**云组态组件开发文档**

**目录**

[1、概述 2](#_Toc51849288)

[1.1文档目的 2](#_Toc51849289)

[1.2系统概述 2](#_Toc51849290)

[1.3运行环境 2](#_Toc51849291)

[2、开发说明 3](#_Toc51849292)

[2.1目录结构介绍 3](#_Toc51849293)

[2.2组件开发 5](#_Toc51849294)

[2.2.1准备工作 5](#_Toc51849295)

[2.2.2 template介绍 8](#_Toc51849296)

[2.2.3 data介绍 8](#_Toc51849297)

[**2.2.4 script介绍** 10](#_Toc51849298)

[2.3新属性开发 12](#_Toc51849299)

[2.4通用类方法 14](#_Toc51849300)

[2.5注意事项 14](#_Toc51849301)

[编辑态&运行态 14](#_Toc51849302)

1、概述

## 1.1文档目的

本文档用于介绍云组态的组件开发流程，主要适用对象为技术工程师。

## 1.2系统概述

本系统名称为正泰可视化云组态平台。在项目中，我们可以开发各类操作页面，来接收硬件传来的数据，并对数据做分析和计算，同时控制硬件，可以理解为组态阶段是对智能控制系统的快速拼装，运行态即得到完成的、可供客户进行使用和操作的智能系统。

目前项目以Vue为核心框架，Vuex为状态管理器，已支持的依赖还包括less，element-ui，echarts，可直接使用。如需有其他第三方库的支持，可根据需求引入。

该软件系统数据位号的修改和管理，在数据中心平台上完成。

## 1.3运行环境

项目以使用谷歌浏览器为主，其他浏览器暂不支持。

2、开发说明

## 2.1目录结构介绍

项目

└─ src

│ └─ api

│ └─ until

│ │ └─ common.js

│ └─ component

│ │ └─ base

│ │ │ └─ text.vue

│ │ …

│ └─ components

│ │ └─ differentClass

│ │ │ └─ pack

│ │ └─ differentClassJson

…

\* 此处仅列出关于组件开发中，必要了解的目录文件。

**src / api / :** 存放项目中所需的接口请求，新增接口也需在此统一管理；

**src / until / common.js:** 实现了组件开发中，通过类型值type，来加载不同组件的操作；

**src / component:** 主要存放组件的vue文件，按组件类型进行分类文件夹，如文件夹base为基本类组件、文件夹high为高级组件等，其文件夹下存放的为相对应类别的组件，如基本类的文本组件 . / base / text.vue；

**src / components:** 存放了关于主页的头部、左右侧菜单、属性栏等多种组件与配置文件；

**src / components / differentClass / pack:** 存放属性栏中的属性组件，通过组件的配置，来加载显示对应的属性组件，如图2-1；



图2-1

**src / components / differentClass / pack / pack.vue:** 逻辑处理文件，新增属性，需要在其中引用，并根据type做挂载操作；

**src / components / differentClass / pack / common.js:** 存放属性组件中通用的配置信息，如时间类型、字体可选大小等；

**src / components / differentClassJson:** 按组件类型存放相应类型的组件列表配置，用于页面的列表组件显示，如图2-2。



图2-2

## 2.2组件开发

### 2.2.1准备工作

以开发【文本】组件为例，共分两步：

**第一步**，组件信息注册。

文本属于基本类型，进入src / components / differentClassJson目录，打开base.json 基本类列表配置文件，添加文本组件的配置信息，如下

{

    "width": 80,

    "height": 20,

    "type": "text",

    "classes": 0,

    "name": "文本",

    "imUrl": "iconwenben"

},

其中对应意义如下：

"width": 组件初始宽度

"height": 组件初始高度

"type": 组件类型--唯一，请设置为具有组件意义的名称，作为common.js挂载组件时的依据

"classes": 类别，每个json都有一致的classes值，参照即可

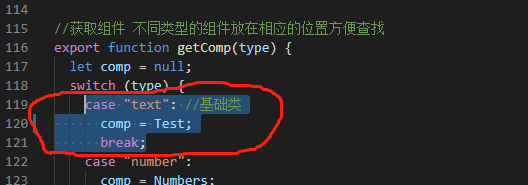
"name": 组件名称，该组件在右侧列表中的文字描述

"imUrl": 组件预览图，该组件在右侧列表中的图片描述

进入src / util / common.js，添加vue组件路径

// 基础

import Test from "@/component/base/test.vue";



