 同时使用

事件

不支持 通用事件，支持的特定事件如下：

名称	描述	参数
on-change	当轮播索引改变时，触发该事件	index：展示的图片索引，number 类型

示例

```
<template>
    <div class="root">
        <slider class="testSlider" :index="index" show-indicators="false" scrollable="true"
duration="1000" auto-play=true  @on-change="onChange(index)" >
            <cell class="cell" v-for="(item, i) in imageList">
                <image class="image" resize="contain" :src="item.src"></image>
            </cell>
        </slider>
    </div>
</template>
<script>
export default {
    data: {
    },
}
</script>>
<style>
    .root {
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        background-color: white;
        width: 100%;
    }

    .image {
        width: 100%;
        height: 100%;
    }

    .cell {
        width: 100%;
        height: 100%;
        background-color: blue;
        flex-direction: column;
        align-items: center;
    }

    .testSlider {
        width: 100%;
        background-color: red;
        margin-top: 100px;
        height: 400px;
    }
</style>
```

5.2.6. scroller

<scroller> 组件是一个容纳子组件进行横向或竖向滚动的容器组件，拥有平滑的滚动和高效的内存管理，适用于长列表的展示。

嵌入组件支持

支持任意组件嵌入。

样式

不支持 通用样式 中的部分能力，包括盒子模型中的 padding、布局中的 flex 容器、flex 成员、背景中的background-image 相关、hover、和动画。

属性

属性	描述	值类型	默认值	可选值	写法	备注
show-scrollbar	设置组件是否显示滚动条	boolean	false	-	show-scrollbar="true"	-
scroll-direction	设置组件滚动方向	string	vertical	vertical horizontal	scroll-direction="vertical"	-
upper-threshhold	设置组件距离顶部/左部多远时，触发事件	string	50 px	-	upper-threshhold="50px"	-
lower-threshhold	设置组件距离底部/右部多远时，触发事件	string	50 px	-	lower-threshhold="50px"	-
offset-accuracy	设置组件滚动过程中 callback 的频率	string	10 px	-	offset-accuracy="10px"	-
allow-bounce	是否允许有弹性效果	boolean	false	-	allow-bounce="true"	10.2.28 支持
always-bounce	内容不满时是否允许滚动（注：必须在 `allow-bounce` 为 true 时才生效）	boolean	false	-	always-bounce="true"	10.2.28 支持

事件

不支持 通用事件，支持的特定事件如下表所示：

名称	描述	参数
on-scroll	在滚动中触发	contentSize：内容区宽高 contentOffset：显示区域偏移量
on-scrollstart	当滚动开始时触发	contentSize：内容区宽高 contentOffset：显示区域偏移量

on-scrollend	当滚动结束时触发	contentSize：内容区宽高 contentOffset：显示区域偏移量
on-scrolltoupper	当滚动到距离顶部小于阈值时触发	-
on-scrolltolower	当滚动到距离底部小于阈值时触发	-

示例

```
<template>
    <scroller class="root" ref="scroller" show-scrollbar="true" scroll-direction="horizontal" @on-
scroll="onScroll()" @on-scrollstart="onScrollStart()" @on-scrolltoupper="onScrollToUpper()" @on-sc
rollend="onScrollEnd()" @on-scrolltolower="onScrollToLower()" offset-accuracy="40px">
        <text class="message" :value="message" @click="onClick()"></text>
        <image class="image"
src="https://gimg2.baidu.com/image_search/src=http%3A%2F%2F1812.img.pp.sohu.com.cn%2Fimages%2Fblog%2
09%2F11%2F18%2F18%2F8%2F125b6560a6ag214.jpg&refer=http%3A%2F%2F1812.img.pp.sohu.com.cn&app=2002&size
999,10000&q=a80&n=0&g=0n&fmt=jpeg?sec=1623901796&t=5e3520844113908195604 2a69907f7f5"></image>
        <text class="message" :value="message" @click="onClick()"></text>
        <text class="message" :value="message" @click="onClick()"></text>
        <text class="message" :value="message" @click="onClick()"></text>
    </scroller>
</template>

<script>
    export default {
        data: {
            message: 'Hello Cube 1'
        },
        beforeCreate() {
            this.data.message = 'Hello Cube 2'
        },
        didAppear() {

        },
        methods: {
            onClick() {
                //NOTE: console log 用 act debug 查看
                console.log('invoke on-click event');
            },

            onScroll(data) {
                console.log("onScroll---" + JSON.stringify(data));
            },

            onScrollStart(data) {
                console.log("onScrollStart---" + JSON.stringify(data));
            },

            onScrollEnd(data) {
                console.log("onScrollEnd---" + JSON.stringify(data));
            },

            onScrollToUpper() {
                console.log("onScrollToUpper---");
            },
```

```
            onScrollToLower() {
                console.log("onScrollToLower---");
            },
        }
    }
</script>>

<style>
    .root {
        display: flex;
        flex-direction: row;
        align-items: center;
        background-color: white;
        width: 100%;
        height: 400px;
    }
    .message {
        color: black;
        font-size: 50rpx;
    }
    .image {
        width: 200px;
        height: 400px;
    }
</style>
```

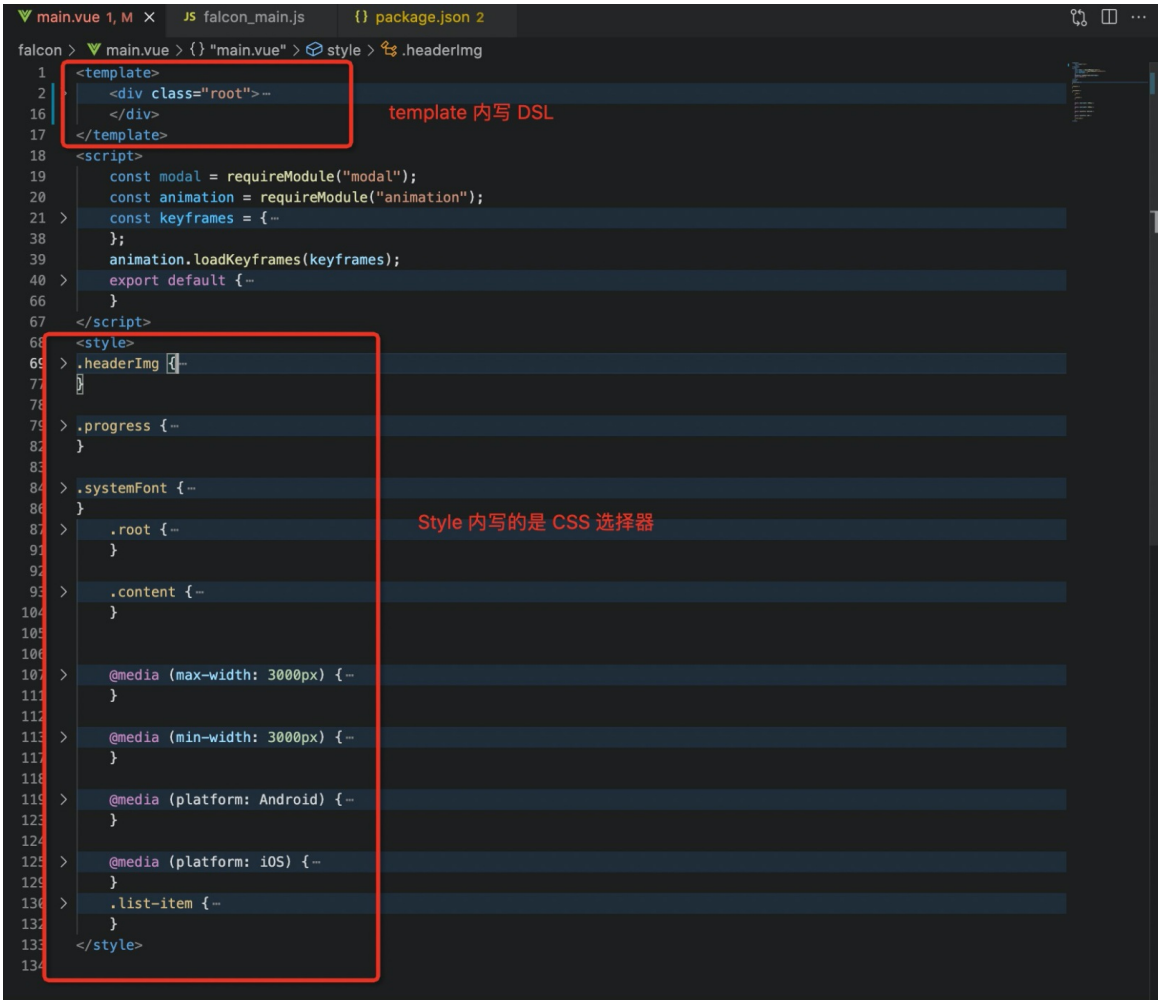
> **重要**
>
> 同方向的 scoller 不支持嵌套使用。

5.3. 卡片样式

5.3.1. 样式语法

在模板模式下，页面样式相关 CSS 放置于 <style></style> 段内，<style></style> 段内支持的样式由 通用样式 决定。

模板模式 <template></template> 段内推荐通过 class 对样式进行设置，例如 `<text class="mytext">□` 。



CSS 样式

支持 class、id、type 选择器，不支持父子、状态等更复杂的组合。支持的三类示例如下：

```
// class
.class {
}

// id
#id {
}

// type
div {
}
```

内联样式

模板模式提供运行时样式注入能力，主要通过组件的内置属性 **style** 字段来实现。

内联样式的书写规范同前端流行框架（VueJS 和 ReactJS）的规范一致，同时支持绑定和非绑定两种格式。

- 绑定内联样式
  - 待绑定的样式字段应统一收敛至一个 JSONObject 内。
  - 待绑定的样式字段的 KEY 应符合驼峰命名规范（如 background-color 应转换为 backgroundColor）。
- 非绑定内联样式

- 待绑定的样式字段应按照 CSS 的书写规范收敛至一个字符串内。
- 待绑定的样式字段的 KEY 应按照 CSS 规范单词间通过连字符-进行拼接（如 background-color）。

样式优先级从高至低依次为 inline style、id、class、type。

```
// main.vue
<template>
  <div class="root">
    <div class="div1" :style="style"></div>
    <div class="div2" style="width: 100px; background-color: blue"></div>
  </div>
</template>

// mock.json
{
  "style": {
    "height": "100px",
    "backgroundColor": "red"
  }
}
```

模板在内联样式上接收的属性值是一个 JS object。

```
<div class="root" :style="{height: height}"> // 上文示例
```

动态绑定 class

组件的 CSS 样式，可以动态绑定不同的选择器（selector）。

```
<div :class="mydiv">
```

媒体查询 @media

模板模式中引入 CSS 规范内的媒体查询能力，主要用于处理移动端 UI 适配相关工作。

相对于 CSS 规范，模板模式中支持的媒体查询能力有限，使用时也有一些特定的用法，具体可参考以下几个方面。媒体查询相关信息可参考 @media 介绍。

- 媒体类型

  无需填写，默认使用 all。

媒体类型	是否支持
all	是
screen	否
print	否
speech	否

- 媒体特性

  卡片中，媒体特性主要以固件特性为基础进行设计，这点同前端浏览器不同。

媒体特性	取值	说明

platform	ios | android	针对平台适配，使用时同 CSS 规范有所区别，@media 后直接设置平台值即可，例如 @media android。
support	safearea	针对 iOS 平台屏幕安全区域适配。

