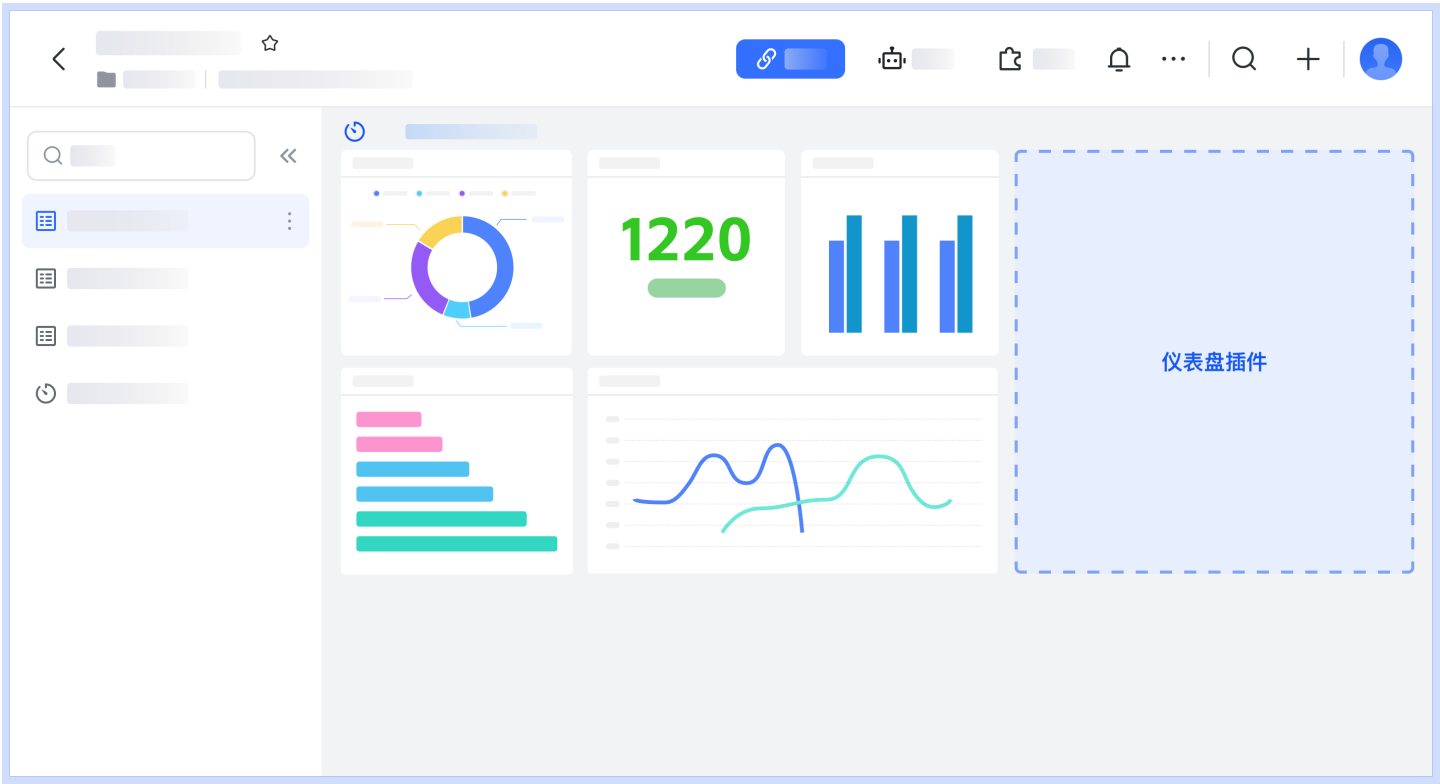
仪表盘插件开发指南

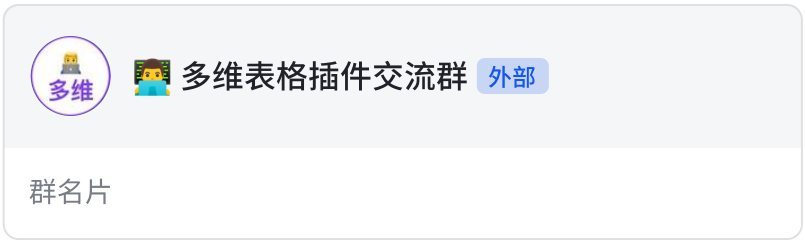
介绍

仪表盘提供了一个可视化呈现表格数据的面板，开发者可以构建仪表板组件来扩展仪表板原生功能。



寻求帮助

如果在开发过程中遇到任何困难，或有任何反馈，请加入交流群，发起话题，与运营人员及其他开发者一起进行讨论。

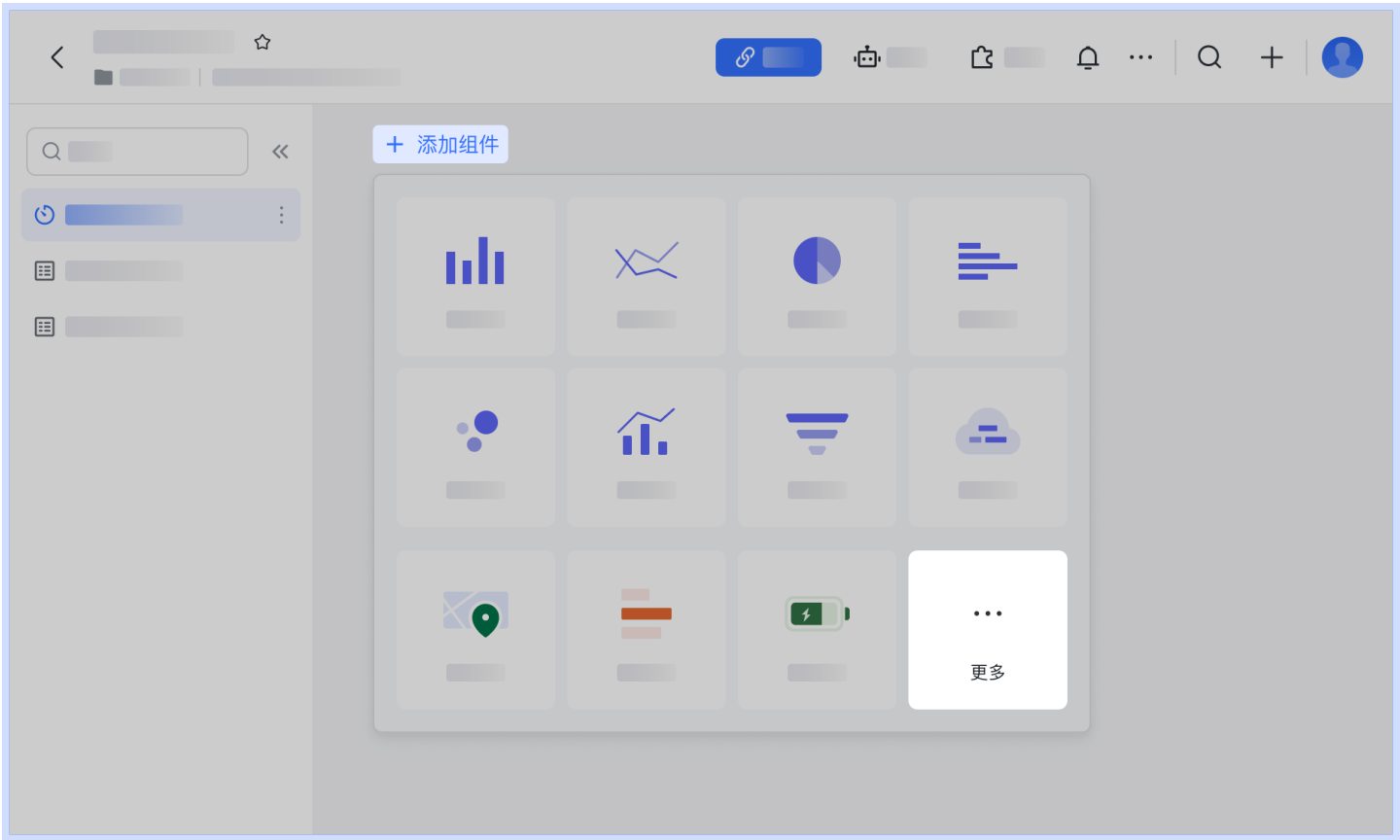


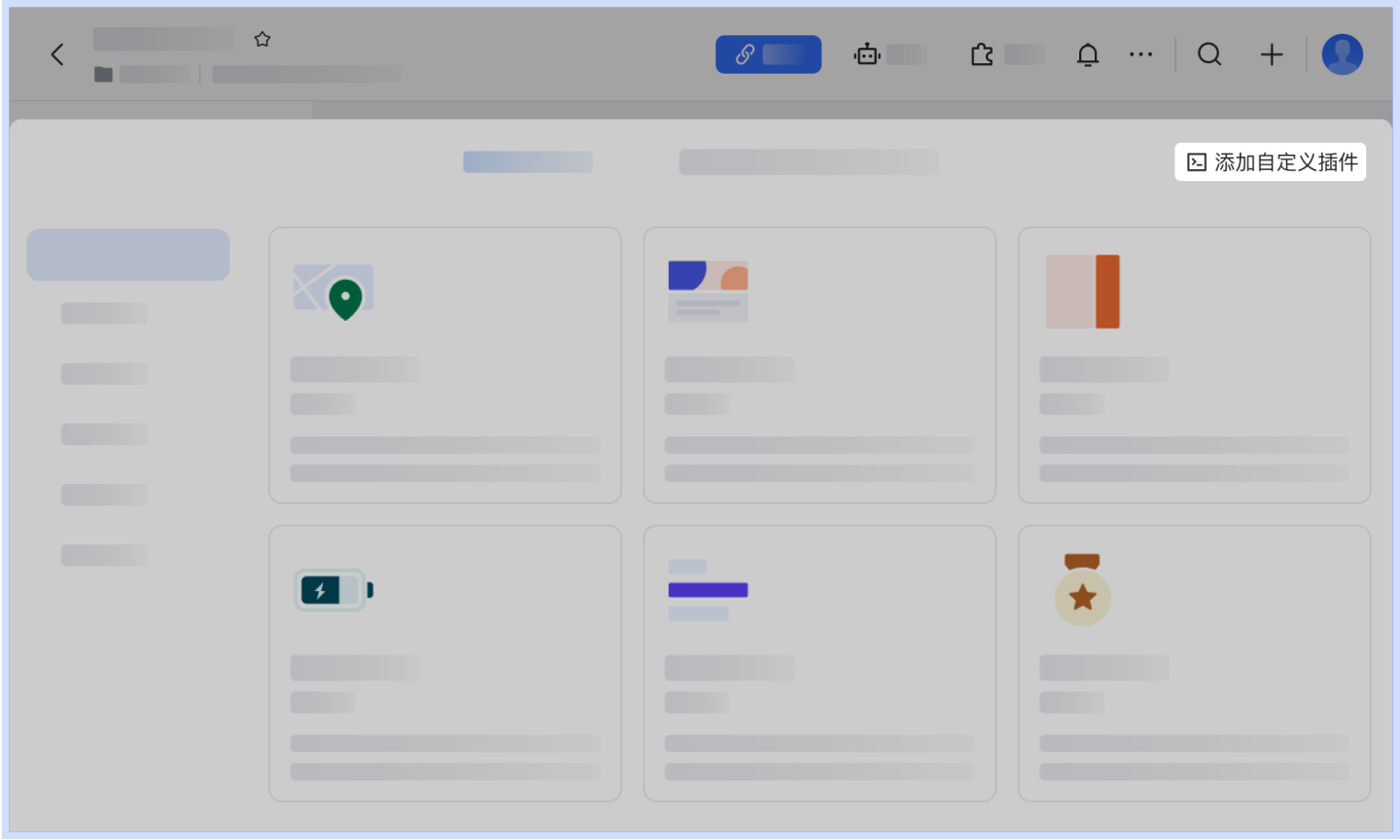
如果你有特殊的需求又没有开发资源，可以向其他开发者 提交插件需求，开发者也可以 查看需求汇总 来认领需求。

开始开发

跟随示例，尝试动手完成一个插件的搭建，对插件的开发流程建立直观认知。无论是 Vercel、Github、localhost，还是你自己的服务器，只要部署了服务，插件都可以在多维表格中正常运行。你可以直接在多维表格的控制台中查看调试信息。

- 新建或打开任意仪表盘，点击 `添加组件` 展开快捷面板，点击 `更多` 打开插件市场
- 点击 `添加自定义插件` 在输入框内填入运行地址后点击 `确定` 添加并运行插件





开发模板

我们提供了插件示例模板供开发者参考，注意此项目为 react 框架。

- 倒计时插件模板

SDK

> ❗ 当前 SDK 处于内测状态，**随时可能发生变更，请密切关注文档变更及开发群通知**

我们为仪表盘插件的开发提供了独立的 SDK 模块，该模块提供了仪表盘插件特有的 `配置` 和 `计算` 接口，传入计算参数后由服务端完成计算并将计算结果以二维数组的形式返回，无需将全量数据 meta 拉取到内存中组织，极大的提升了性能。