**第二步，**编写单个组件vue文件

进入src / component / base目录，新建text.vue（与上方type一致）

打开text.vue，我们准备了模板文件test.vue，可复制src/component/base/test.vue代码到text.vue中。

模板文件内容如下：

<template>

  <div class="symbolComp" :style="styles">我是demo</div>

</template>

<script>

//必要---保持一致：配置转换器，命名规范：test.vue test.js

import translateConfig from "./test.js";

import mixin from "@/component/mixin.js";

export default {

  mixins: [mixin],

  methods: {

    toSearch(val) {

      if (this.$store.state.runtime) {

        console.log(val);

        //val是一个对象，是日期选择器返回的对象

      }

    },

    getData() {

      if (this.$store.state.runtime) {

        let treeId = this.highConfig["树形图"].value;

        if (treeId) {

          console.log(this.$drag.getTreeData(treeId));

          //获取当前绑定的树组件数据

        }

      }

    },

    //配置转换：必要---保持一致：读取南京配置之后，对应的配置要转换成我们自身的配置，该方法是入口，会传入南京这个组件的配置

    translateConfig(item) {

      return translateConfig(item);

    },

    //组件初始化调用，在拿到最新数据之后，在这里设置高度与字体大小初始比例

    init() {

      console.log("init");

    },

    //resetComp（bool）高度变化，字体跟着变化，可以进行样式属性重新赋值

    //bool：false----高度一直在变化，这个时候不能进行图表重绘

    //bool：true---高度最后一次变化，这个时候应该进行图表重绘

    resetComp(bool) {

      console.log(bool);

    },

    //必要：当配置属性更改之后，组件需要做响应的动作，

    //@param {Bollean} bool 配置更改触发initStyle，会执行this.$parent.initStyle方法，其余都不会

    //bool: true表示是右侧属性变化触发的initStyle事件,不应该手动调用这个方法

    //type:变化源,那个基础属性变化触发了这个方法，需要在属性vue中使用：onChange(this.title)

    initStyle(bool, type) {

      console.log(type);

      let { theme, baisicConfig } = this.startInitStyle();

      //这里进行一些当前组件属性的转换与使用

      this.styles = {

        color: theme.color

      };

      console.log(baisicConfig);

      this.endInitStyle(bool);

    }

  },

  data() {

    return {

      styles: {}, //最终应用到template上面，内容由initStyle负责

      white: {

        //由于测试组件只有一个样式，将其提升为变量color

        字体样式0: "#000000"

      },

      black: {

        字体样式0: "#FFFFFF"

      },

      //高级属性，对于已有的属性：可见，闪烁，颜色切换，颜色响应，

      //以及基础事件（切换页面,绑定容器等）这些功能均已实现，运行态功能也已实现

      //如果发现新的高级属性有这样的功能：测点值变换，需要响应到当前组件，这个时候需要报备一下，

      //且对于这些事件的数据转换规则，我已在tranlate.js中包含（日期组件，树形图没有--没有对应的实例）

      highConfig: {},

      baisicConfig: {}

    };

  }

};

</script>

### 2.2.2 template介绍

每个组件的最顶层div，都有样式symbolComp，以及默认的样式，它的作用是保证你的组件高度和宽度都是100%，对于当前组件中可能存在的动态设置样式，比如字体样式、背景样式、透明度等等，可使用style属性绑定了当前组件data中的styles，组件的其他html结构，由组件各自的功能而定。

### 2.2.3 data介绍

每个组件都应该有white、black，（baisicConfig，highConfig可选），目前仅支持white和black两种主题色，通过定义baisicConfig和highConfig来显示页面上的基本属性与高级属性，两种属性在代码方面的处理逻辑基本一致，基本属性侧重样式、界面的配置，而高级属性侧重于数据方面的配置。

#### **baisicConfig:**

组件的基础属性，会根据该配置渲染成界面中的基础属性dom。

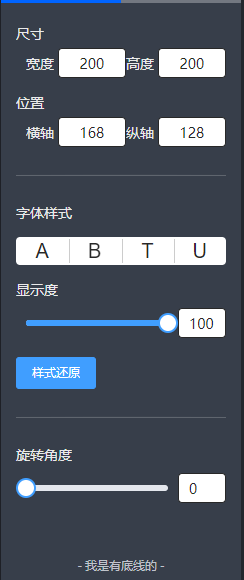
 

图2-2-3

以上为基础属性配置+在界面中展示的样式比对，开发者开发组件时，可先前往src / components / differentClass / pack 中了解现有的属性组件，这里拿出【旋转角度属性】简单描述一下：

