

# 02-Field 相关二开组件 bug 解决

## 不能被 form 代理

### 问题描述

vant-kit 组件库在 **打包后**，**field 相关二开组件被外部宿主项目使用时**，与宿主项目中的原生 vant form 组件结合使用，字段不能被原生 vant form 收集。导致：通过 vant form 的事件、ref api，不能获取到相关 field 字段信息，也不能触发相关字段的校验，与 vant form 完全失联。



vant-kit 组件库中的 field 相关二开组件，如果直接将源码复制到宿主项目中使用，一切功能都是正常的。

- 为了方便调试，我们后续直接将 vant-kit-engineering 作为宿主项目进行调试
- 真实的宿主不仅包含 vant-kit-engineering(用于文档产出), 更重要的是应用在实际的 web 项目中，但是在实际的 web 项目中逐步调试比较麻烦，需要频繁发版。

### 最小问题复现示例

在 vant-kit 中编写一个只有 filed 的组件，没有其他逻辑

但为了和其它 filed 相关组件基础逻辑保持一致，因此有两个 filed ，分别处理字段的展示值与真实收集的值。

### 组件源码

vant-kit/src/TestField/TestField.vue

代码块

```
1  <template>
2    <van-field v-bind="computedFieldProps" v-model="modelFieldValue" />
3    <van-field v-bind="computedFieldProps" readonly :name="undefined" v-
  model="showValue!" />
4  </template>
5
6  <script lang="ts" setup>
7    import type { FieldProps } from 'vant';
8    import { computed } from 'vue';
9
10   const Props = defineProps<{
11     modelValue: string;
```

```

12   fieldProps: Partial<Omit<FieldProps, 'modelValue'>>;
13 }>();
14
15 const computedFieldProps = computed(() => {
16   const defaultProps: typeof Props.fieldProps = {
17     inputAlign: 'right',
18     errorMessageAlign: 'right',
19   };
20   return Object.assign(Props?.fieldProps ?? {}, defaultProps);
21 });
22
23 const Emitter = defineEmits<{
24   (e: 'update:modelValue', value: any);
25 }>();
26
27 const modelFieldValue = computed({
28   get() {
29     return Props.modelValue;
30   },
31   set(newValue) {
32     Emitter('update:modelValue', newValue);
33   },
34 });
35
36 const showValue = computed(() => `prefix--${modelFieldValue.value}--suffix`);
37
38 defineExpose({});
39 </script>
40
41 <style scoped lang="less">
42   .hidden {
43     display: none;
44   }
45 </style>

```

## demo1

然后在宿主项目 vant-kit-engineering（vant-cli 文档工程）中使用：

1. 编写一个 demo 组件
2. 随便在一个已经写好的组件文档 demo 中引入

demo 组件：**TestFieldUsage.vue**

代码块

```
1 <template>
```

```

2    <van-form ref="VantFormRef">
3      <p>modelValue:{{ modelValue }}</p>
4      <TestField
5        ref="TestFieldRef"
6        v-model="modelValue"
7        :field-props="{
8          name: 'testField',
9          label: 'testField',
10         readonly: false,
11         required: true,
12         rules: [{ required: true, message: '请输入' } ]},
13       >
14    />
15  </van-form>
16
17  <van-button type="primary" @click="() => validate()"> 验证 </van-button>
18 </template>
19
20 <script lang="ts" setup>
21 import { useWrapperRef } from '@vmono/vhooks';
22 import { ref } from 'vue';
23 import { TestField } from '@vmono/vant-kit';
24 import { Form as VanForm, Button as VanButton, FormInstance } from 'vant';
25
26 const TestFieldRef = ref<InstanceType<typeof TestField>>();
27
28 const VantFormRef = ref<FormInstance>();
29
30 const validate = async () => {
31   VantFormRef.value?.validate?().then(() => {
32     console.log(VantFormRef.value?.getValues?());
33   });
34 };
35
36 const [modelValue] = useWrapperRef('');
37 </script>
38
39 <style scoped lang="less"></style>

```

效果：

TestField

modelValue:

