ALIBABA CLOUD

阿里云

组件开发

文档版本：20210207

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.概述

IoT Studio提供了组件开发功能，便于开发者开发、发布和管理自己研发的组件，并将其发布到可视化工作台中用于可视化页面搭建。从而满足开发者的需求，提升组件丰富性，为可视化搭建提供无限可能。

使用说明

组件开发功能升级中，暂停新用户申请开通白名单权限。

已开通白名单权限的用户可继续使用组件开发功能。

背景信息

IoT Studio的Web可视化开发中提供了很多组件用于搭建Web页面，由于提供的官方组件数量有限，很难满足用户所有的需求。因此，为了解决组件调用不足的问题，IoT Studio推出了组件开发功能。

IoT Studio组件目前使用的是react技术栈。任何一个普通的react组件都可以成为IoT Studio组件。IoT Studio将赋予react组件一些强大的能力，例如，为组件配置接口数据源，定时调用接口，根据接口的返回结果，动态设置组件本身的字体、颜色等样式；或赋予组件对外提供自定义好的服务功能，以供外部调用。

功能特性

- 个人开发组件和公共组件功能

  组件开发工作台提供了个人组件开发功能。新开发的个人组件仅支持开发者可见并使用，可通过将个人组件打包上传到公开组件包，供其他客户使用，且后续可支持正式商用。

- 强大的本地开发功能

  通过配套的组件本地开发工具（material-cli），支持实时开发和编译组件，并将其上传到组件开发工作台。在组件开发工作台支持实时预览组件效果，或模拟线上环境预览组件效果。

- 组件管理功能

  组件开发工作台提供了组件管理功能，包括创建、搜索、编辑、删除和发布等，便于开发者管理和使用更多的丰富组件。

- 快速上传和构建

  组件开发工作台支持一键上传和查看构建进度功能。将开发好的组件打包成ZIP文件一键上传后，可实时查看构建进度。IoT Studio系统会反馈构建结果到组件开发工作台，并产生相应的构建日志。如果系统反馈构建失败，可通过构建日志查看构建失败的原因。

- 组件操作文档管理

  组件开发工作台提供了可预览且会自动保存的Markdown编辑器，用于编辑组件操作文档，包括概述（必须编辑）、样式配置、数据源配置和动作配置等。针对需要市场化的组件，还提供了组件案例编辑功能，便于组件市场化。

- 一键发布

  组件开发工作台提供了组件一键发布功能，且支持组件的多次发布。代码构建完成后，即可一键完成组件的发布。如果开发者多次发布组件，支持查看发布历史和管理发布状态等功能。

相关文档

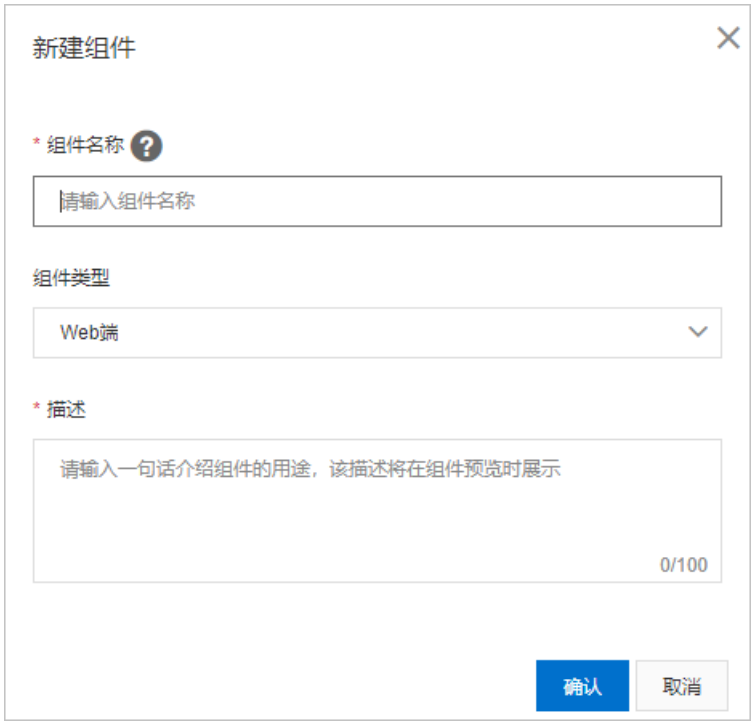
开发组件

2.创建组件

开发组件前，需先创建组件来定义组件名称、类型和功能特性描述，便于组件发布后引导开发者使用。本文介绍如何在组件开发工作台创建组件。

新增组件

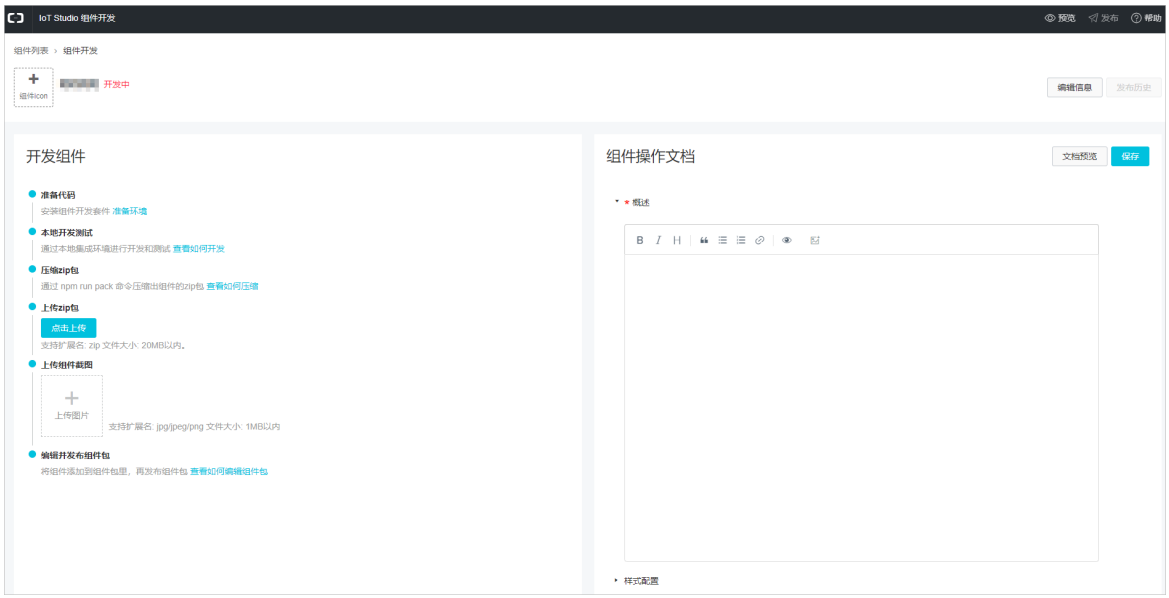
1. 登录物联网应用开发控制台，在页面左上角选择对应实例后，在左侧导航栏单击**应用开发**。
2. 在**应用开发**页面的**开发工具**模块，单击**组件开发**。
3. 在**组件开发 > 个人组件**页签，单击**新建组件**。
4. 在**新建组件**对话框，设置组件基本信息。



参数	配置说明
组件名称	自定义组件名称。建议输入便于识别的名称。 仅支持中文汉字、英文字母、数字、下划线（_）、连接号（-）和英文圆括号（()），且必须以中文汉字、英文字母或数字开头，长度不超过30个字符（一个中文汉字算一个字符）。 **说明** 组件创建完成后，支持修改组件的名称。
组件类型	选择新组件的应用类型。 ○ Web端：用于Web可视化开发 ○ 移动端：用于移动可视化开发 **说明** 组件创建完成后，不支持修改组件类型。

参数	配置说明
描述	输入描述信息介绍组件用途。该描述将在组件预览时展示。

5. 单击**确认**。
   组件创建完成，进入**IoT Studio组件开发**页面，开始开发个人新组件。具体操作，请参见开发个人组件。



查看组件列表

您可在**组件开发 > 个人组件**页面，查看到已创建的个人组件列表及组件的相关信息。

您可在搜索框中输入搜索关键词，定位搜索或筛选目标组件。

定位到目标组件，在组件右侧的操作栏单击以下操作。

操作	说明
预览	查看组件显示效果。 已上传组件代码包，才显示该功能。
开发	进入**IoT Studio组件开发**页面，根据页面提示开发组件。具体操作，请参见开发个人组件。

操作	说明
删除	删除开发中的组件。 注意　如果组件已发布，不再显示该功能。
历史发布版本	查看组件发布历史，支持**下载代码包**和**预览**。 已发布过的组件，才显示该功能。

后续步骤

组件基本信息创建完成后，开始开发组件属性和功能。具体操作，请参见开发个人组件。

3.开发个人组件

3.1. 概述

完成组件创建和相关信息编辑后，进行组件属性、服务和事件的开发。本文介绍如何开发组件。

前提条件

已完成组件的创建。详细内容请参见创建组件。

开发个人组件

在**组件开发**页面，根据组件开发流程图，进行单个组件开发和使用。

1. 准备环境
2. 本地开发测试
3. 打包本地项目文件
4. 上传组件压缩包和截图
5. 编辑组件
6. 发布与使用

公开组件包

将个人组件上传到组件包，公开到市场供其他客户使用。

1. 创建组件包
2. 编辑组件包
3. 发布与使用组件包

3.2. 准备环境

开发个人组件前，需要配置开发环境和初始化开发项目。

步骤一：安装Node.js运行环境

需要安装8.9 LTS以及以上版本：单击并选择版本下载。

步骤二：安装npm包管理工具

内测用户先使用bnpm，正式开放之后会迁移至aliplus源，使用npm即可。

1. 打开CMD命令窗口，执行**npm login**命令登录。

   > 说明　请访问bone.aliyun.com/profile，获取用户名和密码。

   ```
   npm login --registry https://registry-node.aliyun.com/org/YscSvypAq7EUU~8GmyAjOp/registry/aliyunIoT/
   ```