旋转角度属性: 这里用到了type为slider的属性组件，所有的进度条类的属性，都可以使用type：slider来实现（项目中已集结了大量可复用的属性，他们都有对应的type和属性，开发者只需要关注配置即可）。

**样式还原:** type为resetStyle，此属性较为特殊，特别讲解，还原逻辑是在config中以此属性为基点，向上和向下各找一个最近的一个type为line的属性，重置还原这个区间内的属性值，如上图2-2-3中，将还原属性 “线条” 和 “线条1” 之间的所有属性。

#### **highConfig**:****

组件的高级属性，会根据该配置渲染成界面中的高级属性dom。

每种属性都有自身单独的type，这个type很重要，对应界面中的dom，也会反应当前组件该有那些独特的动作，如可见，闪烁，事件，颜色切换等等。

每个属性当被配置好了之后，当用户去操作界面属性的时候，一旦属性发生变化，就会触发对应的组件的方法initStyle中，开发者只需要在该方法中重置对应的属性即可。

#### White，black:

当前系统有两种主题：黑色和白色，这两个属性会收集不同主题下不同的属性配置，且会跟随主题切换，让组件响应不同的主题配置，开发者只需要编写对应状态下属性的配置（key：value）即可。

white: {

        //由于测试组件只有一个样式，将其提升为变量color

        字体样式2: "#000000"，

        背景颜色: "#fff"

},

black: {

        字体样式2: "#FFFFFF"，

        背景颜色: "#000"

}

White与black，收集根据主题变化的属性，一般都是颜色

注意命名: 当颜色值存储在数组中时， 如key名为“字体样式”的属性，其value为[“16px”, “bold”, “#000” ]，【字体样式2】，命名必须是属性中的key加上属性value中对应的位置，这样框架才能知道切换主题的时候需要去改变哪个属性的哪个值，如果value不是数组，只需要key即可。

参考：

white: {

        表格标题字体2: "rgba(0,0,0,1)",

        单元格字体2: "#67757C",

        背景颜色: "#FFFFFF",

        边框颜色: "rgba(221, 221, 221,1)"

},

black: {

        表格标题字体2: "#FFFFFF",

        单元格字体2: "#FFFFFF",

        背景颜色: "#18191D",

        边框颜色: "rgba(67, 74, 90,1)"

 }

**2.2.4 script介绍**

#### Mixins:

包含默认的动作，非必要情况，开发者不需要维护也不应该修改

#### initStyle (bool, type):

组件中至关重要的方法，一般不需要手动调用，**bool**为true时，表示由右侧属性变化触发的initStyle事件； **type**一般情况不会用到，当需要监听由具体哪一个属性引起的变化时，可由属性组件中回调一个标识，从而实现判断逻辑，具体可见目录“[2.3新属性开发](#_2.3新属性开发)”中介绍。

    initStyle(bool, type) {

      console.log(type);

      let { theme, baisicConfig } = this.startInitStyle();

      //这里进行一些当前组件属性的转换与使用

      this.styles = {

        color: theme.color

      };

      console.log(baisicConfig);

      this.endInitStyle(bool);

    }

可以看到initStyle默认调用了

let { theme, baisicConfig } = this.startInitStyle();

this.endInitStyle(bool);

开发者必须也这样做

包含的代码就是你组件变量的赋值，比如styles，比如title等等，详细可参考附件中别的组件文件内容。

#### toSearch(val):

如果组件可以绑定日期选择器，则组件应该有这么一个方法，它会在用户点击日期选择器查询按钮时候触发，val是日期选择器的当前参数。

#### getData():

只需要关注以下几个针对树组件的方法：

this.$drag.getTreeData(treeId)//获取基本的pointId和staticType

this.$drag.getTreeDatas(treeId)//获取对应的name

this.$drag.getTreeDatas(treeId,true)//从根节点获取

如果该组件可以绑定树组件，则可以通过该方法获取当前树组件的选中参数，由于不同组件需要的字段不同，所以提供了不同的方法。

#### translateConfig(item):

该方法在做兼容南京上海的时候需要。

#### Init():

组件初始化会调用的方法，这里会拿到最新的组件配置，所有初始化内容都应该在这里写，而不是create和mounted，比如字体大小与高度的比率。

#### resetComp(bool):

Bool===true表示宽度高度最后一次变化，具体根据组件的需求，选择是否调用此方法，一般会在这里调用图表重绘等。

#### returnConfig()

在保存配置时，自动调用此方法，mixin中默认调用了此方法，但如果该组件需要点位收集，则必须重写该方法，其中collectPointIds为二维数组。

    returnConfig() {

      let collectPointIds = [this.highConfig["数据"].params[0].value];

      return {

        collectPointIds,

        white: this.white,

        black: this.black,

        highConfig: this.highConfig,

        baisicConfig: this.baisicConfig

      };