- 媒体运算符

运算符	是否支持
and	是
not	否
only	否

下面是结合 CSS 特性与 @media 媒体查询能力进行样式适配的示例。

```
<template>
    <div class="banner"></div>
</template>
<style>
    @media android {
        .banner {
                  width: 100px;
                        height: 100px;
            background-color: #00fff0;
        }
    }
    @media ios and (support: safearea) {
        .banner {
            width: 100px;
                        height: 100px;
            background-color: #00fafb;
        }
    }
    .banner {
              width: 100px;
        height: 100px;
        background-color: green;
    }
</style>
```

样式导入

在导入样式之前，先了解以下两种不同类型的样式：

- 导入样式：存在于 .css 文件中的可供统一管理、导入的样式。
- 限定样式：存在于 .vue 文件中 <style></style> 段内仅作用于本模板的样式。

语法格式

样式导入的语法格式如下：

```
<style src="[.css文件相对路径]" />
```

- 文件结构

```
.
└── template_name                                    //                    模板文件夹
以模板ID命名）
    ├── main.vue                                        //                    模板
局、样式描述文件
    ├── manifest.json                                  //                    模板配
文件
    └── mock.json                                       //                    模板
供绑定的测试数据
    └── common.css                                    //                    模板公共
式文件
```

- 模板代码

```
<template>
        ... [模板布局相关描述]
</template>

<style src="./common.css" />

<style>
        ... [仅作用在本模板内的样式]
</style>
```

层叠规则

模板模式在对 .vue 文件内涉及的样式资源进行编译时，仅对 selector 相同的样式字段进行层叠整合，不同的 selector 会完整保留。

导入样式 + 限定样式 = 层叠结果，如下表所示。

导入样式	限定样式	层叠结果
`.special { color: red; width: 100px; } .others { color: red; width: 100px; }`	`.special { background-color: red; width: 200px; } .scoped { background-color: red; width: 200px; }`	`.special { color: red; background-color: red; width: 200px; } .scoped { background-color: red; width: 200px; } .others { color: red; width: 100px; }`

代码示例

下载代码示例 FalconDemo。

5.3.2. 通用样式

5.3.2.1. 背景

蚂蚁动态卡片提供了几种背景元素控制属性。

背景样式

指定一个元素的背景样式有以下几种方式：

- background-color，定义元素的背景颜色。

属性	值类型	默认值	写法
background-color	色彩单位	transparent	background-color:red;

- background-image，定义元素的背景图像。

属性	值类型	默认值	可选值	写法	备注

background-image	string	无	`url("https://xxx")` CDN 地址；`url("./xxx")` 离线包相对地址；`url("data:")` base64 编码；
			linear-gradient(s1,s2,...,slast),第一段（渐变角度设置，取值为具体对角度值以 deg 结尾，或者为方向描述，包括 top、to top、right、to right、bottom、to bottom、left、to left），第二段（渐变起始颜色设置，色彩单位；如果有百分比值，必须为 0%），中间段（渐变过程颜色设置，色彩单位；支持设置百分比值，必须为线性递增方式，不提供百分比时，颜色占比情况将被均分），最后一段（渐变终止颜色设置，色彩单位；如果有百分比值，必须为 100%）；
			none：清除背景；
				background-image: linear-gradient(45deg, red 0%, #333 50%, rgb(255,0, 255) 80%, green 100%);
				background-image: linear-gradient(to top,red, #333, rgb(255, 0, 255), green);
				background-image: url("https://gw-office.alipayobjects.com/basement_prod/e047f6c8-dc14-481f-a22c-8dd9012b01a3.png");
					无

background-size	string 或长度单位	auto	单描述值（cover，contain，auto）； ${x}px ${y}px 双精确值长度单位+百分比； ${x}px 单值长度单位+百分比；	background-size:contain; background-size:100px 200px;	精确值或百分比只单值时，另外一个值默认 auto。
background-position	string	0	单描述值（top，right，bottom，left，center）； 双描述值（bottom right）； ${x}px 单值长度单位/${y}px 单值长度单位+单描述值； ${x}px 单值长度单位+百分比；${y}px 单值长度单位+百分比	background-position:top; background-position:bottom right; background-position:30px left; background-position:100px; background-position:50px 50px;	单值时，另一个值默认居中。
background-repeat	string	repeat	repeat-x, repeat-y, no-repeat, repeat	单值：background-repeat: repeat-x; 双值：background-repeat: repeat no-repeat; 其中 x 轴描述包含（no-repeat、repeat）y 轴描述包含（no-repeat、repeat）	无 无

- background 简写方式：
  - background-color 和 background-image 相关样式可合并简写，无关先后顺序；
  - background-image 相关样式可合并简写，例如 background-image 和 background-repeat；
  - 支持使用 `none` 清除背景；

示例

```
background:url('https://img.alicdn.com/tfs/TB1uCUdfND1gK0jSZFyXXciOVXa-151-164.png') repeat-x;
background:url('https://img.alicdn.com/tfs/TB1uCUdfND1gK0jSZFyXXciOVXa-151-164.png') #f0f no-repea
t;
background:url('https://img.alicdn.com/tfs/TB1uCUdfND1gK0jSZFyXXciOVXa-151-164.png') #00f repeat-x
bottom;
background:#ff0 url('https://img.alicdn.com/tfs/TB1uCUdfND1gK0jSZFyXXciOVXa-151-164.png') repeat-y
right;
```

背景的基本用法示例

```
div
{
 background-image:url('img_tree.png');
 background-repeat:repeat;
}
```

> **重要**
>
> - 同时设置背景图和渐变色时的优先级关系：渐变色（background-repeat）> 背景图（background-image）；
> - 【v-alipay-10.2.0】渐变背景支持多种写法，如：`background: linear-gradient(#FF6010, 50%, #FFD2B3, #FFF2E9, #FFFFFF);`。

阴影

用于设置元素的阴影。

属性	值类型	默认值	可选值	写法	备注
box-shadow	长度单位&色彩单位	无	支持格式 `${x}${y}${size}${color}`，x，y，size 满足长度单位，color 满足色彩单位。	box-shadow: 10px 20px 10px red;	四个值必需

> **重要**
>
> - 目前蚂蚁动态卡片内置组件在 iOS/Android 平台均支持该样式；
> - 每个元素只支持设置一个阴影效果，不支持多个阴影同时作用于一个元素。

阴影的基本使用示例

```
div
{
 width:300px;
 height:100px;
 background-color:yellow;
 box-shadow: 10px 10px 5px #888888;
}
```

透明度

属性	值类型	默认值	可选值	写法
opacity	float	1	0-1 的浮点数	opacity:0.5;

5.3.2.2. filter

CSS 属性将模糊或颜色偏移等图形效果应用于元素。滤镜（filter）通常用于调整图像，背景和边框的渲染。

支持的函数

函数	含义	默认值	取值范围
grayscale()	图像灰度。 值为 100% 表示完全转为灰度图像。	0	0%~100%
opacity()	图像的透明程度。	1	0%~100%
invert()	反转图像。 值为 100% 表示完全反转。	0	0%~100%
sepia()	将图像转换为深褐色。 值为 100% 表示完全为深褐色。	0	0%~100%
saturate()	图像饱和度。 值为 0% 表示完全不饱和；值为 100% 表示图像无变化；其他值是效果的线性乘数；超过 100% 表示有更高的饱和度。	1	0%~ + ∞
contrast()	图像的对比度。 值为 0% 表示图像会全黑；值为 100% 表示图像不变。值可以超过 100%，意味着会运用更低的对比。	1	0%~ + ∞

brightness()	将线性乘法器应用于图像，使其看起来或多或少地变得明亮。 值为 0% 表示将创建全黑图像；值为 100% 表示会使输入保持不变；其他值是效果的线性乘数；如果值大于 100% 则表示提供更明亮的结果。	1	0%~ + ∞

不支持的函数

不支持复合函数，不支持 blur、drop-shadow、hue-rotate、url。

示例

```
<template>
  <div class="root">
    <text>normal</text>
    <image
      class="image-normal"
      src="https://pic49.photophoto.cn/20181202/0021033888940147_b.jpg"
    ></image>

    <text>grayscale:100%</text>
    <image
      class="image-gray"
      style="bg"
      src="https://pic49.photophoto.cn/20181202/0021033888940147_b.jpg"
    ></image>
  </div>
</template>
<script>
export default {
  data: {},
  methods: {},
};
</script>
<style>
.root {
  display: flex;
  align-items: center;
  justify-content: center;
}
.image-normal {
  flex-direction: column;
  flex-shrink: 0;
  align-content: auto;
  width: 350rpx;
  height: 350rpx;
}
.image-gray {
  flex-direction: column;
  flex-shrink: 0;
  align-content: auto;
  width: 350rpx;
  height: 350rpx;
  filter: grayscale(100%);
}
</style>
```

单击此处 detailImageFilter.zip 获取完整示例代码。

5.3.2.3. 盒子模型

蚂蚁动态卡片中的盒模型基于 CSS 盒模型，将所有元素表示为一个个矩形的盒子（box），其他样式决定这些盒子的大小、位置以及属性（例如颜色、背景、边框……）。

盒模型描述了一个元素所占的控件，每个盒子有四个边界：外边距边界（margin edge）、边框边界（border edge）、内边距边界（padding edge）、内容边界（content edge）。

外边距

外边距指元素和元素之间的空白距离，由 margin 属性控制，简写与非简写方式都支持。

属性	描述	值类型	默认值	可选值	写法	备注
margin	外边框距离	长度单位	0	auto; 单值长度单位或百分比	margin: 10px 10px 10px 10px;	四个属性值依次对应上、右、下、左边框的距离
					margin: 10px 10px 10px;	三个属性值依次对应上、左右、下边框的距离
					margin: 10px 10px;	依次对应上下、左右边框的距离
					margin: 10px;	四边外边框距离相同
margin-left		长度单位	0	auto; 单值长度单位或百分比	margin-left: 10px;	-

margin-right		长度单位	0	auto; 单值长度单位或百分比	margin-right: 10px;	-
margin-top		长度单位	0	auto; 单值长度单位或百分比	margin-top: 10px;	-
margin-bottom		长度单位	0	auto; 单值长度单位或百分比	margin-bottom: 10px;	-

内边距

内边距指内容和边框的距离，由 padding 属性控制，简写与非简写方式都支持。

属性	描述	值类型	默认值	可选值	写法	备注
padding	内边距	长度单位	0	auto; 单值长度单位或百分比	padding: 10px 10px 10px 10px;	四个属性值依次对应上、右、下、左边的内边距
					padding: 10px 10px 10px;	三个属性值依次对应上、左右、下边的内边距
					padding: 10px 10px;	两个属性值依次对应上下、左右的内边距
					padding: 10px;	四边的内边距相同
padding-left		长度单位	0	auto; 单值长度单位或百分比	padding-left: 10px;	-
padding-right		长度单位	0	auto; 单值长度单位或百分比	padding-right: 10px;	-
padding-top		长度单位	0	auto; 单值长度单位或百分比	padding-top: 10px;	-
padding-bottom		长度单位	0	auto; 单值长度单位或百分比	padding-bottom: 10px;	-