2. 执行以下命令安装新版bnpm。

   ```
   npm i -g @bone/npm --registry https://registry-node.aliyun.com/org/YscSvypAq7EUU~8GmyAjOp/registry/aliyunIoT/
   ```

步骤三：本地初始化项目

1. 打开CMD命令窗口，执行**bnpm login**命令登录。

   > 说明　请访问bone.aliyun.com/profile，获取用户名和密码。

   ```
   bnpm login --registry https://registry-node.aliyun.com/org/YscSvypAq7EUU~8GmyAjOp/registry/aliyunIoT/
   ```

2. 执行**bnpm install**命令安装material-cli。

   ```
   bnpm install -g @maliang/material-cli
   ```

3. 执行**material create <项目名>**，生成一个项目文件夹（例如Demo）到本地计算机。

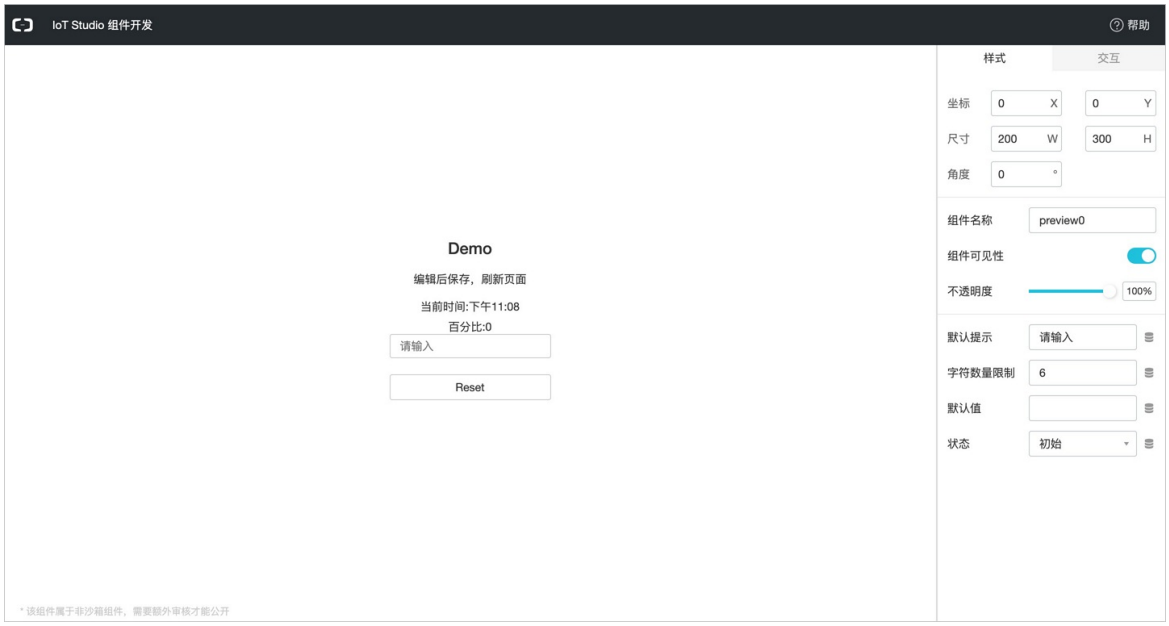
   ```
   material create Demo
   ```

4. 进入上一步创建的项目文件夹后，执行**bnpm install**命令安装必要的依赖文件。

   ```
   cd Demo
   bnpm install
   ```

5. 执行**npm run start**命令，自动打开浏览器，显示组件初始状态。该页面可用于测试组件开发的效

果。如下图所示是组件的初始化配置。



后续步骤

本地开发测试。

3.3. 本地开发测试

下文介绍组件开发过程和测试效果。

前提条件

已完成环境安装和项目文件初始化。详细内容请参见准备环境。

步骤一：开发属性文件specs

组件能力描述内容，主要集中在项目文件的*src/specs/specs.js*文件中，用于开发组件的属性、服务和事件功能。详细说明请参见组件能力描述。

1. 在项目文件中，找到并打开*specs.js*文件，可查看初始化组件配置及说明。
2. 根据业务需求，修改内容并保存。下文以配置布尔（bool）和颜色（color）类型属性为例。有关属性类型说明，请参见属性类型。

> **说明** 下文提供代码仅为开发示例，实际开发中，请替换为您需要设置的实际内容。

```
export default {
 meta: {
  cssPrefixer: 'material-'
 },
 "properties": [{
  "identifier": "$width",
  "defaultValue": 200,
  "options": {
```

```
    "min": 160
   }
  }, {
   "identifier": "$height",
   "defaultValue": 300,
   "options": {
    "min": 200
   }
  }, {
   "identifier": "boolstatus",
   "text": "是否显示",
   "type": "bool",
   "defaultValue": true
  },
  {
   identifier: 'searchBackgrund',
   text: "组件颜色",
   type: 'color',
   defaultValue: { r: 222, g: 222, b: 222, a: 1 },
   disableDataSources: 'all'
  }, ],
  "services": [{
   "identifier": "$getValue",
   "text": "取值",
   "effect": false
  }],
  "events": [{
   "identifier": "$change",
   "text": "值改变",
   "fields": [{
    "identifier": "value",
    "description": "当前值"
   }]
  }]
 };
```

步骤二：开发入口文件index

组件本身的代码开发，主要集中在*src/index.tsx*文件中。

该文件主要通过 `render(): JSX.Element{}` 方法实现组件展示配置，即使用 `this.props` 获取*specs.js*中配置的属性值，然后通过属性值控制组件展示的状态和形态。

有关属性定义的使用说明，请参见定义属性。

1. 找到并打开*index.tsx*文件，可查看初始化组件配置及说明。
2. 根据业务需求，修改内容并保存。下文配置了一个矩形作为组件形态，通过specs中的属性值，控制组件是否显示及显示的背景颜色。

```
import * as React from "react";
import BaseComponent from "@maliang/base-class-component";
import { Input, Button } from '@alifd/next';
import { colorObjectToCss, ColorObject } from '@maliang/visualapp-util-props';
import styles from './index.scss';
// 此处必须import，否则无法触发specs.js到specs.json的实时编译
import "./specs/specs";
export interface Props {
  $width: number;
  $height: number;
  boolstatus: boolean;
}
export interface State {
 value: string;
 defaultValue: string;
}
class BoolTest extends BaseComponent<Props, State, {}> {
  /* eslint-disable @typescript-eslint/no-explicit-any*/
  constructor(props: Props, context: any) {
    /* eslint-enable @typescript-eslint/no-explicit-any*/
    super(props, context);
    this.state = {
    };
  }
  // 如果不需要国际化和主题设置，则删除ConfigContext.Consumer相关代码，直接写jsx
  render(): JSX.Element {
    // 通过this.props获取specs中的配置值
    const { $width, width, $height, height, searchBackgrund, boolstatus } = this.props;
    const style = { width, height };
    // 配置属性作用的组件状态和形态
    return (
      <div className={styles['container']} style={style}>
        { boolstatus && <div className={styles['square']} style={{background:colorObjectToCss(search
Backgrund)}}></div>}
      </div>
    );
  }
}
export default BoolTest;
```

3. 打开*src/index.scss*样式文件，修改在 `render()` 中使用的样式container和square，并保存。

```
.container {
  text-align: center;
}
.square{
  width: 100px;
  height: 100px;
  background-color: #ff0000;
}
```