}

#### 点位收集:

目前项目中支持两种方式的点位收集

**第一种，**当数据更新以1秒频率，并且接口路径为 /api/RealTimeData/BulkGet时，直接参考[returnConfig()](#_returnConfig())方法。

**第二种，当数据更新不以1秒频率时，则需要在init() 中调用**[this.$drag.addShifte()](#_this.$drag.addShifte()) 方法，并且在 src / util / folieShfite.js 中添加配置信息，引入需要点位数据更新调用的api接口请求方法，在config中添加新信息，key名必须为当前组件的type，value对应如下文备注。

// folieShfite.js

import { groupGet } from "@/api/bulk.js";

const config = {

  statistics: {

    name: "能耗标签", // 自定义归类，相同接口请用同一个name

    time: 60000, // 接口更新的频率

    request: groupGet // 接口请求方法

  }

}

#### 点位监听: pointIdChange

通过pointIdChange方法，获取点位数据更新时，回传的数据，具体的数据结构，视具体的组件、点位收集时采用的方式 和 接口返回 而定。

pointIdChange(val) {

      if (val.statusCode != 200) return;

      this.highConfig.statisticInfo = {

        data: val.data.value

      };

},

## 2.3新属性开发

开发者在开发过程中，遇到现有属性组件库中不存在的属性时，则必须通过注册新属性，来实现功能，属性组件的开发须遵循高复用、低耦合的原则，提高其复用性。

**首先，**在src / components / differentClass / pack 文件夹下，新建一个后缀为.vue的属性组件；

**其次，**打开src / components / differentClass / pack / pack.vue，引入新建的属性组件，并根据类型值判断匹配，如下。

// pack.vue, template

<template v-for="(item, title) in config">

      <template v-if="item.type === 'input'">

        <packInput

          :title="title"

          :config="item"

          :key="title"

          :onChange="changes"

        ></packInput>

      </template>

</template>

// pack.vue, script

const packInput = () => import("./input.vue");

export default {

  components: {

    packInput

  },

**然后，**组件引用如下，title为组件配置中在基本属性或高级属性中定义的key名“最大值”，item为对应值，changes为组件的initStyle() 方法，开发者只需理解其意义，无需修改pack.vue中属性组件的传参配置。

最大值: {

   type: "input",

   value: 100

}

**最后，**属性组件的内部代码实现根据其自身功能开发即可，可参考其他属性的代码，值得注意的是，属性组件中传入的函数onChange，当组件需要监听是由哪一个属性变化引起自身调用initStyle(bool, type) 时，则需要在属性组件中回传一个标识位，如下述中的属性组件，title将回传给initStyle(bool, type)中的type；onChange的第二个参数，当为true，则变化将记录历史操作中，flase则不会。

// 属性组件的template

<select v-model="config.value[1]" @change="onChange(title, true)">

   <option

     v-for="item in fontFamily"

     :key="item.value"

     :label="item.label"

     :value="item.value"

   ></option>

</select>

// 属性组件的script

props: {

    title: String,

    config: Object,

    onChange: Function

},

methods: {

  change() {

// 回传title

      this.onChange(this.title, true);

  }

}

## 2.4通用类方法

#### this.$chint

全局定义的一个api，其文件位于**src / until / chint.js**，其中定义了，获取当前页面配置、当前页面id、网络请求、菜单列表、页面列表以及路由跳转等等多种方法，详细内容可查看该文件。

#### this.$drag.addShifte()

特殊点位的收集方式，第一个参数位当前组件的指向，第二个参数为接口请求参数

init() {

this.$drag.addShifte(this, {

    intervalUnit: this.baisicConfig["时间类型选择"].value,

    pointId: points[points.length - 1]

});

}

#### this.$drag.getTreeData(treeId)

通过树id获取树组件中选中节点的pointId和对应的staticType

#### this.$drag.getTreeDatas(treeId)

通过树id获取树组件中选中节点的name、pointId和对应的staticType

#### this.$drag.getTreeDatas(treeId, true)

通过树id获取树组件中选中节点以及其根节点的name、pointId和对应的staticType

## 2.5注意事项

### 编辑态&运行态

开发的组件代码必须同时支持编辑态与运行态两种模式。

this.$store.state.runtime = true 为运行态，反之编辑态，可通过这种方式判断当前的运行模式，区分业务逻辑。