- `Dashboard` 模块仅用于仪表盘插件，其他点位的插件无法调用该模块的接口。
- 仪表盘插件可以调用 JS SDK 其他模块中的接口，但极少部分接口的返回和其他点位有不一致的情况出现，具体请参考具体接口文档中的备注。
- Base JSSDK Dashboard 插件 API 文档
- Base JS SDK

仪表盘插件状态

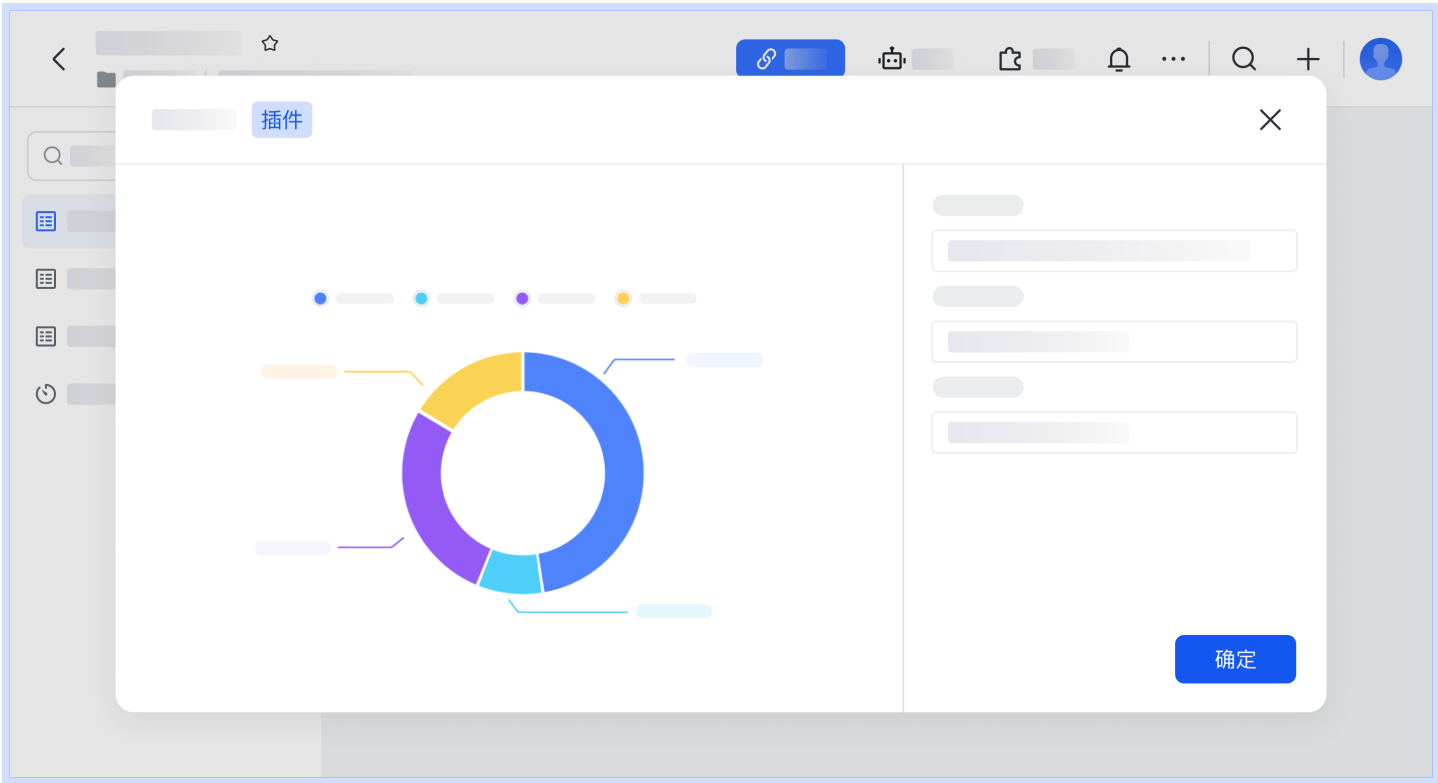
相比侧边栏插件，仪表盘插件存在多个状态，开发者需要在插件代码中感知多个不同状态并做出相应的处理。

代码块

```
enum DashboardState {
    Create = 'Create', // 创建状态
    Config = 'Config', // 配置状态
    View = 'View', // 展示状态
    FullScreen = 'FullScreen' // 全屏状态
}
```