内容边距

内容边距指宽或高减去边框边界和内边距后的距离，即：内容边距＝宽/高-边框边界-内边距。

宽高

卡片的 box-sizing 属性仅支持 border-box，意味着宽高设定的是边框区域的宽和高。

属性	描述	值类型	默认值	可选值	写法
width	元素宽度	长度单位	0	auto; 单值长度单位或百分比	width: 100px;
min-width	最小宽度限定	长度单位	-	-	min-width: 50px;
max-width	最大宽度限定	长度单位	-	-	max-width: 200px;
height	元素高度	长度单位	0	auto; 单值长度单位或百分比	height: 100px;
min-height	最小高度限定	长度单位	-	-	min-height: 50px;
max-height	最大高度限定	长度单位	-	-	max-height: 200px;

边框

指定边框的样式，包括宽度、颜色、样式、圆角。支持 border、border-left、border-top、border-bottom、border-right 的简写方式，也支持 border-style、border-width、border-color、border-radius 的简写方式。

属性	描述	值类型	默认值	可选值	写法	备注
border	边框	string	none	-	border: 1px solid #f32600;	宽度、线样式、颜色，三个元素位置不限
border-left	左边框	string	none	-	border-left: 1px solid #f32600;	
border-right	右边框	string	none	-	border-right: 1px solid #f32600;	
border-top	上边框	string	none	-	border-top: 1px solid #f32600;	
border-bottom	下边框	string	none	-	border-bottom: 1px solid #f32600;	

border-style	边框样式	string	none	dotted solid dashed none	border-style: solid dotted dashed solid	四个属性值依次对应上、右、下、左边框的样式
					border-style: solid dotted solid	三个属性值依次对应上、左右、下边框的样式
					border-style: solid dashed	两个属性值依次对应上下、左右边框的样式
					border-style: solid	四边边框样式相同
border-left-style		string	none		border-left-style: solid	-
border-top-style		string	none		border-top-style: solid	-
border-right-style		string	none		border-right-style: solid	-
border-bottom-style		string	none		border-bottom-style: solid	-
border-width	边框宽度	长度单位	3px	-	border-width: 1px 1px 1px 1px	依次对应上、右、下、左边框的宽度
					border-width: 1px 1px 1px	依次对应上、左右、下边框的宽度
					border-width: 1px 1px	依次对应上下、左右边框的宽度
					border-width: 1px	四边的边框宽度
border-left-width		长度单位	3px	-	border-left-width: 1px	-
border-right-width		长度单位	3px	-	border-right-width: 1px	-

border-top-width		长度单位	3px	-	border-top-width: 1px	-
border-bottom-width		长度单位	3px	-	border-bottom-width: 1px	-
border-color	边框颜色	色彩单位	0x000000	-	border-color: red #333 rgb(255, 255, 0) green	依次对应上、右、下、左边框的颜色
					border-color: red #333 rgb(255, 255, 0)	依次对应上、左右、下边框的颜色
					border-color: red #333	依次对应上下、左右边框的颜色
					border-color: red	四个边框的颜色
border-left-color		色彩单位	0x000000	-	border-left-color: red;	-
border-right-color		色彩单位	0x000000	-	border-right-color: red;	-
border-top-color		色彩单位	0x000000	-	border-top-color: red;	-
border-bottom-color		色彩单位	0x000000	-	border-bottom-color: red;	-
border-radius	边框圆角半径	长度单位	0	-	border-radius: 10px 10px 10px 10px;	详见表格下方的 border-radius 取值说明。
					border-radius: 10px 10px 10px;	
					border-radius: 10px 10px;	
					border-radius: 10px;	

border-top-left-radius		长度单位	0	-	border-top-left-radius: 10px;	-
border-top-right-radius		长度单位	0	-	border-top-right-radius: 10px;	-
border-bottom-left-radius		长度单位	0	-	border-bottom-left-radius: 10px;	-
border-bottom-right-radius		长度单位	0	-	border-bottom-right-radius: 10px;	-

border-radius 的取值说明如下：

- 单值：表示边框四角的圆角半径。
- 双值：表示边框两个角的圆角半径。第 1 个值表示 topLeft/bottomRight；第 2 个值表示 topRight/bottomLeft。
- 三值：表示边框三个角的圆角半径。第 1 个值表示 topLeft；第 2 个值表示 topRight/bottomLeft；第 3 个值表示 bottomRight。
- 四值：表示边框四个角的圆角半径。第 1 个值表示 topLeft；第 2 个值表示 topRight；第 3 个值表示 bottomRight；第 4 个值表示 bottomLeft。

边框的基本用法

下面是边框的使用示例。

```
div {
    border-style:solid;
    border-color:#ff0000;
    border-width:10px;
    border-radius:5px;
}
```

说明

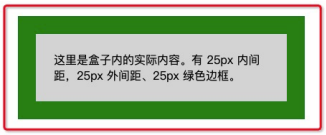
- 当使用属性的简写方式时，简写中没写的样式，将按照默认值处理，例如 `border: 5px red;` 无效果，因为border-style 默认为 none，而 `border: 5px solid;` 会显示 5px 的黑色实线边框，因为 border-color 默认黑色。
- 当属性的简写与非简写同时存在，遵循后者覆盖前者的原则，示例如下：

```
// 显示 5px 黑色实线边框 (black 覆盖 red)
border:5px red solid;
border-color:black;

// 显示无边框（none 覆盖 dotted）
border-style:dotted;
border:5px red
```

盒模型的基本用法

下面是盒模型的一个使用示例。

```
div {
    background-color: lightgrey;
    width: 300px;
    border: 25px solid green;
    padding: 25px;
    margin: 25px;
}
```



示例代码

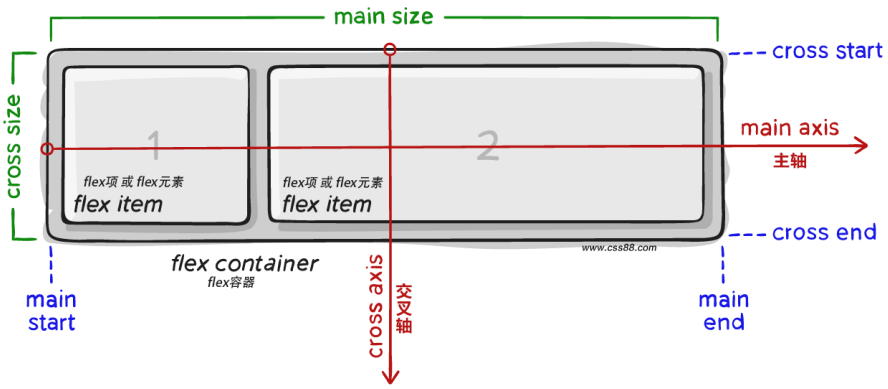
单击此处 detailBoxModel.zip 获取完整示例代码。

5.3.2.4. 布局

本文对蚂蚁动态卡片中的布局样式进行说明。

Flexbox

卡片的布局模型基于 CSS Flexbox，是一种一维的布局模型，使所有页面元素的排版能够一致可预测，同时布局能适应各种设备或者屏幕尺寸。



Flex 容器

卡片中，Flexbox 是唯一的布局模型，需设置 display:flex。如果一个卡片元素可以嵌套容纳其他元素，那么它就成为 Flex 容器。

属性	描述	值类型	默认值	可选值	写法	备注
display	flex 布局	string	flex	flex	display:flex	需要显示指定节点为 flex 布局。

与 Web 的差异在于 text 节点计算。即在卡片中，text 节点的 size 不会大于其父节点的 size。

属性	描述	值类型	默认值	可选值	写法	备注

flex-wrap	决定了 flex 成员项在一行还是多行分布。	string	nowrap	wrap、nowrap	flex-wrap:wrap;	
flex-direction	定义 flex 成员项的排列方向。	string	column	column、row、row-reverse、column-reverse	flex-direction:row;	
align-items	定义 flex 成员项在纵轴方向上如何排列以处理空白部分。	string	stretch	stretch、center、flex-start、flex-end、baseline	align-items:center;	
align-content	定义 flex 成员项如何沿着纵轴在内容项之间和周围分配空间。	string	flex-start（web: stretch）	auto、stretch、center、flex-start、flex-end	align-content:center;	flex-wrap: 为 nowrap 时无效。
justify-content	定义 flex 成员项在主轴方向上如何排列以处理空白部分。	string	flex-stat	flex-start、flex-end、center、space-between、space-around	justify-content:center;	

Flex 成员

与 Web 的差异在于：

- text 节点计算：在没有设置 width、height 的情况下，卡片中 text 节点在 yoga 的计算中作为外部计算 size 的节点，并且计算的结果会遵循 flex 约束再次调整；而 Web 中 text 节点会根据内容计算 size，并且不受 flex 条件的约束。
- flex-basis 差异：在没有设置 width、height 的情况下，卡片中设置 flex-basis 后，会根据 flex-basis 的值为初始值来参与 flex 样式的计算，不会再考虑内容实际 size；而 Web 中则不会考虑 flex 约束，而使用内容实际 size 进行布局。

属性	描述	值类型	默认值	可选值	写法	备注
flex	定义了 flex 成员项可以占用容器中剩余空间的大小。	长度单位或百分比	0		flex:1; flex:1 1 30px;	支持 flex: <flex-grow> | <flex-shrink> | <'flex-basis> 的简写。
flex-grow	定义了 flex 成员项在有可用剩余空间时拉伸比例。	长度单位	0		flex-grow: 1;	
flex-shrink	定义了 flex 成员项的收缩的能力。	长度单位	0（web：1）		flex-shrink: 1;	

flex-basis	定义了在分配剩余空间之前 flex 成员项默认的大小。	长度单位或百分比	auto	auto、像素值、百分比	flex-basis: auto; flex-basis: 50px; flex-basis: 30%;	
align-self	允许某个单独的 flex 成员项覆盖默认的对齐方式。	string	auto	auto、center、stretch、flex-start、flex-end	align-self:flex-start;	

定位

支持定位（position）。position 属性规定元素的定位类型后，可通过 top、bottom、left、right 四个属性设置坐标。

属性	描述	值类型	默认值	可选值	写法
position	定位类型	string	relative	relative、absolute、fixed	position: fixed;
top	距离上方的偏移量	长度单位或百分比	0		top: 10px;
bottom	距离下方的偏移量	长度单位或百分比	0		bottom: 10px;
left	距离左方的偏移量	长度单位或百分比	0		left: 10px;
right	距离右方的偏移量	长度单位或百分比	0		right: 10px;

其他

属性	描述	值类型	默认值	可选值	写法
overflow	控制内容溢出元素框时内容是否被裁剪	string	visible	visible，hidden	overflow:hidden;
visibility	指定一个元素是否是可见的	string	visible	visible，hidden	visibility:hidden;

