01-规避 vant-cli 打包出错问题

why?

解决思路

实际使用时的样式冲突问题

问题复现

解决方案

origin

- 改成原来的包形式（定位为什么 vant-cli 构建的包没有样式互相影响的bug）
- `vue 组件库改造，问题解决写个文档梳理下`

1. 单独抽离库维护
2. vant-cli 只用于文档产出（构建有问题，官方未解决）
3. 解决单独抽离库后，在实际的 vant 项目中使用时，**可能**会造成的样式问题
    a. 问题怎么出现的
        - vant 的某些组件可能依赖了相同的组件
        - 外部直接使用 vant 组件，则组件的样式是按需加载的，比如页面中使用了 filed 组件，那么就可以在控制台的 meta 里面，看到 cell 相关的 style 被加载

- 而二次封装的 vant 组件也被在同一个页面使用，由于这个组件依赖的组件可能和直接使用的 vant 组件存在交集，此时两份相同类名的 css 都被加载，按道理说，css 代码都是相同的，样式应该不会出问题

b. 为什么出现（比较模糊）

- 经过观察，css 代码加载的顺序是不同的，例如

代码块

```
<style>
.d { color: red; }
.c { color: green; }
.b { color: blue; }
.a { color: orange; } /* 最后定义 */
</style>

<div class="d c b a">文字颜色是? </div>
```

- 最后颜色是 .a，因为它在最后被定义、加载
- 而两份相同的 css 被加载，最终效果可能不是原来的 d c b a，可能二开组件的 a 跑到了前面，导致样式出现问题

c. 解决

i. 添加统一父类名（同时也更符合二开库的定义），每个组件使用这个父类名

- 兼容普通组件封装时，不用新增一个div层级的问题 `.vantkit xxxx` 、 `.vantkitxxx`
- 之所以在每个组件使用这个父类名，而不是在实际使用的项目中的根节点添加目标类名，原因两点
  a. 降低使用心智负担
  b. 更方便在 vant-cli 中使用，因为没有直接暴露收集模拟器的根节点配置

ii. 经过完整的第一步操作，也能兼容在 vant-cli 中使用的问题

d. 依旧存在的问题

- 像toast、dialog的全局样式有问题
  - 即便我针对 toast dialog不打上 vankit 的前缀类名，也不行

代码块

```
// 预引入样式
import 'vant/es/toast/style';
import 'vant/es/dialog/style';
```

```
// 不用全量引入所有vant样式,因为 vite.config.ts 中的 VantResolver 已经帮助我们之引入已使用组件的代码和样式了,如果再次引入全量css，会导致很多重复的css代码被打包
// import 'vant/lib/index.css';
```

- 查看 `vant/es/toast/style` 、 `vant/es/dialog/style` 发现里面是依赖其他组件的 css 样式的，所以单纯不给 toast、dialog相关类名添加前缀类名是不可以的
- 所以 vant-kit 决定不主动引入样式，就像之前外部 h5 项目正常使用vant一样，手动引入这些全局样式。
- 对于 vant-cli 文档工程，直接全局引入 vant 样式即可

- 不应该拼接 `vantkitxxx` 因为有些样式是针对元素的，而非类名，例如 button 就会被覆盖为 `vankitbutton`

e. 经过上述改动，注销所有样式引入，只是依赖 `unplugin-vue-components/vite` + `@vant/auto-import-resolver` 的自动引入组件和组件样式。

- 问题：会少一些 vant 的基础样式，例如最简单的按钮中的默认文字颜色等等
- 解决：在库的入口文件引入 `import 'vant/lib/index.css'`
  - 但是我们之前提到过，这会导致许多重复样式的引入（可以观察构建产物），因为 `@vant/auto-import-resolver` 已经帮我们做了这些
  - 但是不用担心，我们可以取消 css 样式的自动引入 `importStyle: false` ，只在入口文件一次性引入 vant 样式即可。

代码块

```
import { defineConfig } from 'vite';
import unpluginComponents from 'unplugin-vue-components/vite';
import { VantResolver } from '@vant/auto-import-resolver';

export default defineConfig({
  plugins: [
    unpluginComponents({
      resolvers: [
        VantResolver({
          importStyle: false,
        }),
      ],
    })
  ],
});
```

f. 经过上述改动过程，终于解决问题

4. 现存问题，https://github.com/youzan/vant/issues/11876；
   https://github.com/youzan/vant/discussions/11875
   a. 估计和 最一开始遇到的样式问题是一个原因，组件多次加载
   b. 为什么这么设计？https://github.com/youzan/vant/issues/11285
   c. 解决过程：
      i. vant-kit 包中，将 vant 也设置成外部依赖，不进行打包
      ii. 外部使用该包时，在入口文件引入两个 css

```
代码块
1  import '@vmono/vant-kit/style.css';
2  import 'vant/lib/index.css';
```

      iii. 最关键一步，记得在 vite.config.ts 中的 VantResolver 插件中，关闭 css 的自动引入。不然当动态引入组件时，默认将会再次引入对应组件的 css 文件，就会与上一步，在默认文件中引入的 vant css 冲突。这也和 最一开始遇到的样式问题 是同一个原因😢。
5. 上面那个问题目前没解决，不过有个思路
   a. 在包里面的 css 中复写 vant-toast 的 z-index，这样外部引入包的css就会自动生效
6. Vant api 自动引入
   a. 由于 vant 我们已经设置为了外部依赖，而 vant 除了组件，还有一些 api，例如 showToast 等等，后期组件可能需要使用这些 api
   b. 使用 unplugin-auto-import/vite 插件

```
代码块
1  import { defineConfig } from 'vite';
2  import UnpluginComponents from 'unplugin-vue-components/vite';
3  import AutoImport from 'unplugin-auto-import/vite';
4  import { VantResolver } from '@vant/auto-import-resolver';
5
6  const VantResolverRes = VantResolver({
7    // 取消样式自动引入，已在入口文件一次性引入 vant 样式
8    importStyle: false,
9  });
10
11 export default defineConfig({
12   plugins: [
13     // 自动引入组件
14     UnpluginComponents({
15       resolvers: [VantResolverRes],
```

```
    }),
    // 自动引入 API
    AutoImport({
      resolvers: [VantResolverRes],
      eslintrc: {
        enabled: true,
      },
    }),
    ……
  ],
  ……
});

```

c. Commit 提交会触发 lint 报错，其实很好理解，因为插件帮我们做了自动引入，代码层并没有显式引入这些 api。导致 lint 报这些 api 未定义就使用。

d. 问题解决：配置插件自动生成 lint 的全局变量配置文件，然后在 eslint 配置文件中引入即可

   i. 配置插件：https://github.com/unplugin/unplugin-auto-import?tab=readme-ov-file#eslint

      会自动生成 .eslintrc-auto-import.json

      代码块

```
    AutoImport({
       resolvers: [VantResolverRes],
       eslintrc: {
         enabled: true,
       },
     })
```

   ii. 配置 eslint.config.js

      代码块

```
import autoImportConfig from './.eslintrc-auto-import.json';

export default [
  ...,
  {
    files: ['**/*.{ts,vue}'],
    languageOptions: {
      globals: {
        ...autoImportConfig.globals,
      },
```

```
    },
  },
];

```