组件化方案

组件定义：组件是对数据和方法的简单封装，是软件中具有相对独立功能、接口由契约指定、和语境有明显依赖关系、可独立部署、可组装的软件实体。

一个优秀的组件应该保证：

- 功能内聚
- 样式统一
- 与父元素仅通过Props通信

1. 组件化要求

- 引入使用
    1. 避免组件框架限制
    2. 避免浏览器版本兼容性限制
    3. 最好纯js实现，类似于echarts，富文本编辑器，可以直接通过js生成
    4. npm包形式发布
    5. 支持CDN形式接入
    6. 支持联邦模块(vite/webpack)
    7. 支持SSR
- 组件设计
    1. 基于单一职责原则封装的特定功能组件
    2. 减少必填的API项，尽可能多地提供默认值，降低组件的使用成本
    3. 使用文档逐步完善：快速开始，版本记录，常见问题，API配置项
    4. 样式主题设置
    5. 好看的样式案例列表
    6. 简单示例，修改设置预览效果
    7. 操作面板，配置完成导出文件

2. 组件化范围

- 基础组件
    1. 按钮
    2. 滚动条
    3. 响应式栅格布局
    4. 图标：svg，iconfont, Font Awesome=》建议统一成svg
    5. 链接
- 导航
    1. 面包屑
    2. 回到顶部
    3. 分页
    4. 顶部导航菜单
    5. 抽屉导航菜单

6. 步骤条
7. 下拉菜单
8. 折叠展开菜单
9. 选项卡（横向纵向）
10. 时间轴

- 表单

1. 下拉框：单选多选，级联
2. 输入框
3. 数字输入
4. 单选
5. 多选
6. 评分星星
7. 时间选择：日期，年选择，月选择，季度选择，周选择，时分秒选择，时间范围选择
8. 穿梭框
9. 文件上传
10. 颜色选择
11. 表单校验
12. 范围滑动条
13. 开关
14. 富文本
15. 搜索自动补全

- 数据展示

1. echarts封装
2. Peity,sparkline
3. 高德地图
4. antV系列
5. 表格：滚动，虚拟，树形展开，多选单选
6. 描述信息列表
7. 虚拟滚动
8. 进度条
9. 卡片：拖动排序
10. swiper
11. 翻牌器

- 反馈组件

1. 对话框
2. 抽屉
3. 加载
4. 提示框
5. Tooltip
6. 弹出框
7. 气泡确认
8. 通知
9. 空数据
10. 结果页面

- 媒体
    1. 图片：双击放大预览，移动，加载错误图
    2. 视频播放器：视频流，监控视频

3. 代码规范化

代码检查

- Typescript 类型检查

`tsconfig.json`

```
{
  "compilerOptions": {
    "target": "ESNext",
    "jsx": "preserve",
    "module": "ESNext",
    "moduleResolution": "Node",
    "allowJs": false,
    "checkJs": false,
    "declaration": true,
    "sourceMap": true,
    "outDir": "dist",
    "allowSyntheticDefaultImports": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "noImplicitAny": false,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true,
    "noUncheckedSideEffectImports": true,
    "skipLibCheck": true
  },
  "include": ["packages/**/*.ts", "@types/*.d.ts"  ],
  "exclude": ["docs/**/*.ts", "examples/**/*/ts","tests/**/*.test.ts"],
}
```

- eslint js代码规范
  `eslint.config.js`

```
import globals from 'globals';
import pluginJs from '@eslint/js';
import tseslint from 'typescript-eslint';

/** @type {import('eslint').Linter.Config[]} */
export default [
  pluginJs.configs.recommended,
```

```
  ...tseslint.configs.recommended,
  {
    files: ['packages/**/*.ts'],
    rules: {
      '@typescript-eslint/no-explicit-any': ['off']
    },
    languageOptions: {globals: globals.browser}
  }
];
```

- `.prettierrc` 代码格式配置

```
{
  "eslintIntegration": true,
  //单行代码120个字符内
  "printWidth": 120,
  //缩进空格2个
  "tabWidth": 2,
  //单引号
  "singleQuote": true,
  //分号开启
  "semi": true,
  //尾部没有逗号
  "trailingComma": "none",
  //括号没有空格
   "bracketSpacing": false,
   //箭头函数有参数有括号
    "arrowParens": "always",
}
```

- commitlint+husky git提交规范

`commitlint.config.js`

```
export default { extends: ['@commitlint/config-conventional'] };
```

`.husky/commit-msg` 提交信息格式检查

```
pnpm exec commitlint --config commitlint.config.js --edit  "${1}"
```