Flexbox 的基本用法

下面是 Flexbox 的一个基本用法示例。

> **重要**
> - 当使用属性的简写方式时，简写中没写的样式，将按照默认值处理。
> - 当属性的简写与非简写同时存在，遵循后者覆盖前者的原则。

```
.flex-container {
    display: flex;
    width: 400px;
    height: 250px;
    background-color: lightgrey;
    flex-direciton: row;
}

.flex-item {
    background-color: cornflowerblue;
    width: 100px;
    height: 100px;
    margin: 10px;
}


<div class="flex-container">
  <text class="flex-item">flex item 1</text>
  <text class="flex-item">flex item 2</text>
  <text class="flex-item">flex item 3</text>
</div>
```

示例代码

单击此处 detailFlex.zip 获取完整示例代码。

5.3.2.5. hover

蚂蚁动态卡片支持配置 hover 属性，通过 hover 属性配置节点样式。当手势点击具有 hover 配置的节点时，该节点显示 hover 属性配置的样式，当手势离开该节点时恢复原始样式。

样式

目前支持以下两个 hover 样式：

属性	值类型	默认值	写法	备注
background-color	色彩单位	0	background-color:red;	hover 发生时的背景色变化。
color	色彩单位	-	color:rgb(255,0,255);	hover 发生时的字体颜色变化。

示例

```
<text
  class="normal-text"
  value="06.hover + touchcancel"
  :hover="hoverDic"
></text>

data: {
    hoverDic: {
      backgroundColor: "#7b8b6f",
      color: "white"
    }
 }
```

注意事项

使用 hover 属性时，需注意以下几点：

- 不支持手势移动检测 hover 变化，即 hover 仅支持点击状态变化。
- 不支持向上透传，即叶子节点优先响应 hover 事件，并且同时最多一个节点响应 hover 事件。

  例如，A 和 B 两个节点都有 hover 属性，并且 B 是 A 的子节点，则当 B 触发 hover 事件时，A 不会再触发 hover 事件。
- 遵循 touch 事件拦截规则，即子节点如果响应 hover 或 touch 事件，则父节点不再响应 hover 或 touch 事件。

  例如，A 和 B 两个节点，B 是 A 的子节点，如果 B 响应 hover 事件或 touch 事件，则 A 不再响应 hover 或 touch 事件。
- 平台差异。受现有手势能力的平台差异影响，具有 hover 属性的节点在动画过程中（位移动画）的响应在不同平台表现不同。Android 平台支持位移动画轨迹过程中的 hover 响应；iOS 平台不支持位移动画轨迹过程中的 hover 响应。

示例代码

单击此处 detailHover.zip 获取完整示例代码。

5.3.2.6. 动画

蚂蚁动态卡片支持动画属性，包括尺寸大小、旋转、平移和颜色等，可以逐渐从一个值变化到另一个值。

Transition

属性值	值类型	默认值	可选值	写法
transition-property	string	空	background-color，opacity，transform，all	transition-property: all; transition-property: background-color, opacity; transition-property: background-color, opacity, transform;
transition-duration	number	0		transition-duration: 200;
transition-delay	number	0		transition-delay: 200;

transition-timing-function	string	ease	ease, ease-in, ease-out, ease-in-out, linear, cubic-bezier(x1,y1,x2,y2)	transition-timing-function: ease-in; transition-timing-function: cubic-bezier(0.3, 0.3, 0.9, 0.9);

Transition 用法示例

```
.panel {
    margin: 10px;
    top:10px;
    align-items: center;
    justify-content: center;
    transition-property: background-color;
    transition-duration: 0.3s;
    transition-delay: 0s;
    transition-timing-function: cubic-bezier(0.25, 0.1, 0.25, 1.0);
  }
```

Transform

属性	值类型	默认值	可选值	备注

transform	string		translateX({<length/percentage>}) |translateY({<length/percentage>}) |translateZ({<length>}) |translate({<length/percentage>} {<length/percentage>}) |translate3D({<length>}, {<length>},{<length>}) {<length/percentage>}) |scaleX(<number>) |scaleY(<number>) |scale(<number>) |rotate(<angle/degree>) |rotateX(<angle/degree>) |rotateY(<angle/degree>) |rotateZ(<angle/degree>) |rotate3D(<angle/degree>, <number>, <number>, <number>) |transform-origin (center) |matrix(n,n,n,n,n,n)	translateX({<length/percentage>})：X 轴方向平移，支持长度单位或百分比。 translateY({<length/percentage>})：Y 轴方向平移，支持长度单位或百分比。 translate({<length/percentage>} {<length/percentage>})：X 轴和 Y 轴方向同时平移，translateX + translateY 简写。 scaleX(<number>)：X 轴方向缩放，值为数值，表示缩放比例，不支持百分比。 scaleY(<number>)：Y 轴方向缩放，值为数值，表示缩放比例，不支持百分比。 scale(<number>)：X 轴和 Y 轴方向同时缩放，scaleX + scaleY 简写。 rotate(<degree>)：将元素围绕一个定点（由 transform-origin 属性指定）旋转而不变形的转换。指定的角度定义了旋转的量度。若角度为正，则顺时针方向旋转，否则逆时针方向旋转。 transform-origin：设置一个元素变形的原点，只支持 center 。 matrix：2D 转换矩阵 translateZ/rotateZ 仅在 transform-style 值为 preserve-3d 时生效，translateZ 不支持 percent 写法。
transform-origin	string	center	left、right、top、bottom、center、数值（支持单值和双值两种写法）	
transform-style	string	flat	preserve-3d, flat	
perspective	length	none	none | <length>	须为正值，负值或 0 与none 效果一致。
perspective-origin	string	center	left、right、top、bottom、center、数值（支持单值和双值两种写法）	

Transform 用法示例

```
.transform {
    align-items: center;
    transform: translate(150px, 200px) rotate(20deg);
    transform-origin: 0 -250px;
    border-color:red;
    border-width:2px;
  }
```

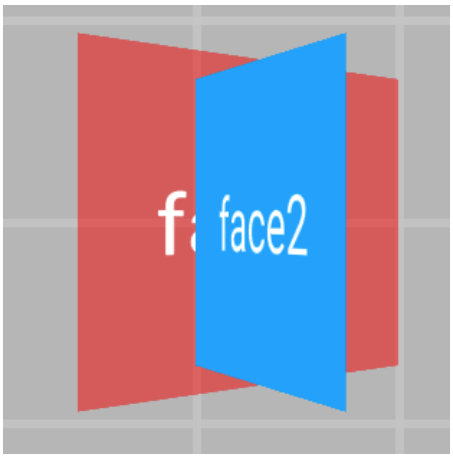
3D 动画示例

完整示例如下：

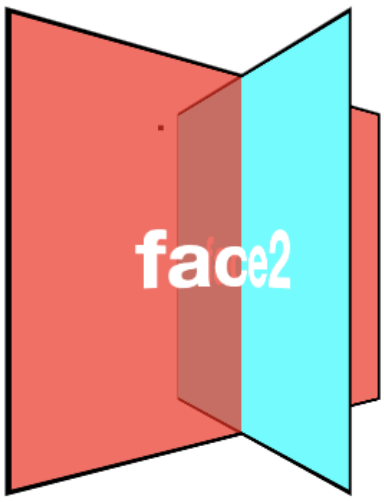
```
.div {
    width: 300px;
    height: 300px;
    transform-style: preserve-3d;
    transform: rotateX(45deg) rotateZ(30deg) translateZ(-50px);
    perspective: 600px;
}
```

3D 动画约束

- **动画嵌套限制 2 层（即父节点和子节点同时有动画），若父节点、子节点、孙子节点同时有 3D 动画，则最终效果可能会受限。**
- **针对 Android 客户端效果，受平台限制，View 不能分割，View 只能显示全部或被遮盖全部。如下图所示：**



Android 效果， 一个 face2 完全压盖 face1



CSS、iOS 可以达到的效果

Animation

属性	值类型	默认值	可选值	写法

animation-name	string			animation-name: demo;
animation-duration	number	0		animation-duration: 100;
animation-delay	number	0		animation-delay: 200;
animation-timing-function	string	ease	ease, ease-in, ease-out, ease-in-out, linear, cubic-bezier(x1,y1,x2,y2)	animation-timing-function: ease-in;
				animation-timing-function: cubic-bezier(0.3, 0.3, 0.9, 0.9);
animation-iteration-count	number		数值 ifinite（等价于 9999）	animation-iteration-count: infinite;
				animation-iteration-count: 10;
animation-direction	enum	normal	normal, alternate	animation-direction: alternate;
animation-fill-mode	enum	forwards	forwards, backwards, both, none	animation-fill-mode: backwards;

Animation 用法示例

```
.moving-node01 {
  width: 200rpx;
  height: 100rpx;
  background-color: red;
  margin-top: 50rpx;
  animation-name: moving-horizontal;
  animation-duration: 5000ms;
  animation-delay: 2000ms;
  animation-timing-function: ease;
  animation-iteration-count: infinite;
  animation-direction: normal;
  animation-fill-mode: forwards;
}
```

KeyFrame 动画

```
<template>
  <div class="root">
    <div class="line">
      <div class="subline"></div>
    </div>
  </div>
</template>

<script>
          const animation = requireModule("animation"); //获取module
    const keyframes = {
        'moving-horizontal': {
            "transform": [
                {
                    "p":0,
                    "v":"translateX(-200px)"
                },
                {
                    "p":0.5,
                    "v":"translateX(-100px)"
                },
                {
                    "p": 1.0,
                    "v": "translateX(0px)"
                }
            ]
        }
    };
    animation.loadKeyframes(keyframes); //加载module
</script>


<style>
    .root {
        display: flex;
        align-items: center;
        justify-content: center;
    }

    .line{
        width:200px;
        height:10px;
        overflow: hidden;
        background-color:gray;
    }
    .subline{
        transform:translate(-200px,0px);
        width:200px;
        height:10px;
        background-color:red;

        animation-name: moving-horizontal;
        animation-duration: 2000ms;
        animation-delay: 000ms;
        animation-timing-function: linear;
    }
</style>
```

动画结束回调

节点定义事件 @on-animationEnd， 动画结束后会回调相应方法传入参数 `{"status":"finish/interrupt"}` 。finish表示动画正常执行结束，cancel 表示中断或取消。

代码示例：

```
<template>
  <div class="root">
    <div class="anim_node" @on-animationEnd="onAnimationEnd()"></div>
  </div>
</template>

 <script>
   ...
  methods: {
    onAnimationEnd(param){
      if(param.status == "finish") {
         console.info("动画执行完成");
      } else if (param.status == "interrupt") {
         console.info("动画中断或取消");
      }
    },
  },
};
</script>

<style>
  ...
</style>
```

注意事项

使用动画属性时，需注意以下几点：

- 根节点不支持动画。
- 实体组件和外接组件不支持动画（input、slider 等）。
- 不支持 skew 动画，效果实现可使用 matrix 代替。
- KeyFrame 动画过程中，提交 CSS 动画无效。
- iOS 平台在移动过程中不支持手势，结束后才能响应。

示例代码

单击此处 detailTransitionAnimation.zip 获取完整示例代码。

5.3.2.7. 无障碍

蚂蚁动态卡片提供无障碍模式。

无障碍模式支持的属性如下：

属性	描述	值类型	可选值

属性	描述	值类型	可选值
role		string	input list slider switch header button img link search
aria-label	用来描述当前元素加上的标签	string	
aria-hidden	用来隐藏元素使其不被识别	string	
aria-disabled	用来控制元素是否激活状态	string	