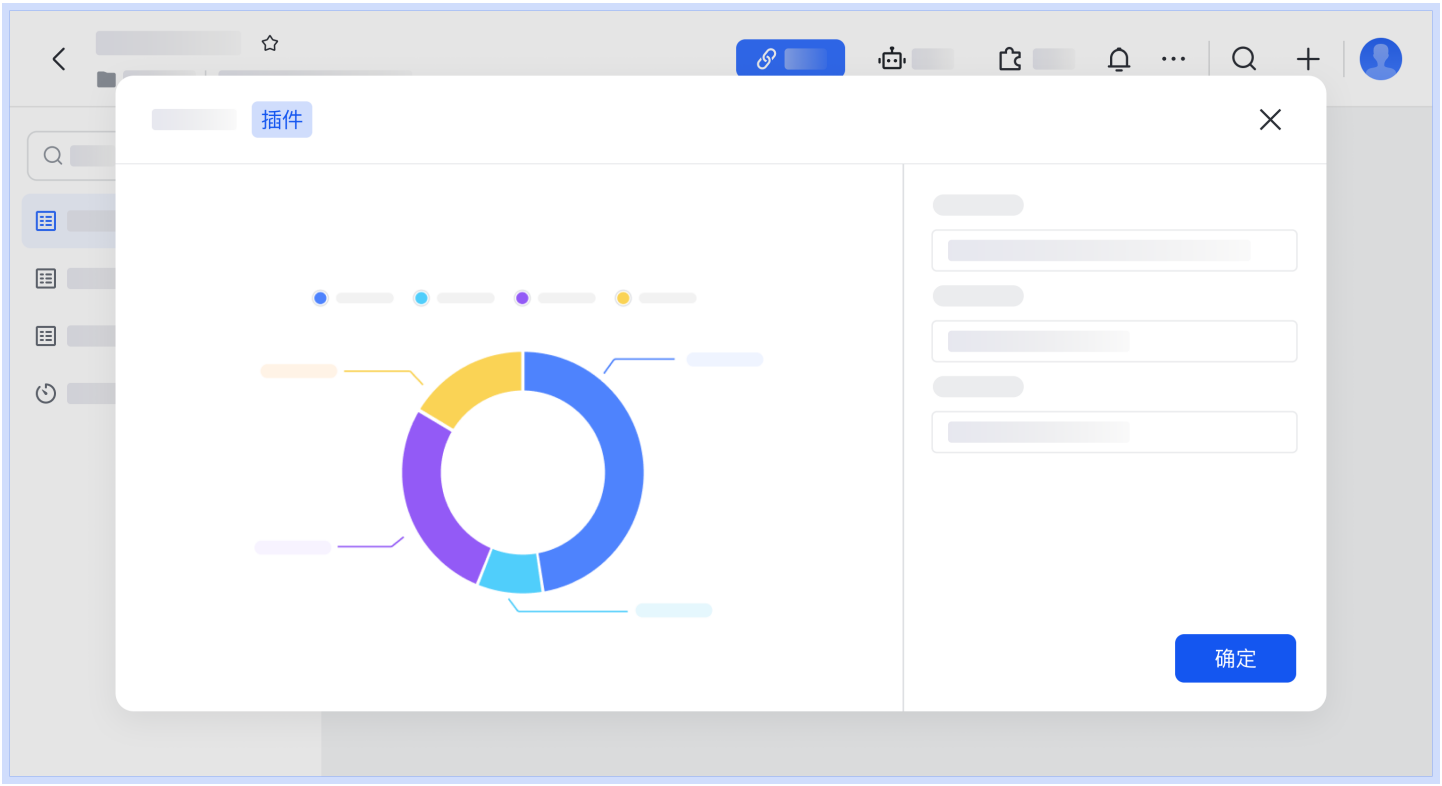
创建状态

图表首次添加至仪表盘时，插件处于创建状态，创建状态下插件相关数据并未落库。如需读取多维表格数据，仅可调用 getPreviewData 完成数据配置，无法调用 getConfig 和 getData 方法。除了初始化逻辑存在差异，创建状态和配置状态逻辑基本一致。