`.husky/pre-commit` 提交前代码检查

```
pnpm exec eslint packages/
```

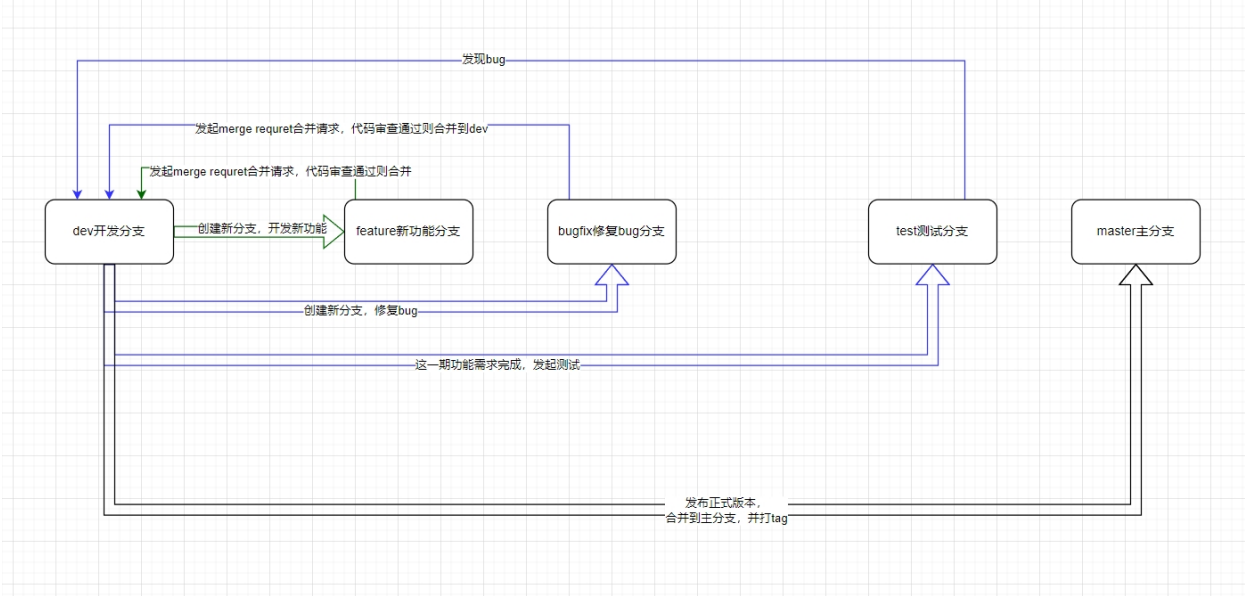
在执行`git commit -m "feat:message"`的时候，自动执行`eslint``commitlint`检查，通过检查才能提交到git

代码提交规范：

```
<type>(<scope>): <subject>
类型(模块范围): 内容
```

其中type类型有一下可选：

- feat：新功能（feature）
- fix：修补bug
- docs：文档（documentation）
- style： 格式（不影响代码运行的变动）
- refactor：重构（即不是新增功能，也不是修改bug的代码变动）
- test：增加测试
- chore：构建过程或辅助工具的变动
- git分支管理规范：所有操作从dev分支出发，最终又回归dev操作



发布版本号：v1.2.3

1. 主版本号：做了不兼容的 API 修改，
2. 次版本号：做了向下兼容的功能性新增，
3. 修订号：做了向下兼容的问题修正。

js/ts编写规范

- 单行代码不超过120个字符，js文件代码行数控制在300以内，适当分文件
- 变量名、属性名及函数名的命名必须遵循 lowerCamelCase（小骆驼拼写法）
- class类的命名必须遵循 UpperCamelCase (Pascal)，即大骆驼拼写法（帕斯卡拼写法）
- 优先使用const声明变量，且一行不能同时声明多个变量。
- 采用ES Module方式 import的方式使用

- 代码紧凑，避免没必要的空格、分号和句号
- 优先使用常见单词进行命名，如：value、visible、size、disabled、label、type等等
- on/offXXX：命名监听/注销事件
- 单独维护类型文件，并将其打包至组件产物包中，这样使用者在开发过程中能够实时看到对应的类型提示
- 为API，类型编写注释；
- 内容复杂、定制化程度高的组件使用Slot插槽方式

css样式编写规范

- 避免样式命名冲突的问题，组件统一前缀sw（水务的拼音字母）
- 控件名，如按钮用 `.sw-button`
- 内部子组件，如按钮内图标 `.sw-button-icon`
- 控件状态架类名，如不可使用的按钮 `.sw-button.disabled`
- 使用scss加var变量的形式，将颜色和大小相关抽离，便于做主题样式