使用示例：

```
<div class="scroll">
    <text class="case_title">这是一个例子</text>
    <image class="image" :src="src" aria-label="这个是一张图片"></image>
    <div class="div" :aria-label="这是一个 div" ></div>
    <div class="row" v-for="row in rows">
      <text class="rowtext"  >{{row}}</text>
    </div>
    <text class="case_title" :role="role">这是一个例子，测试 role</text>
    <text class="case_title" :aria-disabled="true" :role="role">这是一个例子，测试 aria-
disabled</text>
    <text class="div" :aria-hidden="true" value="测试 aria-hidden"></text>
</div>
```

示例代码

单击此处 detailBarrierFree.zip 获取完整示例代码。

5.4. JS 能力

5.4.1. JS 能力

本文介绍了在蚂蚁动态卡片中使用 JS 能力的方法。

如果卡片需要 JS 能力，只需要在模板 <script> 段内编写 JS 代码。

- 在 vue 文件中按照如下格式，添加 script 段的代码。

```
<template>
    <div class="root">
        <text class="message" :value="message" @click="onClick()"></text>
    </div>
</template>

<script>
    export default {
        data: {
            message: 'Hello Cube 1'
        },
        beforeCreate() {
            this.message = 'Hello Cube 2'
        },
        didAppear() {

        },
        methods: {
                        // methods 内部是自定义 JS 的方法，外部是卡片生命周期的方法
            onClick() {
                console.info('invoke on-click event');
            }
        }
    }
</script>

<style>
    .root {
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        background-color: white;
        width: 100%;
        height: 400rpx;
    }
    .message {
        color: black;
        font-size: 50rpx;
    }
</style>
```

- 日志打印

  目前支持 `console.info` 、 `console.warn` 、 `console.error` 来打印日志。用法请参考以上示例。

示例代码

单击此处 detailJSCapacity.zip 获取完整示例代码。

5.4.2. 生命周期

卡片内提供了各种各样的 JS 生命周期函数。在模板里实现如下的生命周期函数，则能在对应的时机调用该方法。

beforeCreate

beforeCreate 是在模版节点创建之前、JS 环境初始化的过程中所提供的一个生命周期函数。其主要作用是对服务端下发的数据进行数据预处理。

beforeCreate 生命周期方法的使用有以下限制：

- 不能调用 JS API。
- 不能使用异步的方法，例如： `setTimeout setInterval` 。

- `beforeCreate` 方法无参数、无返回值。
- 如果服务端下发字段不存在，在使用该字段时需要 `try catch`，否则发生异常时会导致模板渲染不成功。

onCreated

该方法是在模板节点创建完成之后再调用的接口。对于 JS 的写法无限制，一个模板只会调用一次。另外不要在该方法内修改 DOM 节点，可能会引起并发问题。

didAppear

该方法是在模板视图进入屏幕时调用。如果多次进入屏幕会调用多次。

didDisappear

该方法是在模板视图离开屏幕时调用。如果多次离开屏幕会调用多次。

onBackground

该方法是在应用进入后台时调用。如果多次进入后台会调用多次。

onForeground

该方法是在应用进入前台时调用。如果多次进入前台会调用多次。

onUpdated

该方法是在卡片更新时调用。其参数为变更的字段。

onDestroyed

该方法是在卡片销毁时调用。

onDestroyed 方法不具有调用 JS API、组件方法和定时器等有异步逻辑的能力。如果有需要，它能用来打印一个日志 console.info 或者取消定时器等业务逻辑。

注意事项

以上关于生命周期的方法，只支持如下写法，不支持箭头函数。

```
onCreated() {
        console.info();
}
```

示例代码

单击此处 detailLifeCycle.zip 获取完整示例代码。

5.4.3. 定时器

卡片拥有延迟执行和定时执行的能力。下面分别对延迟执行和定时执行的方法进行说明。

setTimeout

该方法允许在一段时间后，调用一个函数或者执行一个代码段。

该用例表示在 1 秒后执行箭头函数。

```
setTimeout(() => {
  console.info("setTimeout");
}, 1000);
```

setInterval

该方法允许间隔相同的时间，重复地调用一个函数或者执行一个代码段。

该用例表示每隔 1 秒都会执行箭头函数。

```
setInterval(() => {
  console.info("setTimeout");
}, 1000);
```

清除定时器

对于 setTimeout，如果在函数触发之前需要取消，则需要手动调用 clearTimeout 来取消该定时器。如果在函数执行完毕后触发，则无需手动清除。

对于 setInterval，如果取消则必须调用 clearInterval 来取消该定时器。否则会产生内存泄漏。

示例代码如下：

```
// setTimeout
var timer1 = setTimeout(() => {
  console.info("setTimeout");
}, 1000);

clearTimeout(timer1);

// setInterval
var timer2 = setInterval(() => {
  console.info("setInterval");
}, 1000);

clearInterval(timer2);
```

示例代码

单击此处 detailTimer.zip 获取完整示例代码。

5.4.4. JS API

5.4.4.1. dom

dom 模块用于对卡片的组件节点进行一部分特定操作。例如可以用它来查询特定的 ref 节点信息。

selectorQuery(queries, callback)

该方法可以查询节点的相关信息，包括所在区域、所在页面的位置等信息。

参数	类型	说明	备注
queries	array	要查询的节点信息数组。 ref：要查询的节点。 type：要查询的信息类型，支持 rect、scroll、viewport。 每个元素都是一个键值对。	只有可滚动组件可查询 scroll type。

参数	类型	说明	备注
callback	function(e)	执行完成后的回调。 e.result_key 为 result，value 为查询到的数据的数组。	不同查询 type 的 value key 不一样。具体如下： • rect • left • top • bottom • right • width • height • scroll • scrollLeft • scrollTop • viewport • width • height

示例代码

```
<template>
    <scroller class="root">
        <text class="message bgColor" ref="text" :value="data" @click="onClick()"></text>
    </scroller>
</template>

<script>
          //引入模块
    const dom = requireModule("dom");
    export default {
        data: {
            data : "点我刷新"
        },
        onCreated() {
            dom.selectorQuery([{ref:"text", type: "rect"}, {ref:"text", type: "viewport"}], (e)=>{
                var results = e.result;
                for (var i = 0; i < results.length; i++) {
                    if (i == 0) {
                        var r = results[0];
                        this.textRect = JSON.stringify(r);
                    }
                    if (i == 1) {
                        var r = results[1];
                        this.textViewPort = JSON.stringify(r);
                    }
                }
            });
        }
    }
</script>

<style>
    .root {
        display: flex;
        flex-direction: column;
        align-items: center;
        background-color: white;
        width: 100%;
        height: 500px;
    }

    .bgColor {
        background-color: gray;
    }

    .message {
        color: black;
        font-size: 50rpx;
    }
</style>
```

单击此处 detailDom.zip 获取完整示例代码。

5.4.4.2. animation

animation 模块是用于在子组件上执行动画的接口，支持对组件执行如位置、大小、旋转角度、背景颜色和不透明度等一系列的简单变换。

调用 loadKeyframes（keyframes）方法加载动画的关键帧。

参数	类型	说明
keyframes	object	关键帧参数。 name{string} 为动画名称。 values{list}_NativeStyleKeyFrameProperty 为数据信息，是一个字典，其格式为{‘p’:number;’v’:string}，其中 p 代表 percent，v 代表 value。

参照如下数据定义，keyframes 是个对象，定义了多个动画效果，外层的 key（例如：fluctuate）是动画名称，表示其中的一个动画效果。fluctuate 内是一个列表，表示会变化的动画属性，例如 transform 是其中一个动画属性，内部的“p”/“v”代表当前动画的进度和属性值。具体使用方法请参见 动画。

5.4.5. keyframe 动画

本文以实例介绍了实现一个 JS keyframe 动画需要进行的操作。

本文以实现线上支付效果页点击动画为例，介绍动画的实现过程。

1. 引入 Animation 模块，且必须在 export default 外面引入（第 19 行）。
2. 定义 keyframe 的具体动画效果（第 21 行）。
3. 加载动画。调用 animation 的 `loadkeyframe` 可夹在定义的动画内容（第 135 行）。
4. 给节点指定已加载的动画（第 159 行）。该示例是动态指定节点的 keyframe 动画，也可以将 animation 的相关属性直接写在节点的 class 中。
5. keyframe 动画中的 animation 属性支持请参考 动画。