步骤三：预览组件效果

回到已在浏览器中打开的**组件开发**页面，该页面可自动更新配置的组件样式。

您可根据配置的属性，设置组件样式，如图所示。

后续步骤

打包本地项目文件

3.4. 打包本地项目文件

完成组件开发后，在组件项目的根目录里，执行以下命令，打包成zip文件。

前提条件

已完成本地开发测试。详细内容请参见本地开发测试。

操作步骤

1. 在组件项目的根目录下，执行**npm run pack**命令。

   ```
   npm run pack
   ```

   正常执行之后，生成一个与项目名相同的zip文件。
2. 上传zip包构建组件到IoT Studio项目中。详细内容请参见上传组件压缩包和截图。

3.5. 上传组件压缩包和截图

完成本地组件开发和打包后，需要上传组件zip文件到IoT Studio进行打包构建。下文介绍组件项目包构建的具体操作。

前提条件

已完成组件的项目文件打包。详细内容请参见打包本地项目文件。

操作步骤

1. 在**组件开发**页面的左侧流程的**上传zip包**下，单击**点击上传**。

2. 选择本地的组件压缩包文件后，单击**打开**。
   IoT Studio开始构建组件相关内容。

> **说明** 如果上传打包构建失败，可以通过构建日志查看失败原因。按照原因，进行修改后再次进行上传。

3. 预览组件并保存预览截图到本地。单击**预览组件**或页面右上角的**预览**，查看组件的实际效果，是否和在本地开发时一致。

您也可以在组件列表页面，定位到目标组件，单击其右侧操作栏的**预览**，直接预览组件效果。详细内容请参见查看组件列表。

4. 上传预览的截图。

> **说明** 上传预览图后，可在IoT Studio中快速预览组件。

i. 在**组件开发**页面的左侧流程的**上传组件截图**下，单击**上传图片**。

ii. 选择保存到本地的预览图片，并单击**打开**。

后续步骤

编辑组件

3.6. 编辑组件

下文介绍如何在组件开发工作台编辑组件的图标、基本信息和操作文档。

前提条件

已完成组件开发后的打包构建。详细内容请参见上传组件压缩包和截图。

步骤一：编辑组件图标

组件图标会显示在Web可视化开发的组件列表中，便于您使用的时候快速预览。

> **说明** 组件完成开发验证后，必须上传组件icon后，才支持发布并使用。

1. 单击**IoT Studio组件开发**页面左上角的**组件icon**，按照提示要求上传组件图标。

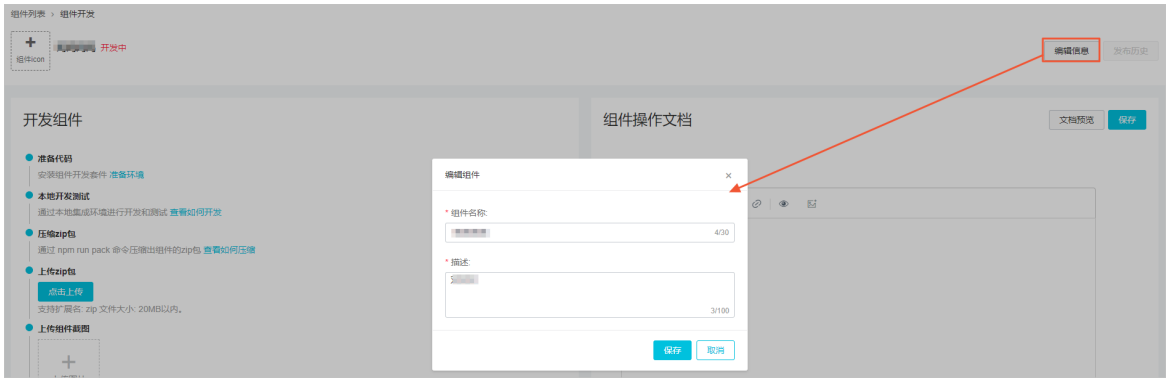
2. 单击**确认**。
   上传icon成功后，页面左上角会显示已上传的图标。鼠标移动至图标，显示**编辑**，可修改图标。

步骤二：编辑组件基本信息

发布组件前，支持修改组件名称及组件的描述信息。

> **说明** 组件发布成功后，仅支持修改组件描述信息，不再支持修改组件名称。

1. 单击**IoT Studio组件开发**页面右上角的**编辑信息**，按照提示要求修改组件信息。



2. 单击**保存**。

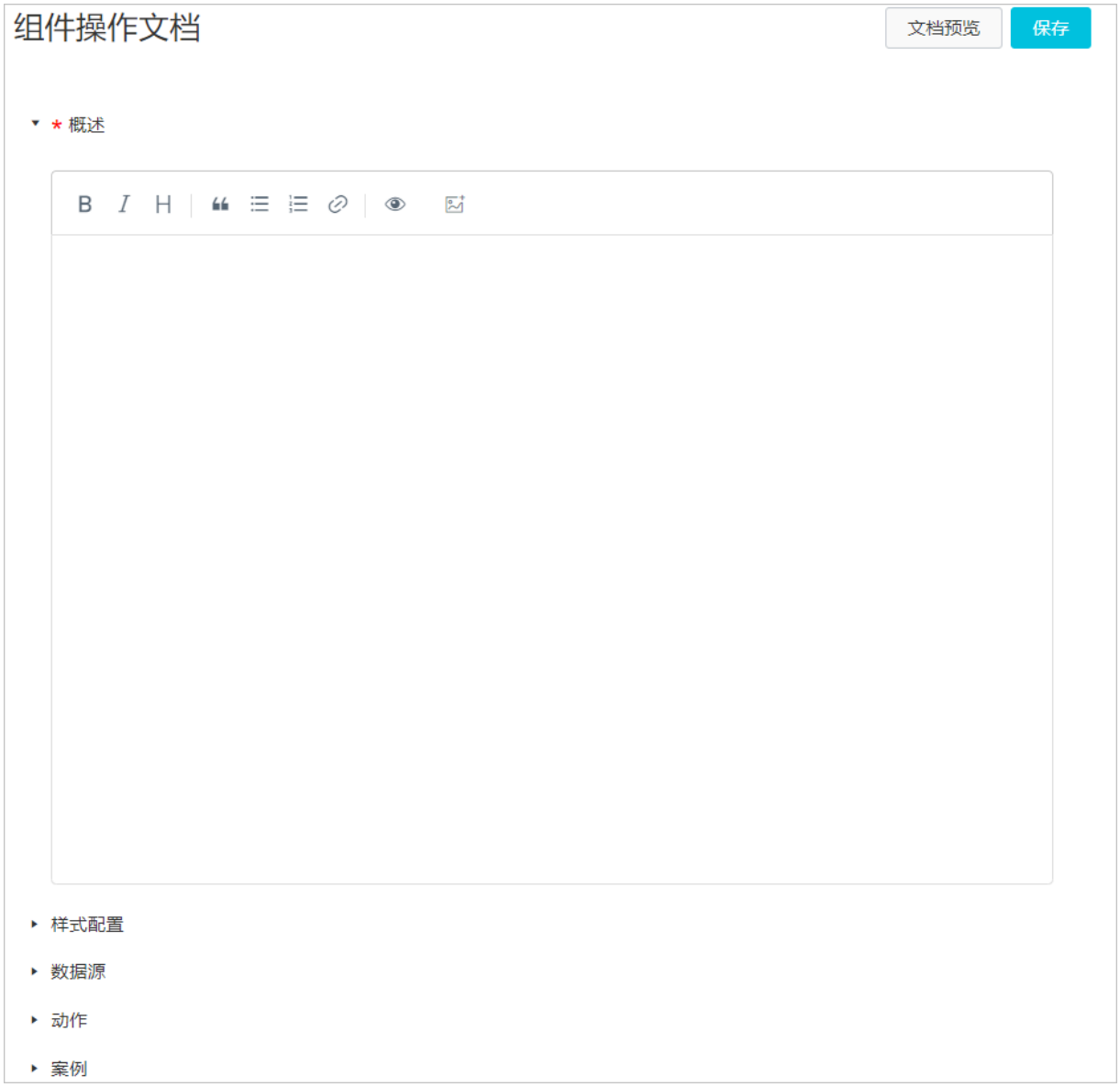
步骤三：编辑组件操作文档

组件的操作文档可帮助其他开发者了解并正确使用已公开的个人组件，该文档共分为五个部分：

- 组件概述：描述组件的基本情况，必填项。
- 样式配置：描述配置组件的样式内容。
- 数据源：描述组件的数据模拟和绑定。
- 动作：描述组件的动作配置。
- 案例：可在此提供组件的使用案例，供其他开发者或者使用者参考。