\*testField

\*testField prefix---suffix

验证

此时并没有在第一行(处理真正收集的值)表单中输入内容，此时点击 验证 按钮，正常情况下是要触发字段校验逻辑的。期望现象为：

1. 视图上出现红色的 '请输入' 提示文案
2. 未通过字段校验方法，因此不会触发字段的打印

而实际的现象与期望现象完全不一致：

1. 未出现必填提示文案
2. 通过了字段校验，并且打印了 `getValues` 获取的结果，且结果为空对象，不包含 `testField` 字段

## demo2

更新示例代码，让宿主项目(vant-kit-engineering)中的原生 vant form 组件同时包裹 vant-kit 中的 filed 二开组件与 vant 原生 field 组件，进行观察

代码块

```
1  <template>
2    <van-form ref="VantFormRef">
3      <p>modelValue:{{ modelValue }}</p>
4      <TestField
5        ref="TestFieldRef"
6        v-model="modelValue"
7        :field-props="{
8          name: 'testField',
9          label: 'testField',
10         readonly: false,
11         required: true,
12         rules: [{ required: true, message: '请输入' }],
13       }"
14    />
15    <van-field
16      v-model="vantFieldModelValue"
17      name="vantField"
```

```

18     label="vantField"
19     required
20     :rules="[{ required: true, message: '请选择' }]"
21   ></van-field>
22 </van-form>
23
24   <van-button type="primary" @click="() => validate()"> 验证 </van-button>
25 </template>
26
27 <script lang="ts" setup>
28 import { useWrapperRef } from '@vmono/vhooks';
29 import { ref } from 'vue';
30 import { TestField } from '@vmono/vant-kit';
31 import {
32   Form as VanForm,
33   Field as VanField,
34   Button as VanButton,
35   FormInstance,
36 } from 'vant';
37
38 const TestFieldRef = ref<InstanceType<typeof TestField>>();
39
40 const VantFormRef = ref<FormInstance>();
41
42 const validate = async () => {
43   VantFormRef.value?.validate?().then(() => {
44     console.log(VantFormRef.value?.getValues?());
45   });
46 };
47
48 const [modelValue] = useWrapperRef('');
49 const [vantFieldModelValue] = useWrapperRef('');
50 </script>
51
52 <style scoped lang="less"></style>

```

效果：

TestField

modelValue:

\*testField

\*testField prefix---suffix

\*vantField

验证

同样的，此时并没有在任何表单中输入内容，点击 `验证` 按钮的现象：

TestField

modelValue:

\*testField

\*testField prefix---suffix

\*vantField

请选择

验证

1. testField 组件表现和上面效果一致，视图上没有出现必填提示
2. vantField 组件表现正常，出现了必填提示，并且由于 form 的 validate 方法并未通过，因此控制台不会出现打印结果

而且由于出现未处理的 promise 错误(validate 方法并未通过)，控制台会自动打印相关错误信息，可以观察到：也只有 vantField 字段，testField 并未被原生 vant form 收集。

```
✖ ▶ Uncaught (in promise) : 3000/mobile.html#/field-date-picker:1
  ▼ [{...}] ⓘ
    ▶ 0: {name: 'vantField', message: '请选择'}
      length: 1
    ▶ [[Prototype]]: Array(0)
```

只填写 vantField 的表单值，不填写 testField 表单，点击 `验证` 按钮的现象：

TestField

modelValue:

\*testField

\*testField prefix---suffix

\*vantField vantField 值

验证

1. testField 组件表现和上面效果一致，视图上没有出现必填提示
2. vantField 组件表现正常，由于有表单值，不会出现必填提示

由于 testField 并未被原生 vant form 收集，form 只代理了 vantField 字段，此时表单会通过校验，并打印字段信息：结果也只有 vantField 字段。

```
vant-kit-engineering\mo index vue.js:332  
▶ {vantField: 'vantField 值'}
```

## 回顾现有信息

vant-kit 组件库中的 field 相关二开组件

- 如果直接将源码复制到宿主项目中使用，一切功能都是正常的。
- 在打包后，被外部宿主项目使用时，则不能够被外部的原生 vant form 收集和处理

**看起来问题好像出现在 vant-kit 组件库的打包上？**

但是为什么，其他二开组件功能都正常，只有 field 相关二开组件有问题？

带着这两个问题，我们需要研读一下 vant form、field 组件的源码，看一下二者是如何关联在一起的。

## 源码研读

### 相互关联的主要逻辑框架

vant form

- 使用 useChildren
  - children 收集
  - 关联 children: 通过 provide 把与 filed 进行关联的 api 进行暴露