示例代码如下：

```
<template>
    <div class="card_root">
        <div class="white_bg"  @click="clickAnimation()">
            <div class = "right_content_div">
                    <text value="点击悬浮水滴执行动画"></text>
                    <div class="image_tip">
                        <image class="image_tip_icon" :style="imageTipeAnimationStyle"
resize="cover"
src="
cGxheVAzAAAokWNgYFJJLCjIYWFgYMjNKykKcndSiIiMUmB/yMAOhLwMYgwKicnFBY4BAT5AJQwwGhV8u8bACKIv64LMOiU1tUm1
XYqbw1YuvRJsw1aMArpTU4mQg/QeIU5MLikoYGBhTgGzl8pICELsDyBYpAjoKyJ4DYqdD2BtA7CQI+whYTUiQM5B9A8hWSM5IBJr
PI1klCEk9HYkPtBQFul8zigpzESoUAYwKuJQOUpFaUgGjn/ILKosz0jBIFR2AopSp45iXr6SgYGRiaMzCAwhyi+nMgOCwZxc4gxJ
zDY7v////9uhJjXfgaGjUCdXDsRYhoWDAyC3AwMJ3YWJBYlgoWYgZgpLY2B4dNyBgbeSAYG4QtAPdHFacZGYHlGHicGBtZ7//9/V
YJ/MwPB3wv//vxf9//93MVDzHQaGA3kAFSFl7jXH0fsAAAA4ZVhJZk1NACoAAAAIAAGHaQAEAAAAAQAAABoAAAAAAAKgAgAEAAAA
AAFCgAwAEAAAAAQAAAFUAAAAA2sjhBQAAIoVJREFUeAHNfAt0XdV55r7ve/W4ekvWw5Yf2LzCw2A7hISAEwikmRmSZpXpyiSdaVn
aSrGRBJm5IUkNoASeTaSkhpU2Tpl2zZhZda2baEgYCxEDAIWCeBmOwsSXL1sOSLOlKurpX93Hm+/6z/6N9ryRjwAa2fe7e+9///+
+ffz3OOQuZ9Gmb33tNZGj+4yQsVi6G2Dc+kz/yjsfejqaH3m1HeGw81zBx98IZYLPsfE7WRFV7ZM/mZ4kB+PvKj+Y7L/7L93Otm3
v68AnHj05q2J2Oh/T3WmLjD1KWPCYR+rYsmYyVmTHcruni813ND08Tsef7+A+L4A8JVXvPiq0f+6LZHO/3GsJ500sZgxIcc0D3B5
Pm/n+iVw2E7ltbM1X7li/fn3+vQbSsfK9MWXkyTvX1Xlv3FXTFb/GNNXB62gSLgWQWQGw7Mf0xuMzZubo3EO50rovtV15w+vvjeV
8pgBOPbf/tVM3EXyR60z0mEfdBcy1SEGkrPVA9EeOimcuZuUOTQ3PZ+q+2XHXHve8ViK6575oN3sBAarb/hzcnmgo3RLvTYROJoG
m1NhDRcBCwA0IJJWrFgCgMT3uxo6M5C7DPfat+69V2fYKpNPu0gTv/6h2dHzKEfpVbFLzfpmkqvcz0uGjal6ZwJIQ7XJI0pFH3b1
KEEso2tjgpnpn92VLXb9p44rb3rptDfCqeBdBXDqse2fS6Snf5BYXd8RTBS0wLWCIEYjZuKVw2bwN/tlIu6+9GyTPrMbIGL88+xY
p3gna3LzJ9U2N5Wbqbmy68s/+HkoJ9WkPrumnrbLxp+5PJ8u/+tN4c/FL6LKYKNBlFTjX6yJh4wGkw7/cY6aPTZl1V15gSqWyOfT
ZpZYvpufxcY8BjigCMAyIh0u4tXbpoSkenzMwxc89E5PJta7Z+ZvK0NcoqPu0ATjz5wwuTsf4fJXsTl5g6dEUCJhcscMGLRU1ueM
uhFk2ptML0fOcuECBZm5VKhbPoff9kUJmbM6k9caBLtjcbMFyrHRQWyBC/NzJmZvuyzuXz3F9s+8Y3dpxPE0wigF848dsvvJ5tnd
65tNNGq9zgKo/ZbLFlzjL/aZwd0HTM8HN5gmdlcuVwgKA3nQrcdePWKGn9lvujefYZrOX20MvFMu7c7BuAi5fN7kDk1NzUzWbmu7
/RuV021MeTguAA7seaG4yv9pR01G6PrQCa7uQ70m+x9kqGQEULzePLvuSyWVyZs3HzjPxxlofPMEOPxZDwTsOL4UX9sFLa5tqTc/
ch0ARsAQ86XRBLRVMeypjMoPezudiWG7u2fu6U76dPOYBTj9+5JVZz9K9Sa5IbTSrhd1PWol1XfQBddu7ouOnH+Fa/stV0X3ImeF
9Kg4AUoooCgwzOP7NpnsoPjphfjZLKz2e/S0o3B44LIWXqaXXp2TzbX/cWOq7ftUvWnIj6FAHrhyce/+4c1LbO3x1bVNcjajl1Px
1lzpsmEz9uwBc2zPYdPz4bNMem2HnWG1SY7nOSRNqjdOHhg2QwCy44Je07xxrb+k4QRD4BVExrwpGDPRpWemMzXfbP/x7Xebfwxh
nQZv1jjRN7NzZGEs8/P3aTu96044uyEMAana9jnl4TzmbNwPwuuJ80fSiC0brucZz2sLGq1VsPIONFqVjETMP7+rHEJBIxqRLhxH
0I0ZmpQ0HEuOqNTJupkfDPMu2f/lrv+ZdNiM538KOmvm0VIw/ffX59w8DfpNbEt5ga3Y5BbQV4yAO87OFRM/Doy6ZhXadZsXmdXy
61wgXItErotVB6NuVuBgsGn95vpO8Nm1dbzTApLHpPHwruiS6OeEoTYpWdvJtOX3Z0pbviDlVd9eY9b1VtNq+lvVU74x3+547O1r
```