其中，组件概述内容必须完成并保存。组件的操作文档也可在组件开发中或开发完成后进行编辑。

编辑并保存内容后，单击**文档预览**可以查看文档实时效果。当组件被导入Web可视化开发后，在右侧的属性面板下面，会提供此文档的链接。



后续步骤

发布与使用

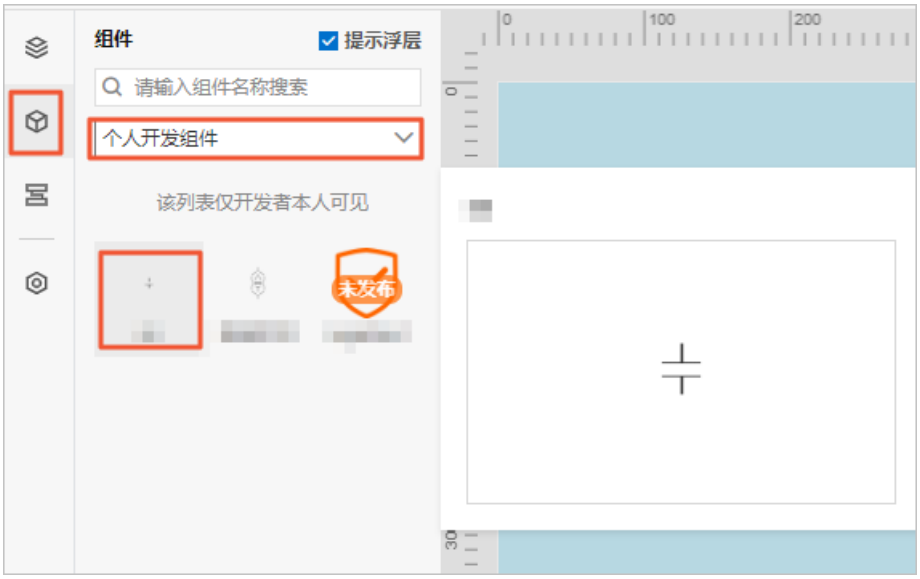
3.7. 发布与使用

个人组件开发验证完成后，为方便开发者使用个人组件开发可视化Web应用，需要完成个人组件的发布。本文介绍如何发布个人组件。

背景信息

个人组件开发完成后：

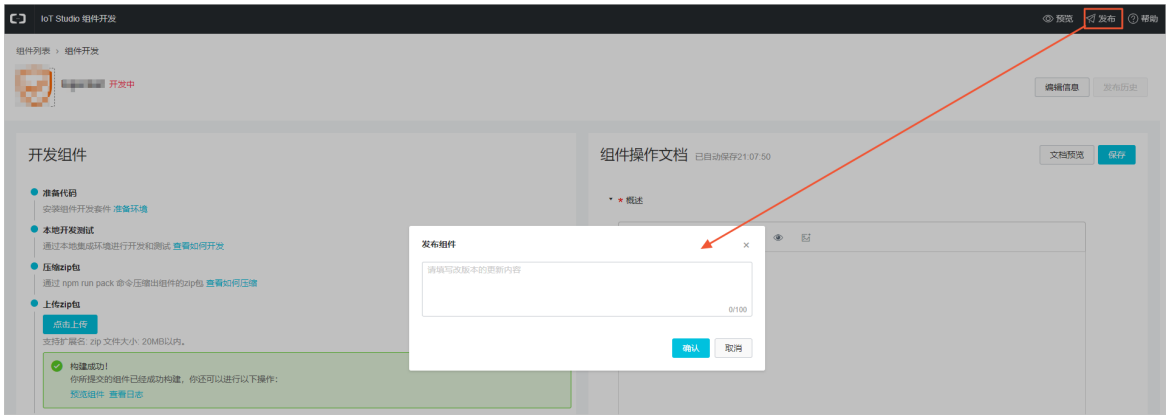
- 支持开发者在Web可视化编辑页面的个人开发组件列表预览未发布组件。



- 支持开发者使用未发布组件开发Web可视化应用。如果开发者在开发Web可视化应用时使用了未发布的个人组件，则发布Web应用后，该组件在Web页面是不可见的。

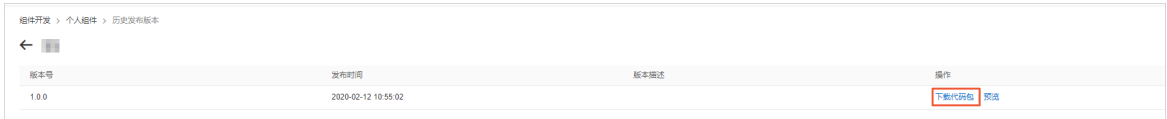
发布个人组件

1. 单击**组件开发**页面右上角的**发布**。



2. 输入组件发布的相关说明，单击**确认**。
   个人组件发布成功后，跳转至**组件开发** > **个人组件**页面。

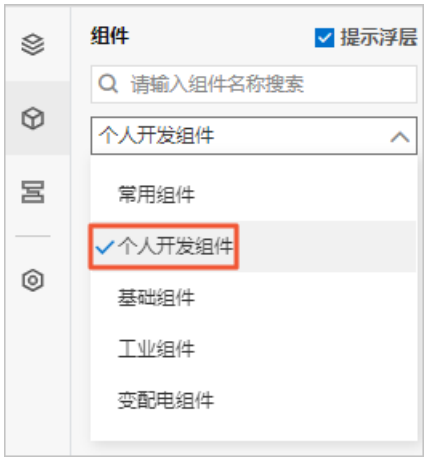
3. （可选）单击已发布的目标组件右侧的**历史发布版本**，可查看组件更新发布的记录。 您可以单击**下载代码包**，获取该版本的组件开发源码，用于组件的更新开发。



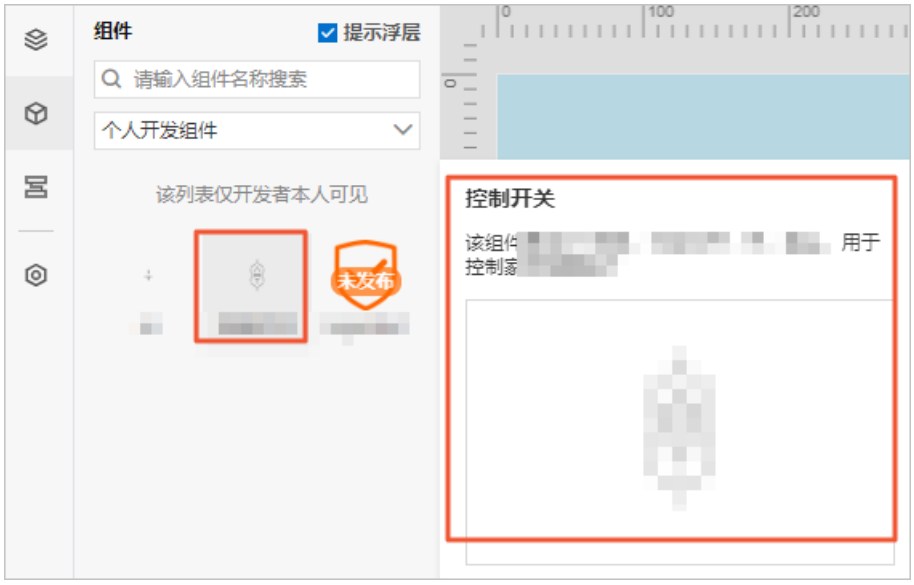
使用个人组件

个人组件仅支持开发者可见并使用。未发布的个人组件仅用于开发者调试，如果需要使用个人组件开发Web可视化应用，则开发者必须在使用个人组件前发布该组件。

1. 在Web可视化编辑页面左侧的组件列表栏，展开组件列表选择框，选择**个人开发组件**。



2. 在**个人开发组件**列表，可查看到开发者新增的个人组件。 鼠标移动到组件上，可预览组件图标和描述信息。将组件拖拽到画布后，即可使用该组件。



> **说明** 可通过将个人组件打包上传到公开组件包，供其他开发者使用。详细内容请参见发布与使用组件包。

4.公开组件包

4.1. 创建组件包

物联网应用开发（IoT Studio）支持将个人开发组件添加到组件包，发布到公开市场供其他开发者使用。本文介绍如何将个人组件以组件包的形式，发布到公开的组件市场。

新建组件包

1. 登录物联网应用开发控制台，在页面左上角选择对应实例后，在左侧导航栏单击**应用开发**。
2. 在**应用开发**页面的**开发工具**模块，单击**组件开发**。
3. 在**组件开发 > 公开组件包**页面，单击**新建组件包**。

4. 在**新建组件包**对话框，设置组件包的基本信息。

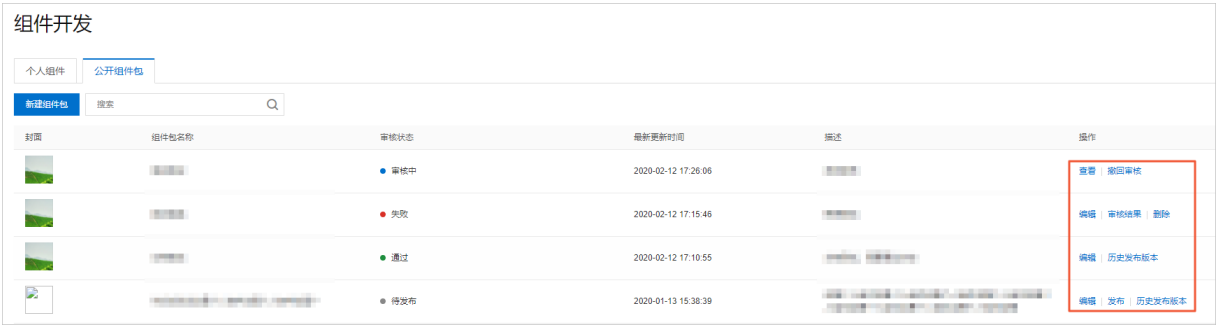
参数	配置说明
中文名/英文名	自定义组件包中、英文名称。建议输入便于识别的名称。 仅支持中文汉字、英文字母、数字、下划线（_）、连接号（-）和英文圆括号（()），且必须以中文汉字、英文字母或数字开头，长度不超过30个字符（一个中文汉字算一个字符）。 **说明** 组件包发布后，不支持修改组件包的中英文名称。
组件包封面	根据页面要求上传组件包的封面显示图。
描述	输入描述信息介绍组件包内容，长度不超过100个字符。该描述将在组件预览时展示。

5. 单击**确认**。

   组件包创建完成后，您可在**公开组件包**页面查看到初始组件包的相关信息（封面、组件包名称、审核状态、最新更新时间、描述）和操作功能（编辑、发布、删除等），更多详细内容请参见编辑组件包。

查看组件包列表

您可在**组件开发 > 公开组件包**页面，查看所有已创建的组件包。



在组件包列表中支持以下操作：

操作	审核状态	说明
编辑	待发布	单击该操作后，在**编辑组件包**页面，支持以下操作： • 添加个人组件 • 编辑组件包信息 • 发布与使用组件包
发布	待发布	单击该操作后，发布组件包。详细内容请参见发布与使用组件包。

操作	审核状态	说明
删除	待发布	单击该操作后，进行二次**确认**，删除组件包。 **说明** 仅支持删除从未发布（待发布或首次发布审核失败）到公开市场的组件包。已经审核通过且正式发布到公开组件市场的组件包，不支持删除功能。
历史发布版本	通过	单击该操作后，查看组件包的发布记录。详细内容请参见查看历史发布版本。
查看	审核中	单击该操作后，查看审核中组件包信息。
撤回审核	审核中	单击该操作并确认后，可撤回发布操作，继续进行组件包内容配置。
审核结果	失败	单击该操作后，可查看组件包发布审核失败的原因。 