## vant field

- 使用 `useParent`
  - 关联父级 form: 通过 `inject` 把 form 暴露的 api 进行使用，与 form 进行关联

## injectionKey

- 将 provide 与 inject 关联的 key，放到了 vant 工程中的 `utils/constant` 中

代码块

```
1 export const FORM_KEY: InjectionKey<FormProvide> = Symbol('van-form');
```

## 抓住嫌疑

form 与 field 组件的关联逻辑是通过 vue3 的 provide 与 inject 实现的。

所以问题很可能出现在 provide 与 inject 断联。


根据 vue 以及 provide 与 inject 的特性，可能导致断联的原因如下：

1. provide 与 inject 使用的 key 不同
2. 使用的 key 完全一致，但还是关联不上
  - a. 多个使用相同 key 的 provide 组件互为嵌套关系，则较外层的组件将不会被关联（provide & inject 的原理是利用原型链逐层攀爬进行查找，因此一旦在攀爬过程中找到了目标，就会停止攀爬）
  - b. 二者不在同一个 vue 实例下，例如多页应用（多个 vue 根节点）

## 定位问题

### ❌多 vue 实例？

相对其它嫌疑点，这个问题比较好验证，因为我们几乎可以排除掉它

 为什么 '几乎可以排除掉它'？

因为在 [📖01-规避 vant-cli 打包出错问题](#) 中，我们已经将 vue 设置成外部依赖了，不会在 vant-kit 中再次打包一次 vue

## demo 验证

1. 在 vant-kit 的入口文件中暴露 vue 的 h 函数，并打包
2. 在宿主项目(vant-kit-engineering)中分别从 vue、vant-kit 中引入 h 函数做比较，观察是否 `===`



## vant-kit/src/index.ts

代码块

```
1  .....  
2  export { h } from 'vue';
```

## vant-kit-engineering 中的 demo 组件：TestFieldUsage.vue

代码块

```
1  <script lang="ts" setup>  
2  import { h } from 'vue';  
3  import { h as kit_h } from '@vmono/vant-kit';  
4  
5  console.log('Vue h function:', h === kit_h);  
6  </script>
```

打印结果为 true

```
vant-kit-engineering...mo_index_vue...  
Vue h function: true
```

## ✅ injectionKey 不同

### 疑惑点

injectionKey 是在 vant 源码中的，而在 [📖 01-规避 vant-cli 打包出错问题](#) 中，我们已经将 vant 设置成外部依赖了，按理说引入的应该是同一份 vant。

我们需要编写 demo 验证下

### demo1



验证是否引入同一份 vant

1. 在 vant-kit 的入口文件中暴露 vant 的 Field 组件，并打包
2. 在宿主项目(vant-kit-engineering)中分别从 vant、vant-kit 中引入 Field 组件做比较，观察是否

===

## vant-kit/src/index.ts

```
1  .....
2  export { h } from 'vue';
3  export { Field } from 'vant';
```

vant-kit-engineering 中的 demo 组件: **TestFieldUsage.vue**

代码块

```
1  <script lang="ts" setup>
2  import { h } from 'vue';
3  import { Field as VanField } from 'vant';
4  import { h as kit_h, Field as kit_Field } from '@vmono/vant-kit';
5
6  console.log('Vue h function:', h === kit_h);
7  console.log('Field cpn:', VanField === kit_Field);
8  </script>
```

打印结果为 true

```
vant-kit-engineering...mo_index_vue...
vant-kit Vue h function: true
vant-kit-engineering...mo_index_vue...
Field cpn: true
```

## demo2



直接验证 injectionKey 是否相等

1. 为了进一步的观察 injectionKey 我们在 vant-kit 的 TestField.vue 中 inject FORM\_KEY 观察是否能够获取到相关信息
2. 在 vant-kit 的入口文件中暴露 'vant/es/utils' 中的 FORM\_KEY, 并打包
3. 在宿主项目(vant-kit-engineering)中分别从 'vant/es/utils'、vant-kit 中引入 FORM\_KEY 做比较, 观察是否 `===`

**vant-kit/src/TestField/TestField.vue**

代码块

```
1  <script lang="ts" setup>
2  import { FORM_KEY } from 'vant/es/utils';
3  import { inject } from 'vue';
4
5  const vanForm = inject(FORM_KEY, null);
```

```
6 console.log('vanForm inject result:', FORM_KEY, vanForm);
7 </script>
```