J1bXtlfOstYDycWlCMIoGnj89UHTfdk5pq63dcFLpBRmyLrXMUcBovtpOohtQvMUw5ia6T9mBp941bSe3W1aLz7D77q8QfRAAROx
Ja7ZvemRqpvUPuz+5/Z/FjLfx41j8VqS9yORjt9xY0zF3a6ynNooFG7ot5NXrZNxDHl2M27GBR16Ubr3yig+YCL3UnSiCammKi4i
PbnaVbKl0VgB9eXsQQcRheTsfsgTdGajCRzWOIEBnwEEjWz7iAcbF/Oj+ZSd/Yec2tP7R3URWeVPyWAeTZXc/UzT9Idxf+i2nBoa
VCXgsf+aNdt2b4Rc+Txvabp7B7TtnHNghcEpoH3ZCxwwVOPrAZReWgH0Bt57qCZhMev/Og5pmZlG0BElyYPu7CAaNMYFwuDGTM1m
20/vuPmtTi48BjnpMPgvu2tavbt+kl4b+gPTiMFfZ1mNiQZvPfJjv3ndHHvhELZfHzCNZ3YtBk8At1WLxxJJewX5qnIWk4exZ2PI
Ahfvlup5Wk2yuNwO4gZyBa7oxLjKwPAhUFDKR2rhJhrOX/+dlOxsylz/5yLP33QyUTy6cNIADuwZS6dQ//Cx9RvTfmjq7vlPgBAw
OTcr5gBh543szP5k3vNRtNohkLae4qgsCG+4YveC0KLUli8kreIQZ5LXRilgWCFiEsZ+LplGlcu8KM7Ok3M4dGTHpVmwlhrBQvpI
pi96yK5Sy4JPdTwzIZdD/c9enIgStWOniWTO3d60Qsj3/pJ4/rQF0yt63lgV/BgWH5k0hx++EXMsitM+8WcZdlNcAlgVI3q3Ma6t
slZ4CvMOQbov8iTJ5aRJcMtZN240Z+lZLNxXX3WBiTVgqcUuTdvkgsNxXMSNLo/OmJHh5O1dn9pxE4x9U0/0p0gavGzwQhd43/6z
eFwwH5MB7IKDAALyZ/YOm74HnTMfm9aYde9VgrFHPIEAK0nJpYbBMyhvYVU1A3iUxXX2CRSAtMF3YYzefu8oc+PluMzd03BhuAUX
O9CSuGcEutaWma3Tb485u+HFR9goRrwpJsxx7+zvVNK/M/jnbYBTIrkq5rK8YDoIkXD5nRPX1m1ZUXmmQbjqu0y4qBtgoXeLcmvQ
SJtvUtp4l3MqKf5Sd/rmLZ0ltN/6GHCiwigTcMLDz/+iln14bNN/WrsgPCgyuezXshlDsbM2YHM/Hi++7O9V3/7PkguG07ogYO//
6ZbZH0TbeXIM1orGAhh43vju/Wb05cNm7ac2m2RL/QJ4UqUFT2+1ZlnGNPVJrHknRtIPrhApVkYKHXmf2dcZKLVEYsiAxXU99s1r
BOfzEXpN5Y8jxROqy+nByVNtZG6/zjvz1wM4fozstH5YFcN8T/1RfGxu6J7GyLu2DByXQH4CIu3n8+YNmbN9RgLfJRGvRvdld1Fj
UzvSgsSVzgWk4+EEMiSENsKX6XJo5I7yqaGszOa9Bb+ne9ikcBo7JelYoJoPaweMw0dyc7ozPP/dX+/R4at3RYFsDW3K5t6d7Exf
RS73gfPm35t0AxjmbL2kxeZaEoXx4qeU5nKsTEaNK2x0peNq+onWotkFxEWtLFIi9nFCSIeoa654nzTh4Pa3GjGB1H48MOYQOI0q
+3hy3ze+sqCsMrUkgIMP7Liotnn+qwbrI1FGhRqwVMmPTpkBdIE1V11oYnVYTNPzJNjKNW2pFZHocvlQqvo1VgG5H/zRSwtOEKt+
GL0eGIGK5VdfeYLq2nGneePgFU8IzFd12+l0ZcgAx0lhjaqLHbxrc9ZdnORqC5CIA773Xi8Rig7cmu+tqRBHvhAZ4k4f1VT8ONDs
FSPFoPwAMTDWMIbFZZjf3iil8WWbEKOjMidgJZ4akqpy5XrqqYIkEAiM1rOkwak0nfTjzMk1MkCFCGPYcXtofNHcmG0rFXbv0dYB
sQiAD9Ud9Mn0u3haygoCqhNuyH2tsO/3ienxtyeBbOtalW+IF+FDA3zW6ccleBVsftMFpHqMs3b4gWFSOmNrCAiI/U7RMrmCmblR
DhPM2J5D/qRCFvLqeIiunK4pfPr79bdcAWpFqADQA8LJxPQ34u21RM1n1ErRdXNYP00cHDErL4U3y8ElLKARywVutxhUx3K8LNdI
fvSkaa5HqpAFL6Q3KVWCJmEBzmYNetBanREdf6jPzU7NVXRlyWB+mW1MRkzuybTs2Fa6mCgD7k1+/PN0SuUwmDnJVGT2I098VOPm
5ZX5HHVVeVXqpMvKOqxVXZKi02uxzTUpVAZDl2VR6UI4ED3RR2J83ru82Rp/YBMPY+y0gMeOHVk4Zkcev1he98WFUwrgAwGc/+Ub
qepAnLB+6YPDuMUuWSaN+BggAtlAqEXeU4UAmPBJHrx49Ioy3w1Teks0DKJNUOGpQL0q/1vxqri6Mpd5/Wa6ePTZnZoXMY+AY56e
G1pSkdLO8BdhjGpfAPDV//nV1am60DU8wwsaqWwwYgRrvg4+SuS4QKNYVuWhoFSGpYxfilYp5efId1K8yzAtQ15UldSDH/SoCEBq
XDD73ht9OYUZD1Qtx4JCI5H/r0M4/71U9gQfWxGf/dbqjBvswi5qNiHwW258S1k4Nq3GuJotlVCgVq5olYpYHCEjGYbLCb6ZDJKp
rxfkdlUBdpqrhKtiobSAs7fnJF047HDdlM1sxhqSbLGgEPnMQDXbu5MZ4uTRz4NyorAHpwyUSqfK2s+1hCIa0MU/vY3sOmCcdCMj
TRUZBZkVLsby9jnEpy0llGdXpoI8rYsMMyVd9JBUj2gysagvCpBG3DYGsHqo7G33YzuG5Chq6JRwCWOE/VIcfpaXdIIgC//5IaeR
c7BlU90QKM/mTBbHVAJg4H2oTButvBKTjgQvbehSfBU8wlylT2mi1PlRuo1Zh5OUtObdMsecgN/RGiihrThIaIMXTg0el7NN7ZDE
dWDRc2nR70514fcJOIjW1k1vqm5LY8zqexxuI8W4G3TeKY6wYX0XjzBsYWGHB8nSyqUw1mAHd0aU8WqbymndjTQcVWGbq0DLGqlC
zLo2VwkhrMyDx4nT2Gd5Pk3UUtRIwema6Lps3Mwc2kigfiROrSSH3VfpnKwTyFJ131XXjyT0C1YkpqWGSgtUroYLJZZV827+qplh
IWuhTZmpFdQCRMgWpYKckC0hS4P07zQ1voVTWZqAAcN3J3Y4LEATlVbEzNefvpDJKPUC8XiZqPsPCjtAoXlSnZsytStAIA8J2PQu
1ugX+eXk0byWI9YkE4EO5VsiVn65aVZGFbvyUm7lVYZZphcq9dPVvEG56icb0lgXpgHgDCcSRwYbWZGIwDtDXvFCs90Lh5/422/U
htbI8YZ0aAGQRD6A5+6Ya2H2doyrlESNthhUFeZvWvBjBjBIoY/NKro5FreW3kU9Cxs2rqgp+R45JvZRX5ZXuxjwXwS6L3biAZ8c
hyaGPQGN06bIprj/3OozXhlDfYEYmFsT4hEy5VDuY8tjURnPuFE3hdQnYeKJTKbEylkmfChqXyLNI7WQG0FLBw6Ut4WWaDKysi+N
MzSmA+CTTOqoAcMfkLKwITeFk/G5f3EAibRRc6FbhwOe23HjzzRFq6NRjqisQhcDMFVDqY81kMxPgdxxgG/JvsrdolVPsGVZ1rzy
W4+WSxl5SXcvZmyQ13qredxyTZPHphkF9Wlay8hHmtIRMzi0ENocgePkZ+as51kZ8sG5ouFQTSg/2xotmVJLIoHdB4U1SDpkChCC
HKNDyilcwqNS6Nxno4tIApVEhBZEnSZL8YdrSAsWW7pf6vFpWUYSM5jUWfZZOmoKouqTc/gQywmg5rKzLjzR7X3EO3/Ko7UHFOH+
A5m2mKxhLROm5hxCo2ipXbvl7EeVmIWzsxiJVQkwYaYPMEVcoWZJXLr9PhrSwIcn4isNTPUoxBYzdNmtDxo2naKRfLLJ2FLl3yVl
F/AQ6JfHsZYV5KvA5x2kw8hAoxCpflUlDtAKRY6fxYaIbyiGGQrSGEfN/ApLRBbkPXViGarEkxabG+QD4BotD9ixAIhyCKhaY2VI
4FJ7tIyabJEoDdLkZRl/bJCkzQtd0wvlys6YpaFQOBT1SuU5vgkvh1ykBheglZd1Cn5FpPtimFCQDIOwCAgyBSj5/JK1QIIiQS0R
Z8jBWPkm7BSyz5WqXxA4tKEdCWiwEK2f5qIRl8kSCaWb9OITYwzKOGAQTqDD4smX/HZt8uJDPZnhM5RsrGsDhM8XwsGieYwDIQbl
mWLkaQicE3QPitMb6sXyS6hE/ztk7hrUq7fEFaeRx5AhDU5dDBWmkH8hX2WV6lBcaxDkzG8/Mmgtk42IFRn/B62PEhE09Ph0te3W
6grDEAhli1xfHJawDsu8nGLla0wSCsWxdRNORu7NJEFIQCJPM4l5fyxwS2TtCvrpEUJZRya8guZdNVp00uVC4+Wy34DTS6bIs4I4
0Nu0DiaBTaFQmg9FI+PRbCF2rDg/nwGxSTZ2CiSEEtj/FnN542ExzWndN4a1Sb/0wZL5x+aVbm1eMmK3Zx0VwXZjJS8qt8xKVxuD
pK/I2EdxI5Cv4bN7VIWl/lyE6sIQroM0evjmJ8zMzPQMQ1eADNhj1JkOtPWPhycI5Y/O54rB4mVUkSuiBeGRJmTzeQ/bHO1ZOQxl
2NDPgv5fbGBbwgL5YlDYyBTgqTZvVVx2yMNIjlLp+mEVeAR7otYyKQdeRJD3jAi1VJbjqL6TWCtaDfhf2xkXwAEMCWyqHBqZovT4
ytfyWO3sr/iIRGNRkX8BoMgznJPKJ5jDdHKLJ/fWDV0CR4FITDe8iASy119yitFvsGiP5AljWJa5qsI8uxuFWUOn3ZFFzCrSycPk
06OTsoXFRAt91LlQZxlLm1I5+tqmTabIDmjm8+Xn+Xiv8u5Qwj+VyOB7DL93qjHVMRgrjHbz4KVHKAAak1/TGiuNeaFZPVounqW6
aCanfzWva8gX6kBc9VgZKKCq6lAegTeOzs3o9RLH28/sU2oU5w5TD8ecBiicA5ufCv5mbRDcVBVTMSvCDuaWhpw0f703gI0C8T8f
BCQtKMcUkFqE0awwLlteXayKAeKxPIunw2rbwaq24ULwaNMkvUuaSswxeU+x2tiEPV/NSMqetoEgwqu2/ZTE3nTTSZfhoa/PPAwd
1OTuePiqlTGi5bwVKKtAXZ6ONbC+yM6AQRjFtksvxpBrdJIxErTvKjFj+StXAVNZW2ZyJOGi0HlXBny6FXV1USuwgbwqqzQy1CpN
nsZlcJAaw/CVqMcrfXCioF7IeMWiyea9kbrVF+NVBgvgNV+7Z3hutvhs8IEKlYrbluVktqG7zYwfGOT+xSqDpBoWdAfQKMdjf5YF
gQ0PS70ZWhnm5yFNdBprYs1wMAQHBymoa2YDu0tQu0Eiu6M4kxCNm/NCQaeLrwPgnAENGuy+P9wql6NM9H/y945D2PZB9OTsX+X/
Z8buNZAe5A21krzeThEVPmOMmgRmgcGAgByjDPAVvLNa7gIy/5nFjT1XxKXzK29UGNBPK4l6tL6S4tSAMFTB55bBxmMWQ1r8bzbw
Z1wJrQpy3VxrO5+Ysb6ZAxkYjJXd//YCI5fqj0Ip7M1HY1yrDV+cMh/4EyUtOKgAmu40qmeaTG6CkxtiCsbyFnd1WWL8vQ8W4fGy
HM9kObJij7VdPUt0oEiU4bnvsQNHTR1eFE2ka2UrRznf+6ACgI5Pzk9H0qseFBH8BAA+fOzu/VOZwhMB6hUeZEznBevM0EsHsQai
iFH447gSGkw6a5jUmr9KFRqsoa3lJk8vqUp6ArryOHpAW63BsER2q18rb+hSQhRuAci6e0dvGXz9iVnxgrRzr++2wOtHuIk6o8+X
Oe+1Y/apcQALh9e6g8k43+XX4cL9e4Xgh5gtq4ptNEcLQ1+hqel/KISw0MQGBFTgMUIGmoS4c+6lS5YEJiI1mGH+rRtOoJ6rOyKh
w5XcEOtLupxdCx4HnkXrhD2/UN7++Q75Lq2Ztl9+baAB/bwYGFiEn8EI9X4UwOsoFVCACBzh0Mr/mV4ePaAwUwTNERBQfnKS/AG0

```
c/NlREpKI2xc0SBW6l7Ku2Cw31jyoMzllUZZGooXgHZkg7ogLKBRj5ZTxtEZ8HLSUD7y+LLclvEvg8zNzpnx1/D57cVnVox9OnmU
YzJqX6y/9QtB9oWWhCzNz7fU7pjOzkbuLU1nfC/XDPIKI9+fSq9pNuqvF9P8aX//gyVRFwwmEGGgbRZnAYDWcsZYjzfI3u1Sv61c
Pb5VG91o4K8EQHqwZ4NBtjX99Te00b3s5KNWHZhpmWdhFcsRfzwDS++SuH6+5aufJSLJgXQoUHkjweWvfTwaOzB/khnt9YEOXuQh
37kPJMZGjPH38CyBmslAUAMssBQSdAQ0OROW0OqG+/ml01bve7NJK/WIUBZG8Ea3FSh+16nXrQw5nH9R9uwXcVpyxCGpVJ2znReu
PkVXIihnnRjOSvj7C2OZ8t62i67+H2yeGxYBuPX3t08en4nflh+HF/IlSo6HYjyU4U5EcT629oqN5tCTe/B3DjBeBmtD2yjllQai
G8pZnKVC1XPhRp8ZK11jBYx7/A/0QEVCF7ssvTBaW15aJNyLNp42ZiYwZeeF1vGB5gYlw3cf2osyz4HEHxrGvXNN+61J/em8RgEQ
fj3x8dzv2KH6GIQrqyVcw9c0Nvh+k8f6157Re7gTHKuOGuBkqB1AaDTRqofBqLJ6DQdjXxWAcE0St5BwSV1Tjghx5JMwK/lkvs16
MZEmMOpysHHnjcrN59taloa8baBfx4gbRbHwcwLjzw+F3lw+LN3/iOxqQ5LAnjudefOZ0otN4wPzmRDuAM+iNYgGoZJpGvTWaYee
HnwGYwMA5BV4FhtLfhjtAqlp20i/gTAJbAuTiaYRQ0UlCNRZRVOgrE4CpCBVgu93Z9pF8OZh7/6Hn8GLRF2mdcMq4+E7Y9rqy9Lu
umjdDY/mMqVv79a2hkD0MQP1OWBJAlm/8/F88M5yJ3lbAhBJC13XvigCDu7Vm60YTQ5fe9+DTPojYK3PglfGFd10aBWMCDyAAuBR
vayaRdsMhvxzF62yLgpJy6YY9cfjqMZxzzGHZe/8XTpgEO0H3xOQAPqwrwy6Qhnoc0HGcaf4pl1jT8ydmf/u6yfxZgWQAJYih29Y
cIv8Aq7bOl8EGEIAcD4GMLaaP01W0wUk8nenz9limhIiG8zwQiCKI1yGy0eClmhAViN6akWDJ/mN3ahW2uesj4g5FsATW+S1W11C
P+rgmJcrl8xrDzxlGnBIsvLS843HJRnaIXXRDgII8ArouiPT0f8z2nnPXcRiuYB+d+Kw+97bV3VF9+3sXJla6yXwfAB3z/+TTMCe
7Rw93te/R58zU0TGz4arNpqa+Rv6gjn+KDR7WwpOcIPLTPkF+l/4BHv46I0gIvsJMYBgYyaV5xASX5QIey9Ft8dFQZnLGHHrsOSx
wXPI9DkT884aaI5xE8nDYDvIGx4t5Q66aPrb3yqyOsZrngtGQ5FmNeuvfGD3Wlxu5vWVHT6PE9GYAmz0gIIIHBQtSAPvTca2YQC+
zzHtK3FZhwHjzKmsBYLoADCH9I0uGniwLzFQ1kW8gTGUgUkpitpAXjoKSGuErDLGDkwYI69dMB0bznbtKzvNR4PBWTYUPCgA55YP
OV48lk2tu3rDp7a/ENS/TMI1fRkWn7z/f3/t2rbU5P9qaE0lPXyIJyDSOP2bCXzohPEwc+SYeGOqBX9ADEDG8UzB01McBbEaQFax
KFMuDtAOWAEdvIwOCAIoMgUEIA7gc1nADT++FV+XMqo9uxIeGmG0FPH+Ika4rQxLGeUwaYzgsyIS7Prvh2h0PiZI3+VnK7GVF9v/
+uIzn5t/XN8UQZLxXK3WWX1k9h6ZHwxCIW4Ud+vcdM9o9guXOGadvQgxdJ0D7+LRcGAhnUXJVeQMrn5a8CpAnJaze15QoeysKpmA
tj+w2Z8X79pPqMH37ecJe8CyM0Er7/LALOCh2fA41PF2cly2+fP/Mx/+78LlZ84FTTjxGwLpYfuu/F3W2ITP65vitd63M5x0iCI2
DLs0tkfTR0fN0adewQPqgunACUfzqhUYNjFTyyNDIIp0KADzRKagof5/a4jNBDOt31VDuHklTCDH+wbxGW4/nnvX4Js+/NUOeh0n
zCh6B42THyRA3/Ph0aWoy1Pl76//Vbf+80No3T53I6mWl+x78ziebzNDfpRsj7WV8nIwHzP6EAhADIAkMt3qgTfYPYbW/35QwvjS
JNqztNshZfeXLpgL8NKN6gtYlF+KFHUQdjDUwyb8tYF+vmk53cTNZMALjMwDETwyTWfv56fFyN1x65juXeFtihIn9MJvACXsmEcX
m2O5OJdn1/5ie8+plWdbCzmniyzy9f30C0bG73BnzbUmws8Hm8JiGiJeCLazpjeyJhjJopmhsfRpfrw1v9xw9dG6jpbsRhvMUkcX
9wUZ2Dh7cTIQA0WV1QVQATuGiSK2kvhzyCPjZnYYb9QDiFroa97Qa1LYVXA2lSFDAefN4vJGPA91gJ/LsPFs5DeF1Or/0LX1m/jG
Hse+ti/kS+x//SVtT/oU/b0zkPhdJRU0Z3TmYnauBpDcRaFwlHEzOjIyZ6YERMzc2KSe9UYAcw18EidfWCLhh5EUXqpIH2fAibqv
nzWbx6DADxXS4+uW0y6Z4OU9vejB0GbgKHB4AnNwE40VsD4KznhbGFm8uVzGwp9TeRlo9+vXnTdXjw/fbCOwJQqsSL1qOXbbu+Nj
6ou3O7RQzAGSuMJGjzPnzQ43lGCNFz0WF5YvBYwQ/IxIq95vBFQRF7ey+PCFkL0ZoIVxWencTwpSzTUyZF7NIl1KZUSMHiTHACIx