后续步骤

为组件包添加组件，详细内容请参见编辑组件包。

4.2. 编辑组件包

本文介绍了如何在组件包中添加已发布的个人组件。

前提条件

已创建组件包。详细内容请参见创建组件包。

添加个人组件

1. 登录物联网应用开发控制台，在页面左上角选择对应实例后，在左侧导航栏单击**应用开发**。
2. 在**应用开发**页面的**开发工具**模块，单击**组件开发**。
3. 在**组件开发 > 公开组件包**页面，定位到目标组件包，单击操作栏的**编辑**。

4. 单击**编辑组件包**页面右上角的**添加组件**。

5. 在右侧的**添加组件**弹框中，勾选待添加的个人组件，单击**添加**。

> **说明**
>
> - 仅支持添加已发布的个人组件。
> - 仅支持添加最新版本的个人组件，不支持添加历史版本的个人组件。

个人组件添加完成后，页面自动跳转至**编辑组件包**。您可在组件列表中查看到已添加的个人组件信息。

您可预览个人组件效果、相关帮助文档，或根据需要从组件包中移除个人组件。

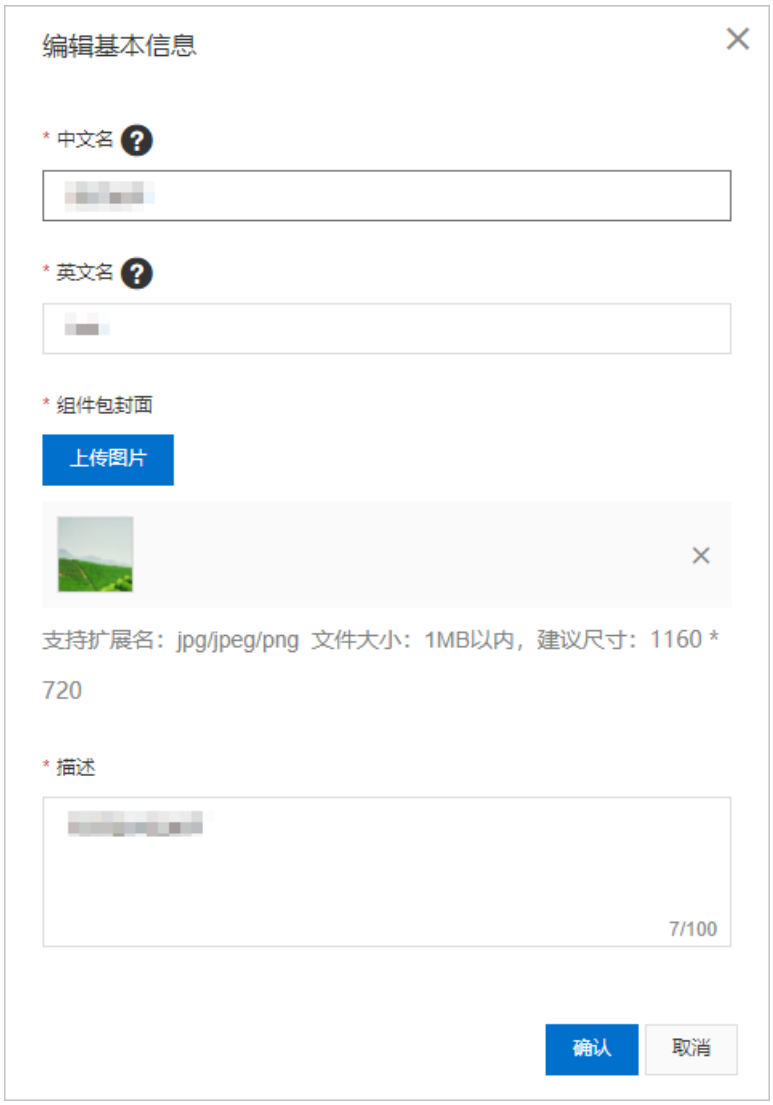
> **说明** 对组件包执行任何操作（例如：修改封面图片、添加组件、移除组件等）后，必须发布组件包并通过审核，对组件包执行的操作才会生效。

编辑组件包信息

1. 单击**编辑组件包**页面右上角的**编辑信息**。

2. 在**编辑基本信息**对话框，修改相关设置，并单击**确认**。

> **说明** 如果组件包有成功发布过的记录，不支持修改组件包的中、英文名称。



3. （可选）如果组件包中已添加了个人组件，您可根据实际需求选择移除个人组件。 在组件列表下定位到目标组件，单击其操作栏的**移除**，并进单击**确认**。

查看历史发布版本

您可以查看已通过发布组件包的历史版本信息。

1. 在**组件开发 > 公开组件包**页面，定位到目标组件包，单击操作栏的**历史发布版本**。
   您可查看组件包的历史发布版本信息。

2. 定位到目标版本号，单击其操作栏的**查看**。
3. 在**查看组件包**页面，查看对应版本下的组件包内容。

后续步骤

发布与使用组件包

4.3. 发布与使用组件包

本文介绍如何发布组件包到市场，实现个人开发组件的共享和使用。

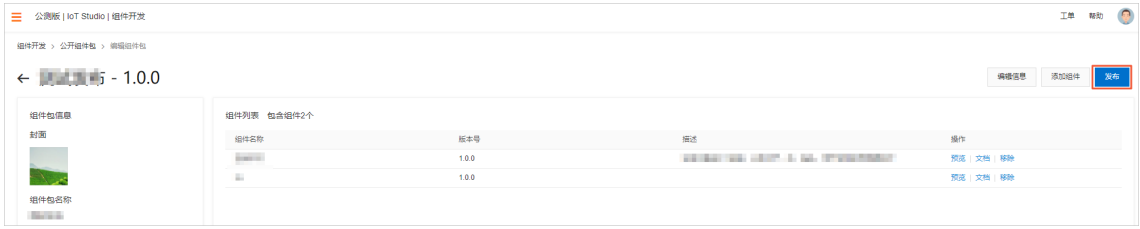
前提条件

已完成组件包的编辑。详细内容请参见编辑组件包。

发布

1. 登录物联网应用开发控制台，在页面左上角选择对应实例后，在左侧导航栏单击**应用开发**。
2. 在**应用开发**页面的**开发工具**模块，单击**组件开发**。
3. 在**组件开发 > 公开组件包**页面，定位到目标组件包。选择以下操作发布组件。
   - 单击操作栏的**发布**。

- 单击操作栏的**编辑**，在**编辑组件包**页面，单击组件列表右上方的**发布**。



4. 在**发布**对话框，输入组件发布的版本更新信息，单击**确认**。

> **注意** 组件包的版本更新信息必须正确且清楚地描述组件包更新的内容，用于帮助IoT Studio后台审核人员快速了解组件包更新内容，以便快速测试审核组件包，对是否通过该组件包的公开申请做出快速且正确的判断。

页面跳转到**组件开发 > 公开组件包**，已提交发布组件包的**审核状态**变为**审核中**。

组件包的状态说明请参见下表。

状态	场景说明
待发布	○ 创建组件包完成。 ○ 已通过发布的组件包，完成编辑更新。
审核中	组件包已提交发布。
通过	审核中组件包成功通过审核。
失败	审核中组件包未通过审核。

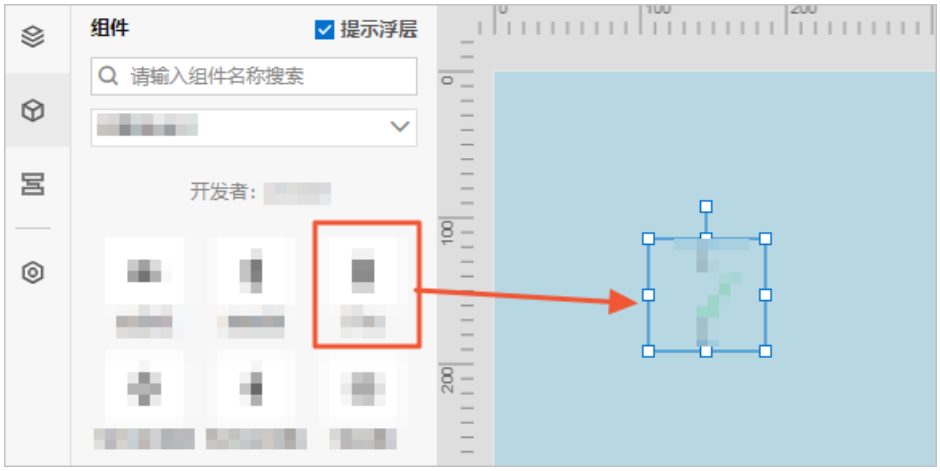
> 注意
>
> - IoT Studio仅支持公开已发布的组件包。
> - 在组件包审核期间，无法修改组件包配置。
> - 在同一个阿里账号下，如果已发布公开的解决方案中包含未公开的个人组件，页面将会出现异常，组件无法正确加载。所以，在发布公开解决方案之前，请确保其中使用的所有个人组件在公开的组件包中。