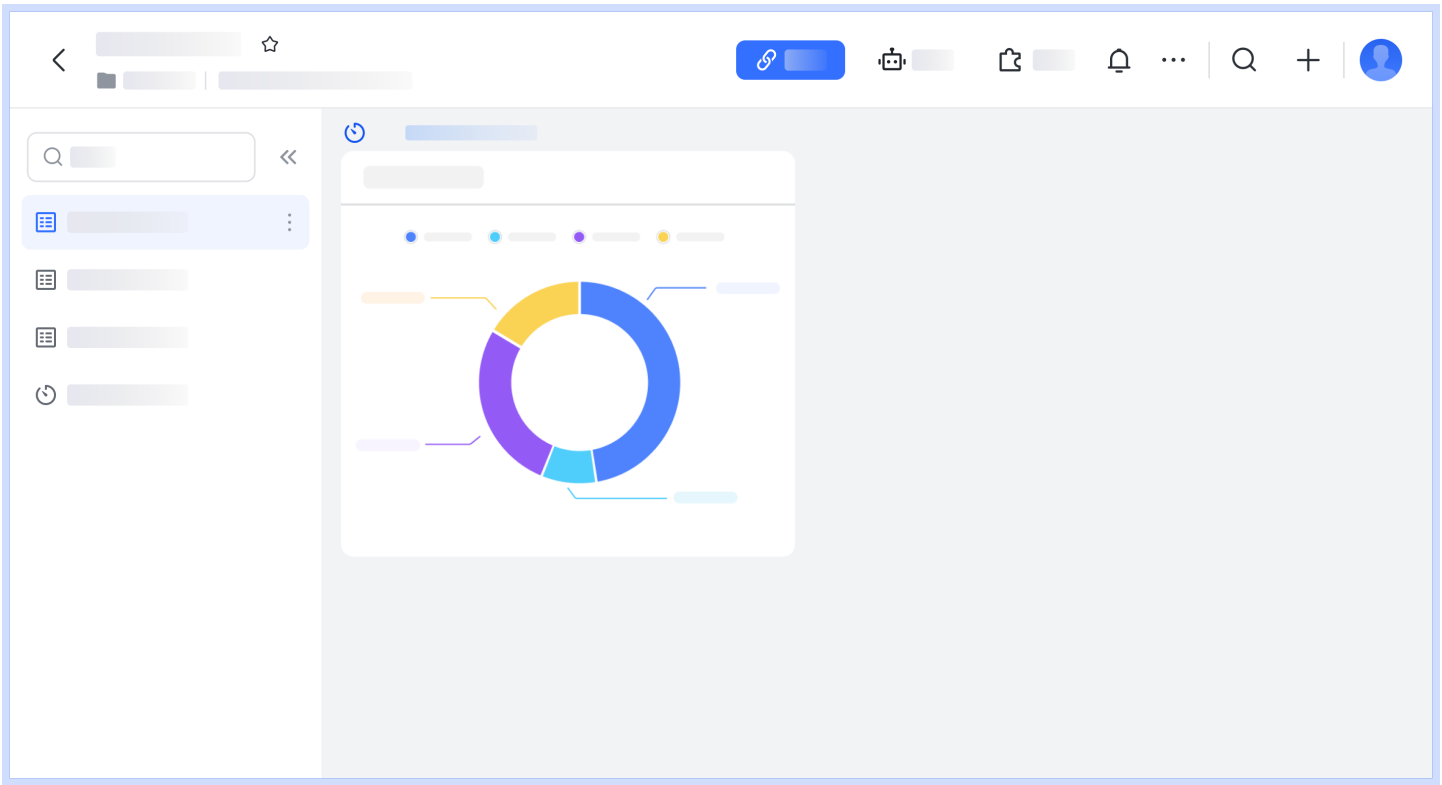
配置状态

在配置状态下用户可以对右侧配置进行修改，插件左侧图表需要跟随右侧配置的变更实时渲染。如需读取多维表格数据，可通过 getPreviewData 接口进行获取。



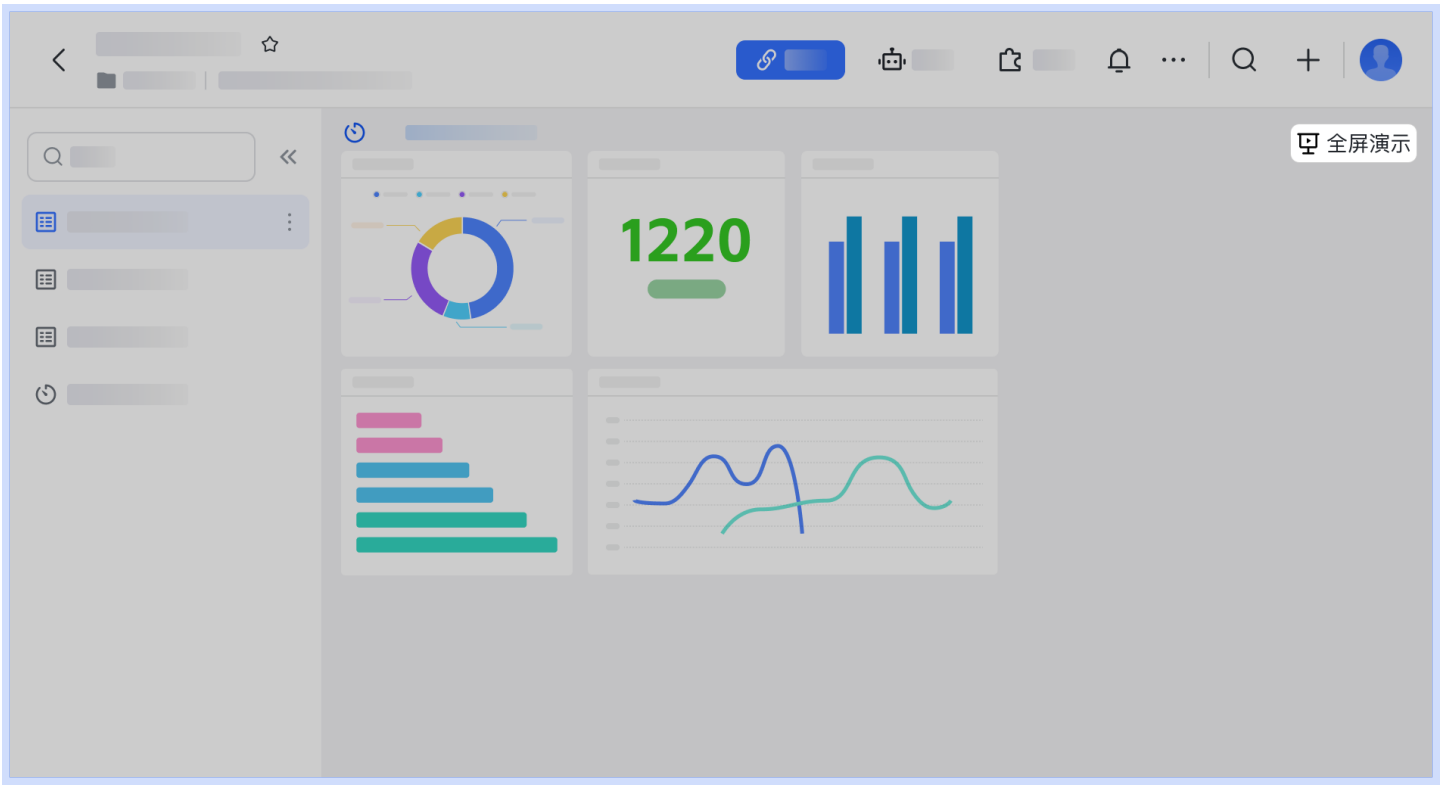
展示状态

展示状态下插件需要隐藏配置相关内容，仅渲染图表相关内容即可。同时展示状态下插件的渲染数据需要通过 getData 接口进行获取，配置和展示这两个状态是互斥的。



全屏状态

仪表盘具有特殊的全屏展示模式，插件需要感知并切换至深色模式，外层容器兼容无需开发者处理。



发布到插件中心

完成插件开发后，你可以将其发布到插件中心，以供所有多维表格用户使用。插件发布到插件中心后，将由官方托管部署。在此之前你需要对插件的基本信息进行补充，我们对每个元素的价值及要求进行了说明，并提供了示例，以帮助你顺利完成发布前的准备。

发布表单：发布到插件中心

Check list

在发布到市场前，请检查插件功能是否完备，及是否良好兼容仪表盘功能。

- 初始化时遍历表结构自动填入适合插件的表和字段
- 拖拽改变容器尺寸时插件自适应良好
- 配置状态下修改表单预览区实时渲染
- 通过 onDataChange 监听多维表格数据变化
- 正确兼容全屏模式
- 自动化推送时可被正确截图
- 字段选择菜单提供搜索能力
- 字段选择菜单通过 icon 标记字段类型
- 正确通过 dataConditions 参数存储和读取表结构保证创建副本时插件可正确获取配置

至少提交以下信息就可以将插件发布到市场，但更加完善的信息有助于插件被更多用户使用。

- 插件名
- 项目代码地址
- 简短描述
- 类别
- 使用录屏

简短描述

用户在浏览插件中心时会看到卡片上的简短描述，使用尽可能精简的语句描述该插件的功能及价值。推荐使用主动动词（如添加、实施、创建、更新、可视化等）撰写基于动作的描述。

- 必要项
- 最多 X 个字符
- *示例：按照一定条件查找重复的记录，并删除它们。*

详细介绍

在插件介绍页展示，它应该具体阐释插件的功能，通过步骤介绍如何使用插件，以及出现使用问题时该如何寻求帮助，确保用户对插件有完整的了解。

- 非必要项，如开发者无法提供，我们的运营人员将通过 AI 为其生成
- 200 至 2,000 个字符
- 建议使用换行符或项目符号列表令版式更为美观
- 支持通过 Markdown 编辑器生成 Markdown 语句

我们建议遵循此结构：

第 1 段：突出显示插件的主要功能、解决的问题以及核心优势，确保用户仅用一段文字就能理解插件的功能。

第 2 段：分享更多用例并提供有关插件的更多背景信息。

第 3 段：提供一个用户可以寻求帮助的路径，例如帮助文档链接或联系方式。

类别

插件中心允许用户根据类别筛选插件，从以下列表中选择插件所属的类别：

- 必要项
- 最多选择三个

效率

图表

如果没找到适合的分类可通过 交流群 向我们反馈。

图标

使用图形语言尽可能的传达插件功能，避免出现复杂细节影响可识别性，并确保不存在版权风险，我们提供了 remixicon 和 iconpark 两套开源图标库可供使用。你可以使用此 模板 创建自己的图标，选择图标背景与元素的颜色搭配，并调整元素大小保持在框架内。

- 必要项，如开发者无法提供，将由我们的运营人员代为生成
- 推荐 SVG 格式
- 或 128 像素 x 128 像素 JPG / PNG

色板

介绍图片

通过若干静态图片来突出插件的主要特征、界面、品牌和标识。这些图片应将裁剪后的、重点突出的界面与简短文字说明结合起来。

- 非必要项
- 推荐 SVG 格式
- 或 1920 像素 x 960 像素 JPG / PNG

在视觉上突出插件的功能和特性，而不是简单地截图。建议使用彩色背景，以确保图像在所有主题（包括深色模式）中脱颖而出。每张图片都应侧重于介绍插件的一个功能点，使用户感受到价值。