63mypKHXQVcI3TYEL5yZ845kTcMfd1z5PZyu+M82aNnbCdKktyNYLTPw1I71Tbmjt8TDud+N1UTRyQCkrBdRBRspF6RYI0FRBaQT
DwC9pLauOCYr1IhnDmK4ADALMs5OzTMsxUYSBUz5bKuW95D+UEt1/0vyRbYer1b+dfNCOtyO8WMYLTT1x8yfj5fFvJqPFD5sEdik
D5Sx1IBEDaqiXCj5u1igmFBDcRpFESgKdpdlcw8AJ4IXgeqyvlSzh5iezMeXV/2vqxHY/4Sk/NrzX71ChTLbs9L3bmk9/67YQ3/a
PCRCL5VBpKYCP3uGIBI1GiBWOGY4iQpVxHEw0AhXdJI8L8FjTS++0lPLs4Vy8Vy6NFCqPbOBy773n3XhU7NX6107XFNdemnJM0/G
+e0r4uXZf4/vKq6JJE2rHMDCO6T9dA8EWQtKIvhhYnGw4AlYLGUe//k2qWjiUIDuWsqXh0vl8P3zJvmz+p3ff8J9CLRY6TujnFYA
yL97RE50bvyrs5T+Fvwx9CTYv3aEkPJNBgGD3Q9qC6tNRRAsJkkAuVOuwKLBl5Tl5hnEYvrer5IXvK8STj9Rt+d6wz316f981AN1
69qyUlc+Q8rF+24JOpizxTOhNelAXwGjGPxGUno2OlCmI8466hXPbyKMLbTqGjAPw1eOOzWEQ+HU00vRzauB1fB7674T0BsLqJO3
F7R8IsmbEfbwuH5VrzA3QSPS4e9coIbiHAkPFcsedORcHkiGomPThsz9sDBD05cd911WL+8t+H/A+Q3uJ3ZZ3xhAAAAAElFTkSuQ
"/>
                        <text class="single_line award_number" :style="imageTipeAnimationStyle">{{a
nimationText}}</text>
                        <text class="plus_number" :style="plusNumberAnimationStyle">{{plusAward}}</
text>
                    </div>
                </div>
        </div>
    </div>

</template>

<script>
    // 引入 animation 模块
    const animation = requireModule("animation");
    // 定义 keyframe 动画
    const keyframes = {
        'fluctuate': {
            "transform": [
                {
                    "p":0,
                    "v":"translateY(0rpx)"
                },
                {
                    "p":0.5,
```

```
                    "v":"translateY(-10rpx)"
                },
                {
                    "p":1.0,
                    "v":"translateY(0rpx)"
                }
            ]
        },
        'integralClicked': {
            "transform": [
                {
                    "p":0,
                    "v":"scale(1.0)"
                },
                {
                    "p":0.5,
                    "v":"scale(1.2)"
                },
                {
                    "p":1.0,
                    "v":"scale(1.0)"
                }
            ],
            "opacity": [
                {
                    "p":0,
                    "v":"1.0"
                },
                {
                    "p":0.3,
                    "v":"1.0"
                },
                {
                    "p":0.6,
                    "v":"0.0"
                },
                {
                    "p":0.9,
                    "v":"0.2"
                },
                {
                    "p":1.0,
                    "v":"1.0"
                }
            ]
        },
        'plusUp': {
            "transform": [
                {
                    "p":0,
                    "v":"translateY(0rpx)"
                },
                {
                    "p":1.0,
                    "v":"translateY(-48rpx)"
                }
            ],
            "opacity": [
                {
                    "p":0,
                    "v":"0"
```

```
                    "v":"0"
                },
                {
                    "p":0.0873,
                    "v":"1.0"
                },
                {
                    "p":0.7205,
                    "v":"1.0"
                },
                {
                    "p":1.0,
                    "v":"0"
                }
            ]
        },
        'rightContentShow': {
            "transform": [
                {
                    "p":0,
                    "v":"scale(0)"
                },
                {
                    "p":0.5704,
                    "v":"scale(1.1)"
                },
                {
                    "p":1.0,
                    "v":"scale(1.0)"
                }
            ],
            "opacity": [
                {
                    "p":0,
                    "v":"1"
                },
                {
                    "p":1.0,
                    "v":"1"
                }
            ]
        }
    };

    // 加载 keyframe 动画
    animation.loadKeyframes(keyframes);
    export default {
        data: {
            animationText:"123",
            plusAward: "123",
            shouldAnim: true
        },

        onCreated: function() {
            this.imageTipeAnimationStyle = {
                animationName: "fluctuate",
                animationDuration: "2000ms",
                animationDelay: "000ms",
                animationTimingFunction: "linear",
                animationIterationCount: "infinite",
            };
```

```
        },

        didAppear() {

        },

        methods: {
            clickAnimation: function() {
                this.imageTipeAnimationStyle = {
                    animationName: "integralClicked",
                    animationDuration: "750ms",
                    animationDelay: "0ms",
                    animationTimingFunction: "ease-in-out",
                    animationIterationCount: "1",
                    animationFillMode: "backwards"
                };
                this.plusNumberAnimationStyle = {
                    opacity: 1.0,
                    animationName: "plusUp",
                    animationDuration: "350ms",
                    animationDelay: "0ms",
                    animationTimingFunction: "ease-in-out",
                    animationIterationCount: "1",
                    animationFillMode: "backwards"
                };
            }
        },
    }
</script>

<style>
    .card_root {
        display: flex;
        width: auto;
        flex-direction: row;
        padding-right: 24rpx;
        padding-left: 24rpx;
    }
    .white_bg {
        display: flex;
        flex-direction: row;
        justify-content: center;
        width: 100%;
        height: 144rpx;
        padding-right: 16rpx;
        padding-left: 16rpx;
        border-width: 0;
        border-radius: 16rpx;
        padding-top: 20px;
    }
    .left_div {
        display: flex;
        flex-direction: column;
        justify-content: center;
        flex:1;
    }
    .left_content {
        font-weight:600;
        font-size: 28rpx;
        color : #333333;
        width: auto;
```

```
}
.tips {
    margin-top:6rpx;
    display:flex;
    font-size: 24rpx;
    color : #99999999;
    width: auto;
}
 .right_award {
    font-size: 40rpx;
    color : #999999;
    display: flex;
    width: auto;
    flex-shrink:0;
}
.right_content_div{
    display:flex;
    flex-direction: row;
    flex-shrink: 1;
    justify-content: center;
}
.right_content_first{
    display:flex;
    flex-shrink: 0;
    flex-direction: row-reverse;
    position: absolute;
    right: 0rpx;
}
.right_div {
    padding-left:1rpx;
    display:flex;
    flex-direction: row;
    flex-shrink: 0;
    width: 272rpx;
    overflow: visible;
}
.image_tip{
    position: relative;
    justify-content:center;
    width:80rpx;
    height: 85rpx;
    margin-left: 20px;
}
.image_tip_wrapper {
    width:80rpx;
    height: 85rpx;
    position: absolute;
    top: 29rpx;
    right: 17rpx;
}
.image_tip_icon {
    width:80rpx;
    height: 85rpx;
}
.arrow{
    display:flex;
    flex-shrink: 0;
    align-self:center;
    color:#CCCCCC;
    font-size: 14pit;
```

```
        font-family: iconfont;
        margin-left: 6rpx;
        margin-right: -7rpx;
    }
    .single_line {
        overflow: hidden;
        flex-shrink: 1;
        text-overflow:ellipsis;
        lines: 1;
    }
    .content_icon{
        width: 36rpx;
        height: 38rpx;
        align-self:center;
    }
    .award{
        margin-left: 10rpx;
        font-size:30rpx;
        color:#999999;
    }
    .award_number{
        position: absolute;
        top: 0;
        right: 0;
        left: 0;
        bottom: 0;
        display: flex;
        justify-content: center;
        align-items: center;
        font-size:36rpx;
        color:#806A00;
        text-align: center;
        line-height: 85rpx;
    }
    .plus_number{
        position: absolute;
        top: 0;
        right: 0;
        left: 0;
        bottom: 0;
        display: flex;
        justify-content: center;
        align-items: center;
        font-size:36rpx;
        color:#806A00;
        text-align: center;
        line-height: 85rpx;
        opacity: 0.0;
    }
</style>
```

如有需要，可单击此处 detailKeyFrame.zip 获取完整示例代码。

6.常见问题

如何解决在安装 AntCubeTool 时遇到的 Permission Denied 报错?

如果在 macOS 中进行全局安装时遇到 Permission Denied 报错，可以尝试通过授权管理员权限解决。通过以下命令进行管理员权限的授权。

```
sudo mkdir -p /usr/local/{share/main,bin,lib/node_modules,include/node}
sudo chown -R $USER /usr/local/{share/main,bin,lib/node_modules,include/node}
```

如何解决在使用 AntCubeTool 时遇到的 Operation not permitted 报错?

如果在 macOS 中使用时遇到 **shell-init: error retrieving current directory: getcwd: cannot access parent directories: Operation not permitted** 错误，可以通过为终端添加文件访问权限解决。添加终端工具文件访问权限的操作路径是 **系统偏好设置 > 安全性与隐私 > 隐私选项 > 完全磁盘访问权限 >** 勾选 **终端**。