使用

组件包的使用分为以下两种场景。

场景一：开发者使用自己公开到市场的组件包。

1. 在Web可视化编辑页面左侧的组件列表栏，展开组件列表选择框，选择已发布的组件包名称。
2. 将组件拖拽到画布后，即可使用该组件。

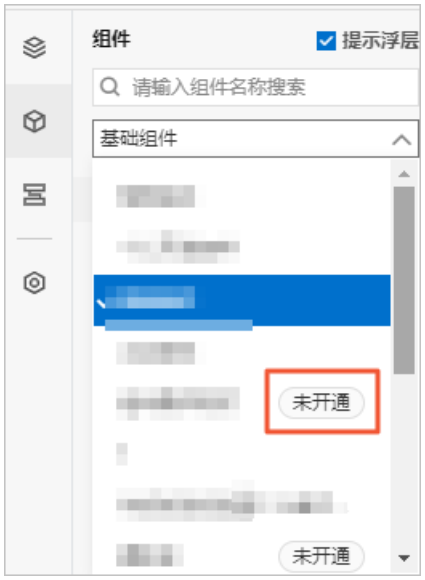


场景二：其他开发者使用非自己公开到市场的组件包。

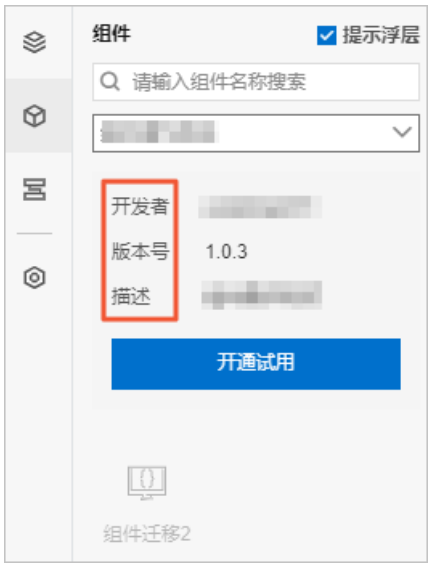
> 说明　其他开发者需要开通试用功能后，才支持使用组件包下的组件。

1. 在Web可视化编辑页面左侧的组件列表栏，展开组件列表选择框，选择已发布的组件包名称。

> 说明　在组件列表中，未开通的组件包右侧会显示**未开通**提示。



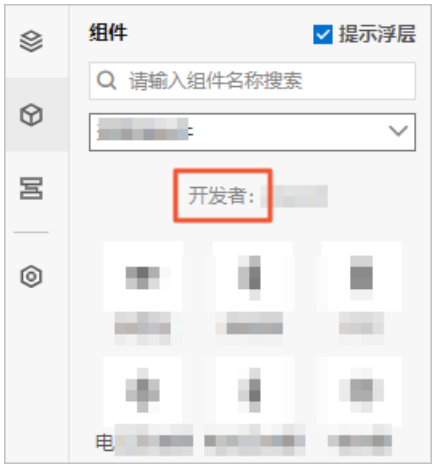
选中某一组件包后，可查看到该组件包的开发者、版本号以及描述信息。



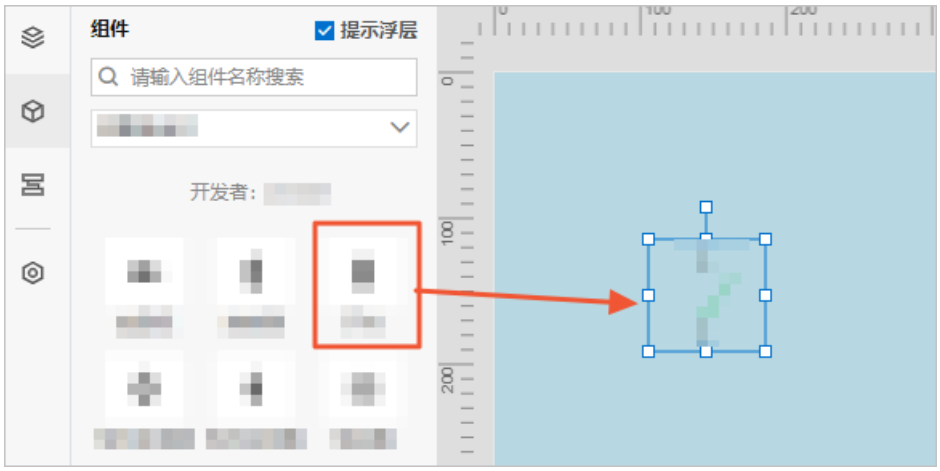
- 单击**查看文档**，可查看组件包帮助文档。
- 组件包下的组件均灰色显示，不支持拖拽使用。
- 鼠标移动到组件上，可预览组件图标和描述信息。

2. 单击**开通试用**后，可开通组件包的使用权限。

> **说明** 开通组件包使用后，组件显示栏仅显示该组件包的开发者信息。



3. 将组件拖拽到画布后，即可使用该组件。



组件包版本升级后说明

- 当新版组件包发布时，如果调整或修改了上一版本里的个人组件：
  - 如果旧版本的组件已经在Web可视化页面中使用，则页面中已经添加的旧版组件仍然可用。
  - 如果需要在页面中新添加该组件，则仅支持添加新版组件。
- 当新版组件包发布时，如果删除了上一版里的组件：
  - 如果旧版本的组件已经在Web可视化页面中使用，则页面中已经添加的旧版组件仍然可用。
  - 新的组件包中不再显示旧版组件，即不再支持添加该组件。

5.组件能力描述

5.1. 概述

组件的描述文件描述了一个IoT Studio组件的属性、服务和事件。本文介绍组件描述文件的结构。

通过设置组件的属性、服务和事件，可以让一个IoT Studio组件根据用户自身业务的需要，进行相关的设置，达到改变组件样式、关联设备数据源等目的。

IoT Studio组件的强大能力，是通过编写*src/specs/specs.js*来实现的，主要是由属性、服务、事件三个部分构成。

```
{
 // properties：属性，描述组件的属性列表，例如一个组件有字体、颜色等属性可以设置，放在此处。
 "properties": [
 {
  "identifier": "fontFamily",
  "text": "字体",
  "type": "enum",
  "dataTypes": [
   "String"
  ],
  "options": {
   "items": [
    {
     "label": "微软雅黑",
     "value": "\"Microsoft YaHei\""
    },
    {
     "label": "宋体",
     "value": "SimSun, STSong"
    },
    {
     "label": "黑体",
     "value": "SimHei, STHeiti"
    }
   ]
 },
 {
  "identifier": "color",
  "text": "颜色",
  "type": "color",
  "defaultValue": {
```

```
        "r": 51,
        "g": 51,
        "b": 51
    },
  }],
  // services：服务，描述组件对外提供的服务列表，例如组件可以提供一个返回自己内部某个属性值的方法。
  "services": [{
   "identifier": "$getValue",
   "text": "取值",
   "effect": false
  }],
  // events：事件，描述组件的动作列表，例如组件提供单击功能。单击组件后，组件可以返回某些属性值。
  "events": [{
   "identifier": "click",
   "text": "点击",
   "fields": [{
    "identifier": "status",
    "description": "当前值"
   }]
  }, {
   "identifier": "doubleClick",
   "text": "双击",
   "fields": [{
    "identifier": "status",
    "description": "当前值"
   }]
  }, {
   "identifier": "mouseEnter",
   "text": "鼠标移入",
   "fields": [{
    "identifier": "status",
    "description": "当前值"
   }]
  }]
}
```

> **说明** *src/index.tsx*代码中 updateValue 方法下 this.$emit 的第二个参数字段必须与*src/specs/specs.js*代码中 events.fields 的 identifier 字段保持一致。
>
> 例如上述示例 events.fields.identifier 为 status，则 this.$emit 为 this.$emit('event',{status}) 。

有关组件开发示例的更多详情，请参见本地开发测试。

属性

IoT Studio为组件的属性提供了可定制的编程能力，来实现一些较为复杂的组件属性联动功能。

开发者可以为一个组件设置一些不同的属性，例如组件的文案、尺寸、位置等；可以根据业务的要求，动态地设置组件的属性，例如一个普通的下拉列表组件里面的内容；可以根据组件数据源（组件可以配置数据的来源来动态地接收数据）的返回值，展示不同的内容列表。

服务

组件可以为自己定义具体的方法，来执行命令，例如返回组件的某个属性值。定义的方法可以提供给外部使用，即该方法就是组件的服务。

事件

组件可以定义一些用户与浏览器的交互行为，由用户在界面操作时触发，触发后，可以返回一些组件内部的状态值。这种交互行为的触发就是组件的事件。

5.2. 定义属性

下文介绍如何定义属性。

示例

1. 开发组件的*specs.js*文件内容：

```
export default {
 "properties": [{
  "identifier": "temperature",
  "defaultValue": 26,
  "text": "温度",
  "type": "text",
 }],
 "services": [],
 "events": []
};
```

   属性（properties）设置：类型（type）是*text*，属性中文名（text）是*温度*，默认值（defaultValue）是 26，组件唯一标识符（identifier）是*temperature*的属性。

2. 开发入口文件*index.tsx*的render方法：

```
render(): JSX.Element {
  const { temperature } = this.props;
  return (
    <div>
      当前温度是：{temperature}
    </div>
  );
}
```

3. 执行以下命令运行组件的应用文件：

```
cd <项目名>
# 安装依赖
bnpm install
# 启动应用
npm run start
```

从组件入口文件*index.tsx*的 `this.props` 中，通过唯一标识符identifier获取到了tempature这个属性值，并展示在页面中：

名词说明

当属性（properties）设置中包含了options、visible或linkage的字段时，如果该字段以函数类型（function）入参时，涉及如下两个名词：

- 配置值：未设置数据源时，该属性在输入框中的值。在options/visible/linkage的函数形式里，是props参数。
- 运行值：设置了数据源时，该属性从数据源中获得的值。在options/visible/linkage的函数形式里，是propValues参数。

字段options/visible/linkage以函数类型入参的格式如下：

> **说明** 其中function可取字段为options、visible或linkage。

```
function({ props, propValues }) {
 ...
}
```