## vant-kit/src/index.ts

代码块

```
1 .....
2 export { h } from 'vue';
3 export { Field } from 'vant';
4 export { FORM_KEY as VanKitFormKey } from 'vant/es/utils';
```

## vant-kit-engineering 中的 demo 组件: TestFieldUsage.vue

代码块

```
1 <script lang="ts" setup>
2 import { h } from 'vue';
3 import { Field as VanField } from 'vant';
4 import { h as kit_h, Field as kit_Field, VanKitFormKey } from '@vmono/vant-
  kit';
5
6 console.log('Vue h function:', h === kit_h);
7 console.log('Field cpn:', VanField === kit_Field);
8 console.log(
9   'FORM_KEY:',
10   VanKitFormKey === FORM_KEY,
11   VanKitFormKey?.description,
12   FORM_KEY.description,
13 );
14 </script>
```

打印结果:

- 虽然 FORM\_KEY 的 description 都一致，但二者不相同
- 所以：宿主项目中的 FORM\_KEY 与 vant-kit 中的 FORM\_KEY 不是从同一个文件中引入的。

```
vant-kit-engineering...mo_index_vue...
Vue h function: true
vant-kit-engineering...mo_index_vue...
Field cpn: true
vant-kit-engineering...mo_index_vue...
FORM_KEY: false van-form van-form
```

当然了，在上述的三个用例中，在 vant-kit 中的 TestField 组件的打印结果中，也并没有成功的 inject 到信息

```
packages_vant-kit_di...ooks_es_js.js:14...
vanForm inject result: Symbol(van-form)
null
```

## demo3



尝试 vant-kit 中的 form 能否代理 vant-kit 中的 field

1. 在 vant-kit 中编写 FormWrapper 组件，并在入口文件中暴露，打包。
2. 在 vant-kit-engineering 宿主项目中使用 FormWrapper 组件包裹 field 二开组件，测试功能

### vant-kit/src/FormWrapper/FormWrapper.vue

代码块

```
1  <template>
2    <van-form ref="formRef" v-bind="Props">
3      <!-- 暴露默认支持的插槽 -->
4      <template v-for="(_slot, name) in $slots" #[name]="slotData" :key="name">
5        <slot :name="name" v-bind="slotData" :key="name"></slot>
6      </template>
7    </van-form>
8  </template>
9
10 <script lang="ts" setup>
11   import type { FormInstance } from 'vant';
12   import { ref } from 'vue';
13   const Props = defineProps<{ formProps: any }>();
14   const formRef = ref<FormInstance>();
15
16   defineExpose({ formRef });
17 </script>
18
19 <style scoped lang="less"></style>
```

### vant-kit/src/index.ts

代码块

```
1  .....
2  // 测试表单组件
3  export { default as TestField } from './TestField/TestField.vue';
4  // 表单包装组件
```

```
5 export { default as FormWrapper } from './FormWrapper/FormWrapper.vue';
6
7 export { h } from 'vue';
8 export { Field } from 'vant';
9 export { FORM_KEY as VanKitFormKey } from 'vant/es/utils';
```

## vant-kit-engineering 中的 demo 组件: **TestFieldUsage.vue**

代码块

```
1 <template>
2   <van-form ref="VantFormRef">
3     <FormWrapper ref="FormWrapperRef">
4       <p>modelValue:{{ modelValue }}</p>
5       <TestField
6         v-model="modelValue"
7         :field-props="{
8           name: 'testField',
9           label: 'testField',
10          readonly: false,
11          required: true,
12          rules: [{ required: true, message: '请输入' }],
13        }"
14      />
15      <van-field
16        v-model="vantFieldModelValue"
17        name="vantField"
18        label="vantField"
19        required
20        :rules="[{ required: true, message: '请选择' }]"
21      ></van-field>
22    </FormWrapper>
23  </van-form>
24
25  <van-button type="primary" @click="validate"> 验证 </van-button>
26 </template>
27
28 <script lang="ts" setup>
29 import { useWrapperRef } from '@vmono/vhooks';
30 import { ref } from 'vue';
31 import { TestField, FormWrapper } from '@vmono/vant-kit';
32 import {
33   Form as VanForm,
34   Field as VanField,
35   Button as VanButton,
36   FormInstance,
```