4. 实现思路

采用monorepo的方式管理项目

monorepo：所有项目在一个版本库，可以实现代码共享、工具链统一、减少版本冲突等问题。

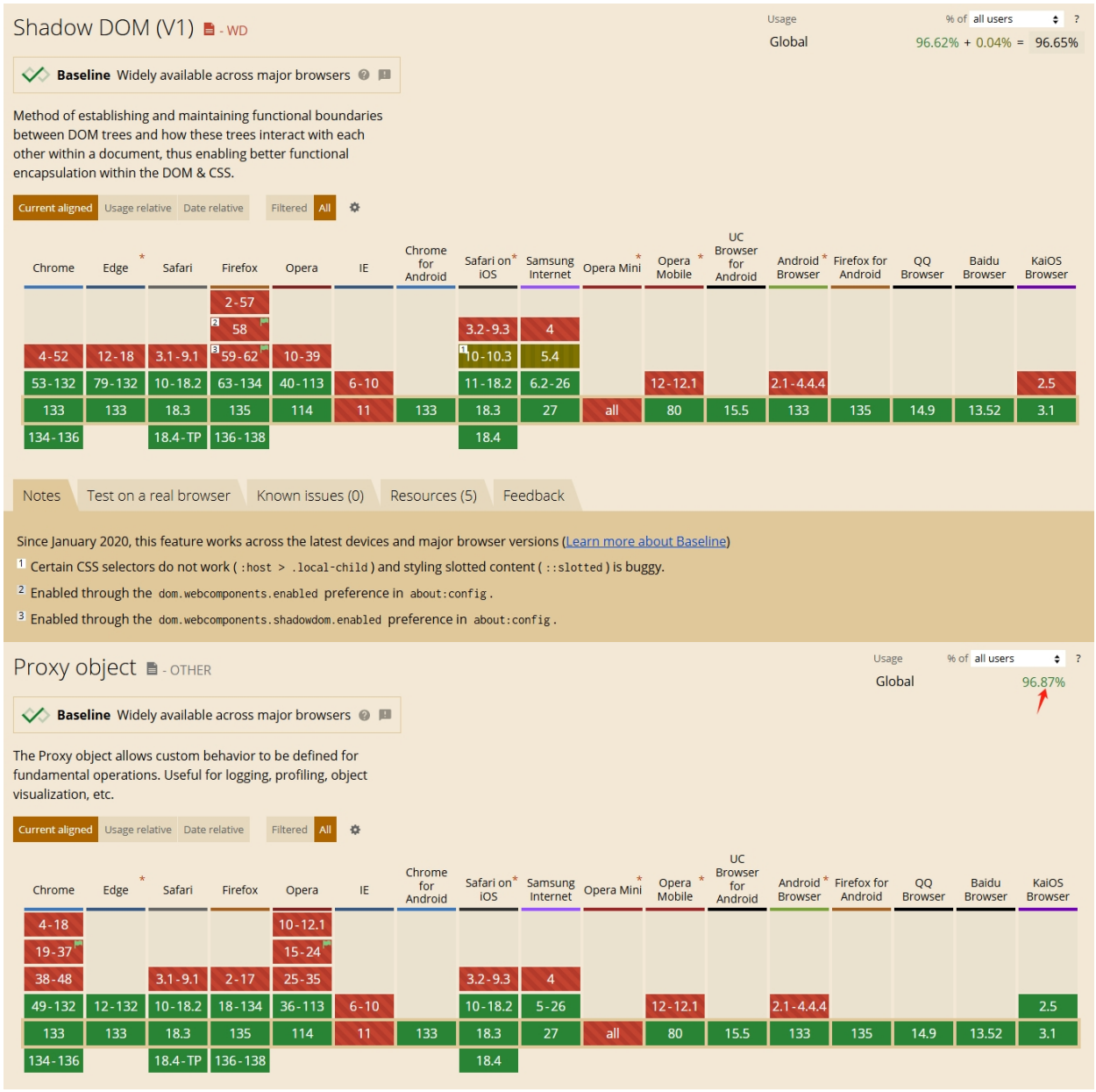
目录结构

- `examples` 组件使用示例
  - `vuedemos` vue使用示例
  - `reactdemos` react使用示例
  - `angulardemos` angular使用示例
- `docs` 文档
  - `apis` 组件配置API
  - `guide` 使用指南,版本记录，常见问题
  - package.json
- `packages` 组件库
  - `components` class组件
  - `webcomponents` webcomponent组件
  - `types` 公共类型
  - `utils` 公共工具
  - `configs` 公共配置参数
  - `index.ts` 所有组件入口
- `@types` typescript类型
- `tests` 组件测试
- `scripts` 构建脚本