具体每个函数可支持设置的参数字段，详细内容请参见属性列表。

属性列表

如果把属性（properties）设置 `"properties": [{"props1":"","props12":"",...},{},...]` 中的每一个 `{"props1":"","props12":"",...}` 看作一个入参对象，支持配置的字段（props）如下表所示。

字段	类型	必须	说明
identifier	string	是	属性标识符，可以在组件使用 `this.props[identifier]` 获取配置值。
text	string	是	属性名称，显示在Web可视化工作台右侧属性编辑栏。
description	string	否	当前属性的描述文案，会展示成问号icon。鼠标移动至该icon上可以看到属性的描述。
type	string	是	接口类型标识符，设置**type**时，按需填入，包括以下几种类型：bool、text、number、color、enum、image、group、pureDataSource。详细内容请参见属性类型。
options	object/function	否	接口的配置，会传递给**type**对应的接口。 如果参数类型是function，则传入以下两个参数： • **props**：属性的配置值 • **propValues**：属性的运行值，需要返回object
defaultValue	any	否	属性初始值。
mock	any	否	属性为空时的mock值。与**defaultValue**的区别是不会展示在接口配置中。
visible	boolean/function	否	是否展示该属性。 如果参数类型是boolean，取值： • **true**：展示 • **false**：不展示 如果参数类型是function，则传入以下两个参数： • **props**：属性的配置值 • **propValues**：属性的运行值，返回*true*或*false*

字段	类型	必须	说明
linkage	function	否	用作属性联动。接收以下参数： • changedProps：发生改变的属性 • props：当前属性变化后的配置值 • propValues：当前属性变化后的运行值 • prevProps：当前组件未变化前，属性的配置值 • revPropValues：当前组件未变化前，属性的运行值 该函数的返回有三种值： • null：表示清空当前属性 • undefined：表示保持当前属性的值 • 其他：设置该属性的值为其他值
validate	function	否	属性校验。传入以下两个参数： • **props** • **propValues**
displayIdentifier	string	否	用来判断是否启用当前配置项的prop，最终组件也会获取这个prop值。
displayDefaultValue	boolean	否	是否显示属性初始值。 • true：显示 • false：不显示
displayLinkage	function	否	用作属性display联动。接收以下参数： • changedProps：发生改变的属性 • props：组件配置值 • propValues：组件运行值 • prevProps：当前组件未变化前，属性的配置值 • revPropValues：当前组件未变化前，属性的运行值 该函数的返回有三种值： • null：表示清空当前属性 • undefined：表示保持当前属性的值 • 其他：设置该属性的值为其他值

5.3. 属性类型

下文介绍了属性配置支持的类型。

bool：开关

属性的value取值：*true*/*false*

属性的配置栏显示的示例：

代码示例：

```
export default {
  "properties": [{
    "identifier": "switch",
    "defaultValue": false,
    "text": "电源开关",
    "type": "bool",
    "description": "这是电源开关",
  }],
  "services": [],
  "events": []
};
```

有关bool类型属性使用示例，可参见本地开发测试。

number：数字输入框

属性的value取值：number型数字

属性的配置栏显示的示例：

属性相关的字段：

参数字段	类型	默认	描述
options	object	/	设置额外参数的字段
options.min	number	-Infinity	允许的最小值
options.max	number	Infinity	允许的最大值
options.step	number	1	步长值，输入框右侧会出现上下箭头
options.placeholder	string	null	占位符，没有输入时，在输入框中显示的文案

代码示例：

```
export default {
 "properties": [{
  "identifier": "temperature",
  "defaultValue": 26,
  "text": "温度",
  "type": "number",
  "options": {
   "min": -10,
   "max": 36,
   "step": 2,
   "placeholder": "请输入温度"
  }
 }],
 "services": [],
 "events": []
};
```

text：文本输入框

属性的value取值：string型的文本框

属性的配置栏显示的示例：

属性相关的字段：

参数字段	类型	默认	描述
options	object	/	设置额外参数的字段
options.maxLength	number	undefined	最大输入长度
options.placeholder	string	请输入	占位符
options.readOnly	boolean	false	是否只读，若为true，则不能手动输入

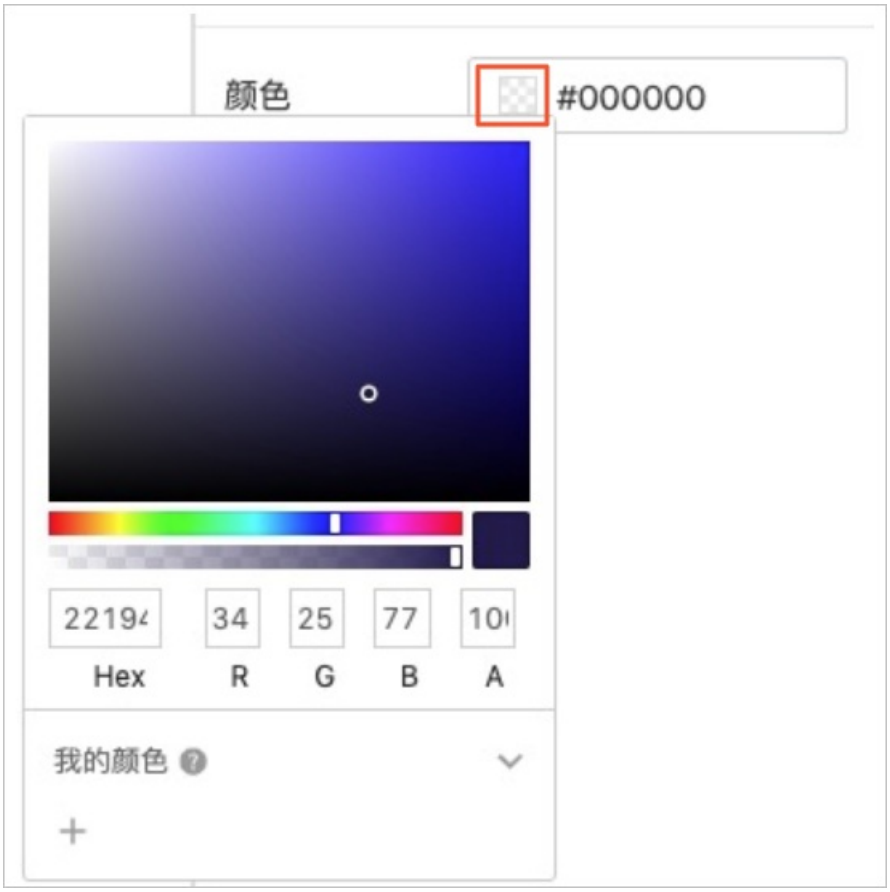
代码示例：

```
export default {
 "properties": [{
  "identifier": "temperature",
  "text": "温度",
  "type": "text",
  "defaultValue": "温度文案",
  "options": {
   "placeholder": "请输入温度",
   "maxLength": 10,
   "readOnly": true,
  }
 }],
 "services": [],
 "events": []
};
```

color：选择颜色

属性的value值：{ r: number, g: number, b: number, a: number }

属性的配置栏显示的示例：



color属性可提供配置选择颜色功能：单击输入框中左侧的色块icon，弹出如上图所示的调色盘。

属性相关字段：

参数字段	类型	默认	描述
options	object	/	设置额外参数的字段
options.alpha	boolean / number	true	boolean类型：是否隐藏alpha值配置 • false：隐藏 • true：显示 number类型：初始alpha值，默认值为1

> **注意** 数据格式为 { r: number, g: number, b: number, a: number } ， 暂不支持其他格式。render的时候，建议手动转换成十六进制。

代码示例：

```
export default {
 "properties": [{
  "identifier": "color",
  "text": "颜色",
  "type": "color",
 }],
 "services": [],
 "events": []
};
```