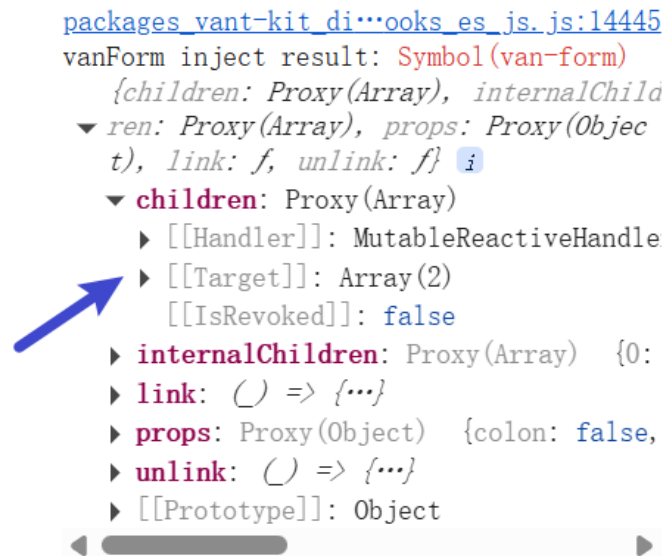
```

37 } from 'vant';
38
39 import { h } from 'vue';
40 import { h as kit_h } from '@vmono/vant-kit';
41 import { Field as kit_Field, VanKitFormKey } from '@vmono/vant-kit';
42 import { FORM_KEY } from 'vant/es/utils';
43
44 console.log('Vue h function:', h === kit_h);
45 console.log('Field cpn:', VanField === kit_Field);
46 console.log(
47   'FORM_KEY:',
48   VanKitFormKey === FORM_KEY,
49   VanKitFormKey?.description,
50   FORM_KEY.description,
51 );
52
53 const FormWrapperRef = ref<InstanceType<typeof FormWrapper>>>();
54 const VantFormRef = ref<FormInstance>();
55
56 const validate = async () => {
57   // 原生、和组件库的表单只能分开验证
58   await Promise.all([
59     FormWrapperRef.value?.formRef?.validate?(),
60     VantFormRef.value?.validate?(),
61   ]);
62   console.log(FormWrapperRef.value?.formRef?.getValues?());
63   console.log(VantFormRef.value?.getValues?());
64 };
65
66 const [modelValue] = useWrapperRef('');
67 const [vantFieldModelValue] = useWrapperRef('');
68 </script>
69
70 <style scoped lang="less"></style>

```

能够立刻观察到，在 vant-kit 中的 TestField 组件的打印结果中，已经能够 inject 到信息了

- 展开 children 数组，发现有两个对象，点开观察 name 字段，发现是用 TestField 组件生成的 field 表单字段，分别是 testField (真实绑定的 name)，undefined (绑定给用于展示的 field)



- 这里其实已经说明 vant-kit 中的 form 是可以代理 vant-kit 中的 field 让我们进行不同用例测试，进一步了解情况👉

**用例1：填写 testField、vantField，点击验证按钮，观察结果。**

TestField

modelValue:testField value

\* testField

testField value

\* testField

prefix--testField value--suffix

\* vantField

vantField value

验证

VantFormRef、FormWrapperRef 校验均通过，getValues 也只能获取到各自的表单组件 field 字段信息：

```
vant-kit-engineering...mo_index_vue...
  ▶ {testField: 'testField value'}
vant-kit-engineering...mo_index_vue...
  ▶ {vantField: 'vantField value'}
```

**用例2：填写 testField，不填写 vantField，点击验证按钮，观察结果。**





```
Uncaught :3000/mobile.html#/field-date-picker:1
ht (in promise)
  ↳ [...]] i
    ↳ 0: {name: 'testField', message: '请输入'}
      length: 1
    ↳ [[Prototype]]: Array(0)
```

## 结论

通过编写 demo 示例，现象为：

1. vant-kit 中的 vue 与宿主环境一致
2. vant-kit 中的组件实例与宿主环境一致
3. vant-kit 中的 injectionKey 与宿主环境引入的不一致