    - `createIndex.js` 生成clazz组件导出文件 `index.ts`
    - `createIndex1.js` 生成webcomponent组件导出文件 `index.ts`
    - `createApi.js` 根据组件属性和方法自动生成配置API文档，用于 `vitepress`
    - `createMd.js` 生成API文件Markdown路径数组，用于 `vitepress`
    - `createDts.js` 打包所有类型和接口文件成 `d.ts`
    - `createStyle.js` 打包所有scss文件成一个 `style.css`
    - `vite.config.ts` , `rollup.config.js` 打包clazz文件
    - `vite.config.webcomponent.ts` , `rollup.config.webcomponent.js` 打包clazz文件
    - `vitest.config.ts``vitest` 组件测试
- `README.md` 记录相关开发注意事项
- `package.json`

`WebComponent Shadow DOM` 的形式组件实现

目前支持 `Shadow DOM` 的浏览器使用比例为 `96.65%` ，与Vue3用到的 `Proxy` 占比 `96.87%` 相近，代表大部分浏览器使用起来兼容性方面都没问题！



另外如果有兼容性问题可以使用官方的PollyFill 解决

- @webcomponents/webcomponentsjs

实现逻辑

1. `connectedCallback` 挂载后，在shadowDOM内创建一个DOM实例。
2. 利用 `getAttribute` 方法获取shadowDOM上的属性值并渲染DOM实例的内容，由于属性都是字符串，需要对其进行转换。
3. 给shadowDOM添加MutationObserver监听其属性改变，更新渲染。
4. 如果有需要，给DOM实例添加动作监听并处理。
5. `disconnectedCallback` 销毁前注销相关变量，避免内存泄漏。

```
const attrsMap: AttrsConfig = {
  color: 'arr',
  data: 'arr',
  title: 'str',
  legend: 'obj',
  grid: 'obj',
  dataZoom: 'obj',
  xAxis: 'obj',
  yAxis: 'obj',
  dataProps: 'obj',
  tooltipFormatter: 'obj',
  series: 'arr',
  tooltipType: 'str',
  loop: 'bool',
  time: 'num',
  titleStyle: 'obj',
  toolTipStyle: 'obj',
  seriesLine: 'obj',
  seriesBar: 'obj'
};
/**
 * @description 折线条形图 webcomponent
 */
class LineBarChart extends HTMLElement {
  chartDom: HTMLDivElement | undefined;
  interval: any = undefined;
  isHover = false;
  isLock = false;
  chart: echarts.ECharts | undefined;
  option: any | undefined;
  observer: MutationObserver | undefined;
  config: LineBarChartConfig1 | undefined;
  constructor() {
    super();
  }
  /**
   * @description 删除组件
```

```
   */
  disconnectedCallback() {
    if (this.interval) {
      clearInterval(this.interval);
    }
    if (this.chart) {
      this.chart.dispose();
    }
    if (this.observer) {
      this.observer.disconnect();
    }
    window.removeEventListener('resize', this.resize.bind(this));
  }

  /**
   * @description 组件属性变化
   */
  attributeChangedCallback() {
    this.update();
  }
  /**
   * @description 组件更新渲染
   */
  update() {
    if (!this.chart || !this.chartDom || this.isLock) return;
    this.isLock = true;
    const chartDom = this.chartDom;
    if (!this.style.width) {
      chartDom.style.width = (this.getAttribute('width') ||
this.parentElement?.offsetWidth || 300) + 'px';
    } else {
      chartDom.style.width = this.style.width;
    }
    if (!this.style.height) {
      chartDom.style.height = (this.getAttribute('height') ||
this.parentElement?.offsetHeight || 300) + 'px';
    } else {
      chartDom.style.height = this.style.height;
    }
    const config = getAttrs<LineBarChartConfig1>(this, attrsMap,
defaultConfig, LineBarChartOverwriteKeys);
    this.config = config;
    setChart(this, config, defaultConfig);
    this.isLock = false;
  }
  /**
   * @description 调整大小
   */
  resize() {
    if (this.chart) {
      this.chart.resize();
    }
```

```
  }

  /**
   * @description 创建组件
   */
 connectedCallback() {
    const chartDom = document.createElement('div');
    chartDom.style.display = 'inline-block';
    this.chartDom = chartDom;
    this.appendChild(chartDom);

    this.chart = echarts.init(this.chartDom);

    this.chartDom.addEventListener('mouseenter', () => {
      this.isHover = true;
    });
    this.chartDom.addEventListener('mouseleave', () => {
      this.isHover = false;
    });

    this.update();
    const observer = new MutationObserver(this.update.bind(this));
    this.observer = observer;
    observer.observe(this, {attributes: true});
    window.addEventListener('resize', this.resize.bind(this));
  }
}
```