介绍视频

使用此视频演示插件的特性、功能和用户界面，以帮助用户快速了解如何操作使用该插件。

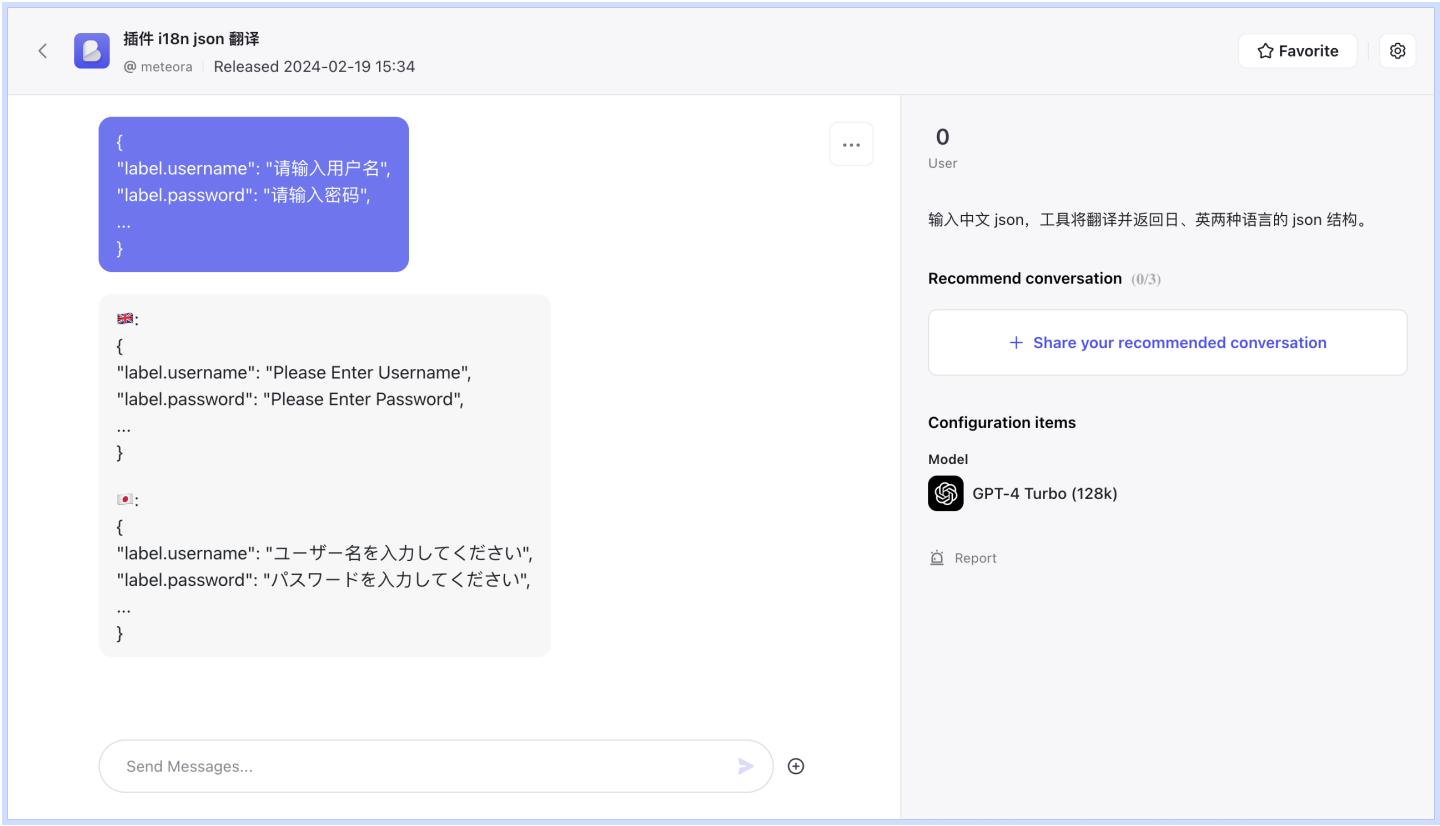
- 必要项
- 不超过 20 秒

- MP4 或 GIF 图

推荐使用桌面端即时消息截图工具，使用快捷键 **Alt** + **Shift** + **R**（Windows）或 **Option** + **Shift** + **R**（Mac），框选录屏区域。 或是鼠标悬浮于 **截图** 按钮，选择 **录屏**，框选录屏区域，选择 **MP4** 或者 **GIF** 格式，点击 **开始录制** 即可。

国际化

由于国际化和市场团队的要求，发布到市场的插件必须通过 i18n 能力支持中、日、英三种语言。请务必使用 插件 i18n json 翻译 工具完成国际化，以确保专有名词的正确性。输入中文 json，工具将翻译并返回日、英两种语言的 json 结构。



示例：

./locales/zh.json

代码块

```
{
"label.username": "请输入用户名",
"label.password": "请输入密码",
...
}
```

./locales/en.json

代码块

```
{
"label.username": "Please enter your username",
"label.password": "Please enter your password",
...
}
```

./locales/jp.json

代码块

```
{
"label.username": "ユーザー名を入力してください",
"label.password": "パスワードを入力してください",
...
}
```

./i18n.ts

代码块

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';

import translationEN from './locales/en.json';
import translationZH from './locales/zh.json';
import translationZH from './locales/jp.json';

// 设置支持的语言列表
const supportedLanguages = ['en', 'zh','jp'];


export function initI18n(lang:'en'|'zh'|'jp'){
  // 初始化 i18n
i18n.use(initReactI18next).init({
  resources: {
    en: {
      translation: translationEN,
    },
    zh: {
      translation: translationZH,
    },
  },
  lng: lang, // 设置默认语言
  fallbackLng: 'en', // 如果没有对应的语言文件，则使用默认语言
  interpolation: {
    escapeValue: false, // 不进行 HTML 转义
```