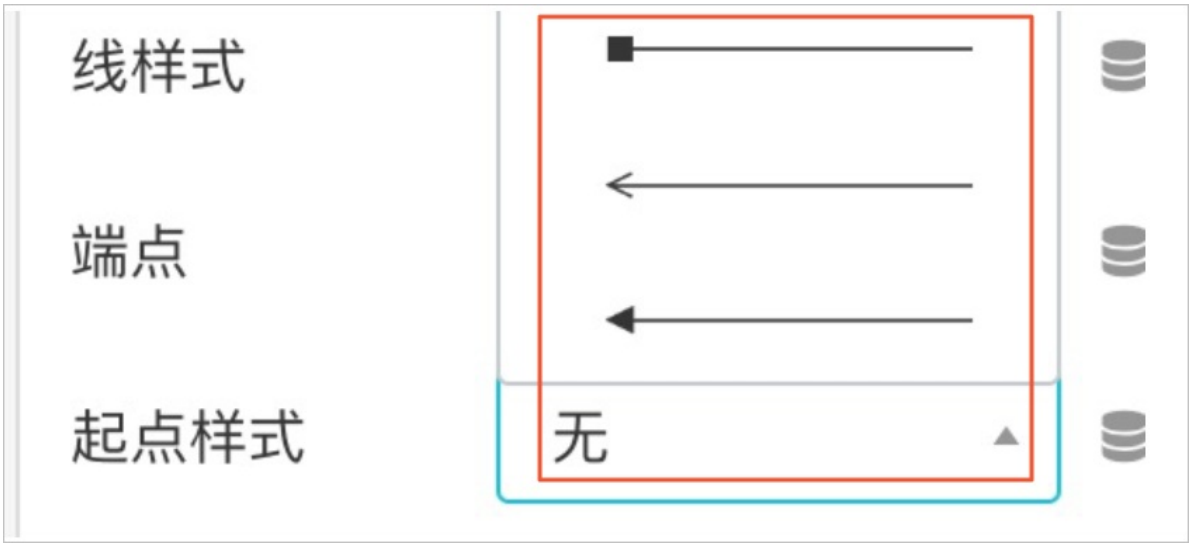
enum：下拉选择

属性的value值为string型的数据，支持的item.type类型：*text*、*image*、*icon-text*

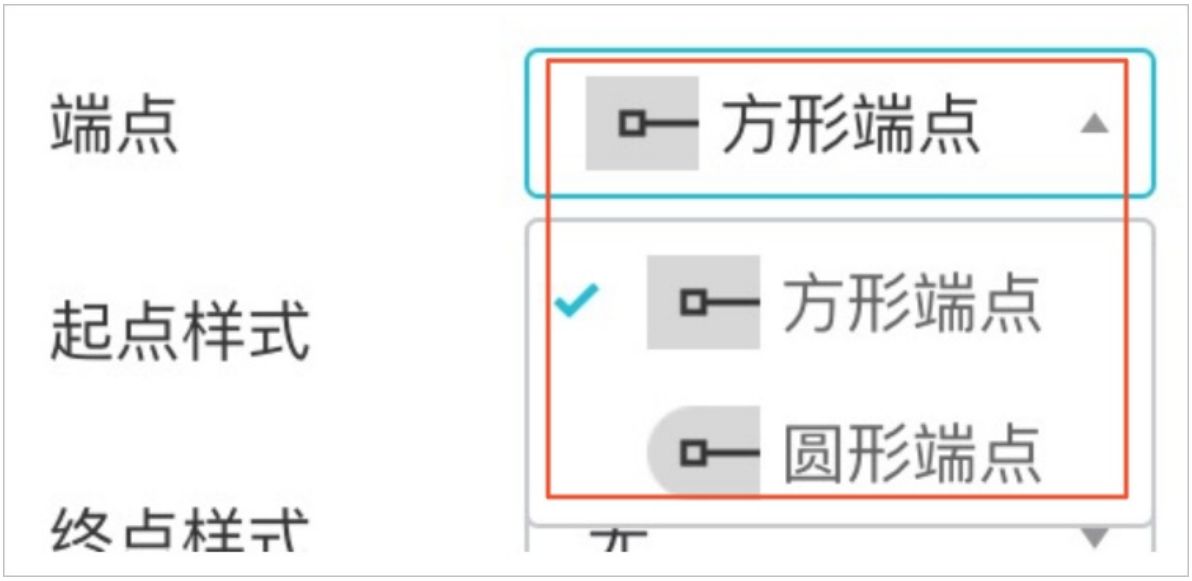
属性的配置栏显示的示例：

- item.type = text

- item.type = image



- item.type = icon-text



属性相关字段：

参数字段	类型	默认	描述
options	object	/	设置额外参数的字段
options.items	Array	[]	选项列表
options.items[].label	string	null	选项的展示文字
options.items[].type	string，以下值三选一：imageicon-texttext	text	选项的展示类型，可以混搭，例如第一个item.type可以是text，第二个item.type可以是image
options.items[].value	any	undefined	选项的实际值

参数字段	类型	默认	描述
options.items[].url	string	null	选项的type为image或icon-text时，图片的地址
options.multiple	boolean	false	是否可以多选
options.placeholder	string	null	默认填充文案

代码示例：

```
{
  identifier: 'valueFont',
  text: '测试',
  type: 'enum',
  options: {
    items: [
     { label: '16', value: '16' },
     { label: '14', value: '14' },
      {
        "type": "image",
        "url": "https://img.alicdn.com/tfs/TB1mAL1khD1gK0jSZFsXXbldVXa-240-16.png",
        "label": "虚线",
        "value": 2
      },
      {
       "type": "icon-text",
       "url": "https://img.alicdn.com/tfs/TB1FNnCj.D1gK0jSZFGXXbd3FXa-34-28.png",
       "label": "圆形端点",
       "value": 1
      },
    ],
    multiple: true,
  },
  defaultValue: 16,
 },
```

options支持function的写法，可动态调整选项值，如下示例：

```
export default {
 "properties": [{
  identifier: 'value',
  type: 'text',
  text: '属性值'
 }, {
  identifier: 'directionOption',
  text: '测试',
  type: 'enum',
  disableDataSources: 'all',
  options: ({ props }) => {
   let opt;
   if (props.value === 'boolean'){
    opt = {
     items: [
      { label: '1', value: true },
      { label: '0', value: false },
     ]
    }
   }
   if (props.value === 'number') {
    opt = {
     items: [
      { label: '>', value: '>' },
      { label: '<', value: '<' },
     ]
    }
   }
   return opt;
  },
 }],
 "services": [],
 "events": []
};
```

显示效果：

image：图片

属性的value值：string型的图片URL

image可提供图片地址选择框，即属性的配置栏显示的示例：

属性相关字段：

参数字段	类型	默认	描述
options	object	/	设置额外参数的字段
options.items	Array	[]	默认的图片列表

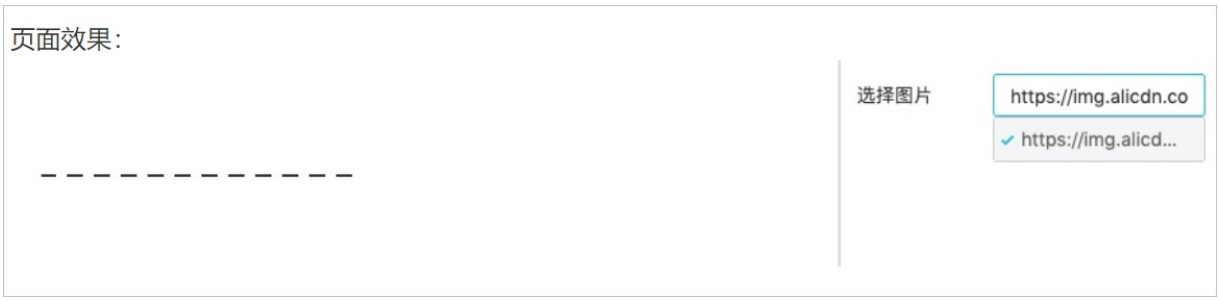
代码示例：

```
export default {
 "properties": [{
  identifier: 'avatar',
  type: 'image',
  text: '选择图片',
  defaultValue: "https://img.alicdn.com/tfs/TB1S8vZqIj_B1NjSZFHXXaDWpXa-244-244.png",
  options: {
   items: [
   'https://img.alicdn.com/tfs/TB1FNnCj.D1gK0jSZFGXXbd3FXa-34-28.png',
   'https://img.alicdn.com/tfs/TB1mAL1khD1gK0jSZFsXXbldVXa-240-16.png',
  ]
  }
 }],
 "services": [],
 "events": []
};
```

*index.js*的 `render()` :

```
render(): JSX.Element {
    const { avatar } = this.props;
    return (
      <div>
        <img src={avatar} />
      </div>
    );
  }
```

显示效果:



group：成组

将多个属性在右侧编辑栏编成一组。

属性的配置栏显示的示例:

属性相关字段：

参数字段	类型	默认	描述
displayIdentifier	string	null	表示这个group的checkbox的值，false或者true
displayDefaultValue	boolean	无	配置了这个字段的话，分组名左侧会出现一个checkbox；不配置就不出现该checkbox。false：表示没有勾选该group，group下的属性组不会展示，但是值仍然能在代码中获取到；true：表示勾选了该group，group下的属性组的展示出来。

代码示例：

```
export default {
 "properties": [{
  identifier: 'arrowGroup',
  text: '流动箭头',
  type: 'group',
  displayIdentifier: 'showArrow',
  displayDefaultValue: true,
  options: {
   items: [
    {
     identifier: 'arrowDirection',
     text: '箭头方向',
     type: 'enum',
     options: {
      items: [
       { label: '向左', value: 'left' },
       { label: '向右', value: 'right' },
      ],
     },
     defaultValue: 'left',
    },
    {
     identifier: 'arrowColor',
     text: '箭头颜色',
     type: 'color',
     defaultValue: { r: 14, g: 155, b: 255, a: 1 },
    },
   ]
  }
 }],
 "services": [],
 "events": []
};
```

pureDataSource：纯数据源属性

表示该属性是用于设置数据源的。

属性的配置栏显示的示例：

属性相关字段：

参数字段	类型	默认	描述
dataTypes	array	[]	可以接受的数据源返回的数据类型，如果返回的不是dataTypes里指定的类型，则在可视化工作台里报错。可选值：Array、Boolean、DateTime、Enum、Interger、JSONObject、Number、OneDemesionData、TwoDemesionData、PagingTwoDemesionData、RgbaColor、String
needPropValue	boolean	false	在options/linkage等函数里，如果属性配置了数据源，且函数实现希望用到propValues，则需要设置needDataSourceValue为true，才能访问到propValues
options	object	/	设置额外参数的字段
options.dataSourceInfoErrorTip	string	null	数据源配置错误时的提示文案
options.showDataSourceInfo	boolean	false	是否展示数据源的来源

代码示例：

```
export default {
 "properties": [{
  identifier: 'dataSource',
  text: '设备名称',
  type: 'pureDataSource',
  dataTypes: ['Number', 'Enum', 'Boolean'],
  options:{
   dataSourceInfoErrorTip:'接口需要返回数值、布尔或枚举值才能进行风机的状态配置',
   showDataSourceInfo: true,
  },
 }],
 "services": [],
 "events": []
};
```