vue中使用方式

```
<template>
  <div>
    <line-bar-chart
      color='["#3fb1e3","#6be6c1"]'
      ref="chartRef"
      time="2000"
      style="height: 500px; width: 500px"
      series='[{"name":"销量","type":"line"},{"name":"盈利","type":"bar"}]'
      dataProps='{"name":"name","value0":"value","value1":"value1"}'
      :data="JSON.stringify(dataRef)"
    ></line-bar-chart>
  </div>
</template>

<script setup lang="ts">
  import {ref} from 'vue';
  import '../../../build/webcomponents/index.css';
  import {install} from '../../../build/webcomponents/index';

  //引入注册组件
  install();
```

```
  const dataRef = ref([]);

  fetch('https://www.xiaolidan00.top/getRandNum.php')
    .then((res) => res.json())
    .then(({data}) => {
      console.log('🚀 ~ WebComponents ~ data:', data);
      dataRef.value = data;
    });
</script>
```

react中使用

```
import {useEffect, useState} from 'react';
import '../../../build/webcomponents/index.css';
import {install} from '../../../build/webcomponents/index';
install();

export const WebComp = () => {
  console.log('WebComp');
  const [dataList, setDataList] = useState([]);
  useEffect(() => {
    //引入注册组件
    fetch('https://www.xiaolidan00.top/getRandNum.php')
      .then((res) => res.json())
      .then(({data}) => {
        console.log('🚀 ~ WebComponents ~ data:', data);
        setDataList(data);
      });

    return () => {};
  }, []);

  return (
    <line-bar-chart
      id="chartDom"
      style={{height: '500px', width: '500px'}}
      series='[{"name":"销量","type":"line"},{"name":"盈利","type":"bar"}]'
      dataProps='{"name":"name","value0":"value","value1":"value1"}'
      data={JSON.stringify(dataList)}
    ></line-bar-chart>
  );
};
```

优势：

- webcomponent的使用方式更接近于html+js的编码模式，可以利用Vue和React虚拟DOM的方式增删改，比较容易理解。
- webcomponent组件可以字符串化，其内部会在到达浏览器时才渲染，可以支持SSR。

劣势:

- webcomponent中对象数组属性赋值都需要字符串化。
- 编写webcomponent组件编写于有一定的注意事项，需要转换平时的开发思维。
- 组件使用时属性值没有提示选项，需要文档足够完善。
- 自定义组件因为在react和vue中找不到，会报错

`Class`的形式组件实现

利用`Class`给DOM实例设置样式和操作

```
/**
 * @description 折线条形图
 */
class LineBarChart {
  el: HTMLElement | undefined;
  chart: echarts.ECharts | undefined | typeof MockEcharts;
  data: DataItemType[] = [];
  config: LineBarChartConfig | undefined;
  option: any | undefined;
  private isFirst = true;
  isHover = false;
  private interval: any;
  constructor() {}
  /**
   * @description 初始化配置样式和属性，设置数据，渲染组件
   * @param   el  DOM实例
   * @param  config  echarts图表配置
   * @param   data  数组
   */
  init(el: HTMLElement, config: LineBarChartConfig, data: DataItemType[]) {
    this.el = el;
    if (this.isFirst) {
      this.chart = isJsDom() ? MockEcharts : echarts.init(el);
      //@ts-ignore
      this.el['_chart'] = this.chart;
      this.isFirst = false;
      //事件监听
      this.el.addEventListener('mouseenter', () => {
        this.isHover = true;
      });
      this.el.addEventListener('mouseleave', () => {
        this.isHover = false;
      });
    }
```

```
    this.config = mergeConfig<LineBarChartConfig>(config, defaultConfig,
LineBarChartOverwriteKeys);
    this.data = data;
    this.render();
  }
  /**
   * @description 设置数据，渲染组件
   * @param  data  数组
   */
  setData(data: DataItemType[]) {
    this.data = data;
    this.render();
  }
  /**
   * @description 配置样式和属性，渲染组件
   * @param   config  echarts图表配置
   */
  setConfig(config: LineBarChartConfig) {
    this.config = mergeConfig<LineBarChartConfig>(config, defaultConfig,
LineBarChartOverwriteKeys);
    this.render();
  }

  /**
   * @description 渲染更新
   */
  render() {
    if (!this.config || !this.chart || !this.el) return;
    setChart(this, {...this.config, data: this.data}, defaultConfig);
  }
  /**
   * @description 调整大小
   */
  resize() {
    this.chart?.resize();
  }
  /**
   * @description 销毁
   */
  destory() {
    if (this.interval) {
      clearInterval(this.interval);
    }
    if (this.chart) this.chart.dispose();
  }
}
```