```
27     },
28   });
29
30   }
```

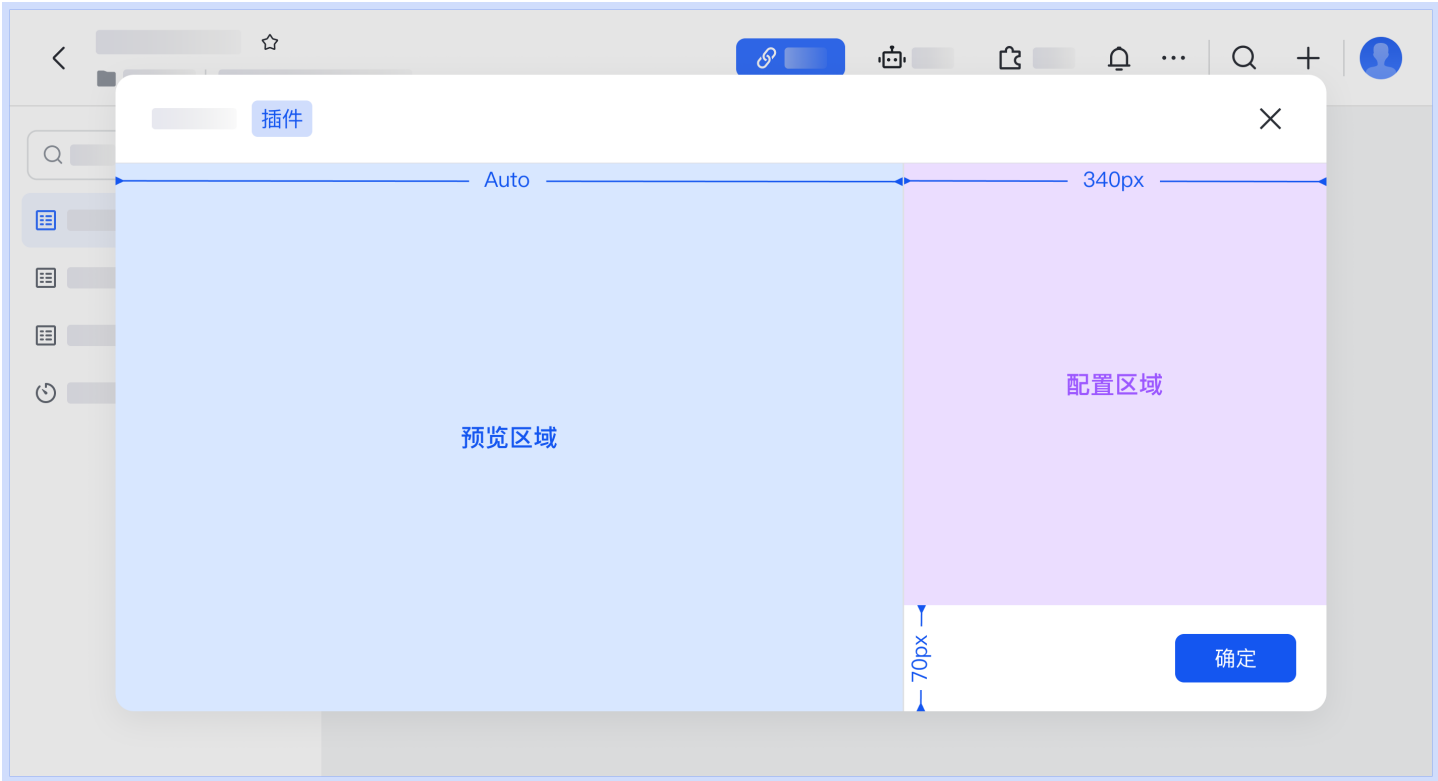
UI & 交互

在插件审核时，我们会确保一些基础的设计规范符合标准，如果想进一步提高插件的品质，可以参阅 Base 开放设计规范。

布局

仪表盘插件配置状态整体由开发者自由搭建，但需符合官方 UI 规范：

- 左右结构布局，右侧为配置表单，左侧为渲染区
- 配置表单宽度固定 340px，渲染区域自适应
- 配置表单需设置 70px 底部高度为确认按钮预留空间，避免遮挡
- 当仪表盘插件进入配置状态时，必须做到配置改变时，预览图实时同步刷新渲染



示例：

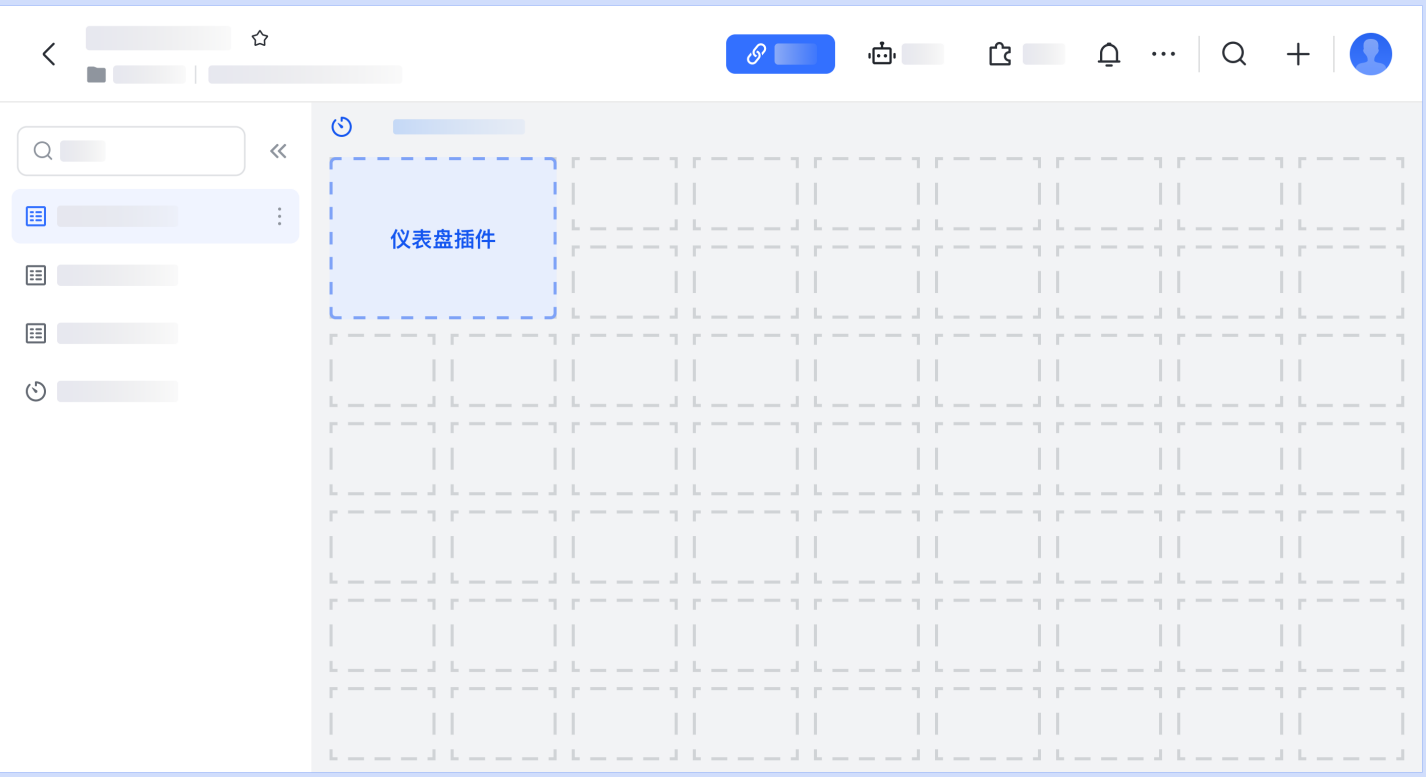
```html
<div class="config-widget-dialog-container">
    <div class="config-widget-dialog-preview"></div>
    <div class="config-widget-dialog-settings"></div>
</div>
```

代码块

```css
.config-widget-dialog-container{
    display: flex;
    height: 100%;
    position: relative;
    ...
}

.config-widget-dialog-preview{
    position: relative;
    ...
}

.config-widget-dialog-settings{
    border-left: 1px solid #5f5f5f;
    flex: 0 340px;
    height: 100%;
    padding-bottom: 70px;
    position: relative;
    ...
}
```



仪表盘被分隔为若干网格，插件可在仪表盘内自由拖拽改变尺寸，最小需占据 2*2 格。因此开发者需兼容各尺寸下的展示状态，必要时可以在小容器尺寸下隐藏部分页面元素，且 2*2 和 3*3 格插件的展示表现，也决定了他们在移动端呈现的样子。页面元素尺寸及字号，都应合理使用动态单位（如 %、vw、vh、vmin、vmax 等）以确保适应容器弹缩，以达到良好的展示效果。

表单

- 使用 Select 组件时，无论是选择字段，还是选择字段聚合，都应提供搜索能力。
- 当调用接口拉取字段列表时，请使用 SDK 中 View 模块的接口，以保证列表有序返回，避免使用 Table 模块来拉取字段列表，从而造成列表无序返回。

字体

优先使用系统默认的界面字体，同时提供一套备用字体库，来维护在不同平台以及浏览器的显示下，字体始终保持良好的易读性和可读性，建议开发者同样使用这套字体规则以保证兼容性。

代码块

```
//中英文环境
font-family:-apple-system,BlinkMacSystemFont,Helvetica Neue,Tahoma,PingFang SC,Microsoft Yahei,Arial,Hiragino Sans GB,sans-serif,Apple Color Emoji,Segoe UI Emoji,Segoe UI Symbol,Noto Color Emoji;
//日文环境
font-family:"ヒラギノ角ゴ Pro W3", "Hiragino Kaku Gothic Pro", "Yu Gothic UI", "游ゴシック体", "Noto Sans Japanese",“Microsoft Jhenghei UI”,“Microsoft Yahei UI”,"ＭＳ Ｐゴシック", Arial, sans-serif,Apple Color Emoji,Segoe UI Emoji,Segoe UI Symbol,Noto Color Emoji;
```

主题色兼容

多维表格支持切换「浅色（light mode）」和「深色（dark mode）」两种外观模式，因此插件在视觉上也需要进行兼容。插件的 iframe 容器天然兼容两种主题色，因此开发者无需额外设置插件内元素的背景色，只需要关注元素本身的颜色即可。

统一 UI 库

为了保持插件的 UI 一致性，我们基于 semi 定制了一套 UI 样式。

安装依赖包

代码块

```
npm i @douyinfe/semi-ui @semi-bot/semi-theme-feishu-dashboard vite-plugin-semi-theming @douyinfe/semi-foundation
```

在vite项目中接入

然后修改 vite.config.js

代码块

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import { semiTheming } from "vite-plugin-semi-theming";

// https://vitejs.dev/config/
export default defineConfig({
    base: "./",
    plugins: [
        react(),
        semiTheming({
            theme: "@semi-bot/semi-theme-feishu-dashboard",
        }),
    ],
    server: {
        host: "0.0.0.0",
    },
});

```

在webpack项目中接入

安装webpack插件

代码块

```
npm i @douyinfe/semi-webpack-plugin
```

使用插件

代码块

```
// webpack.config.js
const SemiWebpackPlugin = require('@douyinfe/semi-webpack-plugin').default;
plugins: [
  new SemiWebpackPlugin({
    theme: '@semi-bot/semi-theme-feishu-dashboard',
    include: '~@semi-bot/semi-theme-feishu-dashboard/scss/local.scss'
  })
]
```

代码块

```
// next.config.js
const semi = require('@douyinfe/semi-next').default({
    /* the extension options */
});
module.exports = semi({
// your custom Next.js configuration
});
```

手动引入主题 在 global.css 中引入全量的 semi css

代码块

```
/* styles/globals.css */
@import '~@douyinfe/semi-ui/dist/css/semi.min.css'; // 当你希望使用 Semi 默认主题时，直接从 semi-ui 中引即可

@import '~@semi-bot/semi-theme-feishu-dashboard/semi.min.css';
```

在 NextJs 项目接入

当你在 Next.js 项目中使用 Semi 时，需要搭配 Semi 提供的编译插件（由于 Next.js 不允许 npm 包从 node_modules 中 import 样式文件，需要配合插件将默认的 import CSS 语句移除，并且手动引入 CSS）

配置 Semi Next Plugin

代码块

```
// next.config.js
const semi = require('@douyinfe/semi-next').default({
    /* the extension options */
});
module.exports = semi({
// your custom Next.js configuration
});
```

手动引入主题 在 global.css 中引入全量的 semi css

代码块

```
/* styles/globals.css */
@import '~@douyinfe/semi-ui/dist/css/semi.min.css'; // 当你希望使用 Semi 默认主题时，直接从 semi-ui 中引即可

@import '~@semi-bot/semi-theme-feishu-dashboard/semi.min.css';
```

代码规范

在插件审核时，我们会对代码进行 review，以规避在数据安全和性能等方面存在的隐患。

数据安全

- 为确保数据安全，除插件功能必要的 API 请求外，禁止将多维表格数据向外部发送。
- 确保插件所依赖的三方仓库为开源

部署

vite配置

纯前端的静态项目，必须包含 `vite.config.js` 文件，并一定要包含 `base: './',` 语句，否则将导致部署失败。开发者可在提交之前进行自测，在 node 16.19.0环境下运行两条命令：npm

install 和 npm run build，如果两条都能成功，则符合部署条件。

代码块

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

// https://vitejs.dev/config/
export default defineConfig({
  base: './', //请勿删除此语句，否则将导致部署失败
  plugins: [react()],
  server: {
    host: '0.0.0.0',
  }
})
```

本地自测

如遇部署失败，建议先在本地测试是否可以部署

1. 本地切换 Nodejs 版本为 `v16.19.0`， npm 版本为 `8.19.3`
2. 执行 `rm -rf node_modules` 删除本地依赖
3. 执行 `npm install` 重新下载依赖
4. 执行 `npm run build` 验证本地构建

指定构建产物目录

在 `package.json` 中设置 `output` 属性可以跳过打包过程，直接上传 `output` 指定的目录

代码块

```
{
  "output": "dist" // 指定直接上传 dist 目录了
}
```

常见问题