vue中使用

```
<template>
```

```
  <div>
    <div style="height: 500px; width: 500px" ref="chartRef"></div>
  </div>
</template>

<script setup lang="ts">
  import {ref, useTemplateRef, onMounted, onBeforeMount} from 'vue';
  import '../../../build/clazz/index.css';
  import SWCOMP from '../../../build/clazz/index';

  const dataRef = ref([]);
  const chartRef = useTemplateRef<HTMLElement>('chartRef');
  const chart = new SWCOMP.LineBarChart();

  const getData = () => {
    fetch('https://www.xiaolidan00.top/getRandNum.php')
      .then((res) => res.json())
      .then(({data}) => {
        dataRef.value = data;
        chartRef.value &&
          chart.init(
            chartRef.value,
            {
              loop: false,
              time: 2000,
              dataProps: {name: 'name', value0: 'value', value1: 'value1'},
              series: [
                {name: '销量', type: 'line'},
                {name: '盈利', type: 'bar'}
              ]
            },
            data
          );
        console.log(chart);
      });
  };
  onMounted(() => {
    getData();
  });
  onBeforeMount(() => {
    chart.destory();
  });
</script>
```

react中使用

```
import SWCOMP from '../../../build/clazz/index';
import '../../../build/clazz/index.css';
import type {DataItemType} from '../../../build/';
import {useEffect, useRef} from 'react';
export const ClassComp = () => {
```

```
  const chartRef = useRef<HTMLDivElement>(null);
  const chart = new SWCOMP.LineBarChart();
  useEffect(() => {
    fetch('https://www.xiaolidan00.top/getRandNum.php')
      .then((res) => res.json())
      .then(({data}) => {
        chartRef.current &&
          chart.init(
            chartRef.current as HTMLElement,
            {
              loop: true,
              time: 2000,
              dataProps: {name: 'name', value0: 'value', value1: 'value1'},
              series: [
                {name: '销量', type: 'line'},
                {name: '盈利', type: 'bar'}
              ]
            },
            data as DataItemType[]
          );
      });
    return () => {
      chart.destory();
    };
  }, []);
  return <div style={{height: '500px', width: '500px'}} ref={chartRef}></div>;
};
```

优势：

- 开发方式上很灵活。
- `class`的使用方式使用起来很方便，与echarts这样的纯js库类似。

劣势：

- 依赖于组件外的生命周期，组件的挂载、销毁和resize都需要调用class进行操作。
- 不能利用Vue和React虚拟DOM的方式增删改。
- 若需要支持SSR，代码复杂度增加，要将DOM实例转化成字符串。

组件测试

测试类型

- **单元测试**：检查给定函数、类或组合式函数的输入是否产生预期的输出或副作用。
- **组件测试**：检查你的组件是否正常挂载和渲染、是否可以与之互动，以及表现是否符合预期。这些测试比单元测试导入了更多的代码，更复杂，需要更多时间来执行。
- **端到端测试**：检查跨越多个页面的功能，并对生产构建的应用进行实际的网络请求。这些测试通常涉及到建立一个数据库或其他后端。

组件的单元测试

- **白盒：单元测试**

  白盒测试知晓一个组件的实现细节和依赖关系。它们更专注于将组件进行更 独立 的测试。这些测试通常会涉及到模拟一些组件的部分子组件，以及设置插件的状态和依赖性（例如 Pinia）。

- **黑盒：组件测试**

  黑盒测试不知晓一个组件的实现细节。这些测试尽可能少地模拟，以测试组件在整个系统中的集成情况。它们通常会渲染所有子组件，因而会被认为更像一种“集成测试”。

vitest+jsdom

`webcomponent组件测试`

```
/**
 * @vitest-environment jsdom
 */
import {describe, expect, test} from 'vitest';
import {install} from '../../packages/webcomponents/LineBarChart';
install('line-bar-chart');
describe('line-bar-chart tooltipType', () => {
  const types = ['axis', 'item'];
  types.forEach((item) => {
    test(item, () => {
      const element = document.createElement('div');
      element.innerHTML = `<line-bar-chart
    tooltipType="${item}"
      loop="true"
      time="2000"
      style="height: 500px; width: 500px"
      series='[{"name":"销量","type":"line"},{"name":"盈利","type":"bar"}]'
      dataProps='{"name":"name","value0":"value","value1":"value1"}'
      data='[{"name":"h3cjj","value":137,"value1":336},
{"name":"ha1f9","value":719,"value1":38},
{"name":"2mklu","value":58,"value1":258},
{"name":"8xk71","value":941,"value1":106},
{"name":"q0z9b","value":327,"value1":781},
{"name":"vg5cr","value":137,"value1":776},
{"name":"zut1h","value":379,"value1":615},
{"name":"cfvjd","value":450,"value1":716},
{"name":"det5n","value":805,"value1":377},
{"name":"y56ul","value":275,"value1":186}]'
    ></line-bar-chart>`;
      const dom = element.querySelector('line-bar-chart');
      expect(dom?.getAttribute('tooltipType')).eq(item);
    });
  });
```

```
});
```

Class组件测试

```
import {describe, expect, test} from 'vitest';
import LineBarChart from
'../../packages/components/LineBarChart/LineBarChart';
import type {LineBarChartConfig} from '.../../build/index.d.ts';
const data = [
  {name: 'h3cjj', value: 137, value1: 336},
  {name: 'ha1f9', value: 719, value1: 38},
  {name: '2mklu', value: 58, value1: 258},
  {name: '8xk71', value: 941, value1: 106},
  {name: 'q0z9b', value: 327, value1: 781},
  {name: 'vg5cr', value: 137, value1: 776},
  {name: 'zut1h', value: 379, value1: 615},
  {name: 'cfvjd', value: 450, value1: 716},
  {name: 'det5n', value: 805, value1: 377},
  {name: 'y56ul', value: 275, value1: 186}
];

describe('LineBarChart tooltipType', () => {
  const types = ['axis', 'item'];
  types.forEach((item) => {
    test(item, () => {
      const chart = new LineBarChart();
      const element = document.createElement('div');
      chart.init(
        element,
        {
          tooltipType: item as LineBarChartConfig['tooltipType'],
          loop: true,
          time: 2000,
          dataProps: {name: 'name', value0: 'value', value1: 'value1'},
          series: [
            {name: '销量', type: 'line'},
            {name: '盈利', type: 'bar'}
          ]
        },
        data
      );

      expect(chart.option.tooltip.trigger).eq(item);
    });
  });
});
```

```
✓ tests/clazz/LineBarChart.test.ts (2 tests) 9ms
  ✓ LineBarChart tooltipType (2)
    ✓ axis
    ✓ item
✓ tests/webcomp/LineBarChart.test.ts (2 tests) 27ms
  ✓ line-bar-chart tooltipType (2)
    ✓ axis
    ✓ item

Test Files  2 passed (2)
     Tests  4 passed (4)
  Start at  19:41:45
  Duration  3.52s (transform 768ms, setup 0ms, collect 3.74s, tests 36ms, environment 2.20s, prepare 318ms)
```

端对端测试

以下只是使用示例，端对端测试需到具体项目场景中执行!

```
import puppeteer from 'puppeteer';

(async () => {
  //打开浏览器
  const browser = await puppeteer.launch({headless: false});
  const page = await browser.newPage();

  //跳转到地址
  await page.goto('https://www.xiaolidan00.top/speed.html');
  //获取输入框
  const input = await page.waitForSelector('#text');
  //输入内容
  await input.type('Hello World!Hello World!Hello World!Hello World!Hello
World!Hello World!Hello World!');
  //点击按钮
  await page.click('button:first-child');
  //   await browser.close();
})();
```

组件使用手册

vitepress

- 静态站点生成器 (SSG)
- 友好支持Markdown
- vite生态

自动生成API文档

```
import {Project} from 'ts-morph';
import {resolvePath} from './utils.js';
import fs from 'node:fs';

const CLAZZAPI = 'docs/api/clazz/';
```

```
const WEBAPI = 'docs/api/webcomponents/';
const TYPEAPI = 'docs/api/index.md';

const project = new Project({tsConfigFilePath:
resolvePath('./tsconfig.json')});

function getTypeStr(t) {
  if (t.indexOf('undefined') >= 0) {
    t = t.replace('|', '').replace('undefined', '');
  }
  const i = t.indexOf('").');
  if (i >= 0) return t.substring(i + 3).trim();
  else return t.trim();
}
const TypeMap = {};
function getLinkStr(s) {
  const ss = s.replace(/[\[\]<>]/g, '');
  if (!TypeMap[ss]) {
    return `[\`${s}\`](https://developer.mozilla.org/zh-
CN/docs/Web/API/${s})`;
  } else return `[\`${s}\`](/api/#${ss.toLowerCase()})`;
}
function createClassDoc(file, API) {
  const cls = file.getClasses()[0];
  if (cls) {
    const className = cls.getName();
    const writer = fs.createWriteStream(resolvePath(API + className + '.md'));
    writer.write('# ' + className + '\n');
    const clsDoc = cls.getJsDocs()[0];
    if (clsDoc) {
      clsDoc.getTags().forEach((a) => {
        writer.write('\n- ' + a.getComment() + '\n');
      });
    }
    writer.write('##  属性\n');
    //组件属性
    if (cls.getInstanceProperty('config')) {
      writer.write('###  config\n');
      const config = cls.getInstanceProperty('config').getType().getText();
      writer.write(getLinkStr(getTypeStr(config)) + '\n\n');
    }
    if (cls.getInstanceProperty('data')) {
      writer.write('###  data\n');
      const config = cls.getInstanceProperty('data').getType().getText();

      writer.write(getLinkStr(getTypeStr(config)) + '\n\n');
    }

    writer.write('##  方法\n');
    //方法
    cls.getMethods().forEach((method) => {
      const params = {};
```

```
      method.getParameters().forEach((p) => {
        const t = p.getType().getText();
        params[p.getName()] = getTypeStr(t);
      });
      let returnType = method.getReturnType()?.getText();
      if (returnType) returnType = getTypeStr(returnType);
      const doc = method.getJsDocs()[0];
      if (doc) {
        writer.write('\n### ' + method.getName() + '\n');

        doc.getTags().forEach((a) => {
          const tag = a.getTagName();
          if (tag === 'description') {
            writer.write('\n**' + a.getComment() + '**\n\n');
          } else if (tag === 'param') {
            const n = a.getName();

            writer.write('- `' + n + '`{' + getLinkStr(params[n]) + '}:' +
a.getComment() + '\n');
          } else if (tag === 'returns' && returnType) {
            writer.write('- return {' + getLinkStr(returnType) + '}:' +
a.getComment() + '\n');
          }
        });
      }
    });
    writer.close();
  }
}

const writerTypes = fs.createWriteStream(resolvePath(TYPEAPI));
project.getSourceFiles().map((file) => {
  //type类型
  const types = file.getTypeAliases();

  types.forEach((t) => {
    const typeName = t.getName().trim();
    TypeMap[typeName] = 1;
    writerTypes.write('# ' + typeName + '\n');
    const doc = t.getJsDocs()[0];
    if (doc) {
      doc.getTags().forEach((a) => {
        writerTypes.write('\n**' + a.getComment() + '**\n\n');
      });
    }
    writerTypes.write('\n```ts\n');
    writerTypes.write(`type ${typeName}=${t.getStructure().type}`);
    writerTypes.write('\n```\n');
  });
  //interface类型
  const interfaces = file.getInterfaces();
  interfaces.forEach((it) => {
```

```
    const iName = it.getName().trim();
    TypeMap[iName] = 1;
    writerTypes.write('# ' + iName + '\n');
    const doc = it.getJsDocs()[0];
    if (doc) {
      doc.getTags().forEach((a) => {
        writerTypes.write('\n**' + a.getComment() + '**\n\n');
      });
    }
    writerTypes.write('\n```ts\n');
    writerTypes.write(`interface ${iName}{\n`);
    it.getMembers().forEach((m) => {
      writerTypes.write(m.getName() + ':' + getTypeStr(m.getType().getText())
+ ';\n');
    });
    writerTypes.write(`}`);
    writerTypes.write('\n```\n');
  });
});
writerTypes.close();

project.getSourceFiles().map((file) => {
  const filePath = file.getFilePath();
  if (filePath.indexOf('/components/') >= 0) {
    createClassDoc(file, CLAZZAPI);
  } else if (filePath.indexOf('/webcomponents/') >= 0) {
    createClassDoc(file, WEBAPI);
  }
});
```

LineBarChart

- 折线条形图

属性

config

`LineBarChartConfig`

data

`DataItemType[]`

方法

init

初始化配置样式和属性，设置数据，渲染组件

- `el` { `HTMLElement` }:DOM实例
- `config` { `LineBarChartConfig` }:echarts图表配置
- `data` { `DataItemType[]` }:数组

On this page

LineBarChart
属性
config
data
方法
init
setData
setConfig
getConfig
render
resize
destory

参考

1. echats
2. element-plus
3. antd
4. 字节跳动-国际化电商-S 项目团队《浅谈前端组件设计》
5. 阮一峰-Commit message 和 Change log 编写指南
6. vue测试
7. ts-morph
8. vitepress