并且，vant-kit 中的 injectionKey 也能正常运转，具有代理能力，只不过和宿主环境的 injectionKey 隔离的

## Bug 解决

上述得到的结论中: vant-kit 中的组件实例与宿主环境一致，但是 injectionKey 却不一致。

这个现象还挺反直觉的。

因为 vant-kit 项目的 **vite.config.ts** 中，我们确实把 vant 作为了外部依赖

代码块

```
1      rollupOptions: {
2          /**
3           * 📌 告诉 Rollup: 不要打包这些模块
4           * 因为这些模块在使用该库的外部项目中会被提供
5           * 1. 这样可以避免重复打包，减小库的体积
6           * 2. 防止 vant 组件库被多次引入，避免样式冲突
7           */
8          external: ['vue', 'vant'],
9          output: {
10             // 📌 将使用该库的外部项目中，需要自行引入的模块映射为全局变量
11             globals: {
12                 vue: 'Vue',
13                 vant: 'Vant',
14             },
15         },
16     },
```

而 injectionKey 也是通过从 vant 包内部去引入的。

只不过是通過其子目錄 `vant/es/utils` 中進行引入的（當然了，也可以從 `vant/lib/utils` 中引入）

經過了解後，這裡只是將 `vant` 進行了外部化，並沒有將 `vant` 的子模塊進行外部化 😞。

因此我們要新增配置項，將 `vant` 的子包也進行外部化

1. external 新增 `/^vant\\//` 匹配 `vant` 的子包
2. 在 `globals` 中添加 `'vant/es/utils': 'Vant.utils'` 映射。這將體現在構建產物中的 `umd.js` 文件中

代碼塊

```
1      rollupOptions: {
2        /**
3         * 📌 告訴 Rollup: 不要打包這些模塊
4         * 因為這些模塊在使用該庫的外部項目中會被提供
5         * 1. 這樣可以避免重複打包，減小庫的體積
6         * 2. 防止 vant 組件庫被多次引入，避免樣式衝突
7         */
8        external: ['vue', 'vant', /^vant\\//],
9        output: {
10          // 📌 將使用該庫的外部項目中，需要自行引入的模塊映射為全局變量
11          globals: {
12            vue: 'Vue',
13            vant: 'Vant',
14            'vant/es/utils': 'Vant.utils',
15          },
16        },
17      },
```

應用該配置將 `vant-kit` 打包後，更新宿主項目(`vant-kit-engineering`)中的 `demo` 組件：

**TestFieldUsage.vue** 代碼

1. 只使用宿主環境中的原生 `vant form` 組件包裹 `TestField`、原生 `vant filed` 組件
2. 驗證、獲取字段也只使用 `VantFormRef` 進行操作

代碼塊

```
1  <template>
2    <van-form ref="VantFormRef">
3      <p>modelValue:{{ modelValue }}</p>
4      <TestField
5        v-model="modelValue"
6        :field-props="{
7          name: 'testField',
8          label: 'testField',
9          readonly: false,
```

```

10         required: true,
11         rules: [{ required: true, message: '请输入' }],
12     }"
13 />
14 <van-field
15     v-model="vantFieldModelValue"
16     name="vantField"
17     label="vantField"
18     required
19     :rules="[{ required: true, message: '请选择' }]"
20 ></van-field>
21 </van-form>
22
23 <van-button type="primary" @click="validate"> 验证 </van-button>
24 </template>
25
26 <script lang="ts" setup>
27 import { useWrapperRef } from '@vmono/vhooks';
28 import { ref } from 'vue';
29 import { TestField, FormWrapper } from '@vmono/vant-kit';
30 import {
31     Form as VanForm,
32     Field as VanField,
33     Button as VanButton,
34     FormInstance,
35 } from 'vant';
36
37 import { h } from 'vue';
38 import { h as kit_h } from '@vmono/vant-kit';
39 import { Field as kit_Field, VanKitFormKey } from '@vmono/vant-kit';
40 import { FORM_KEY } from 'vant/es/utils';
41
42 console.log('Vue h function:', h === kit_h);
43 console.log('Field cpn:', VanField === kit_Field);
44 console.log(
45     'FORM_KEY:',
46     VanKitFormKey === FORM_KEY,
47     VanKitFormKey?.description,
48     FORM_KEY.description,
49 );
50
51 const FormWrapperRef = ref<InstanceType<typeof FormWrapper>>();
52 const VantFormRef = ref<FormInstance>();
53
54 const validate = async () => {
55     await VantFormRef.value?.validate?();
56     console.log(VantFormRef.value?.getValues?());

```

```

57   };
58
59   const [modelValue] = useWrapperRef('');
60   const [vantFieldModelValue] = useWrapperRef('');
61   </script>
62
63   <style scoped lang="less"></style>

```

vant-kit 中的 FORM\_KEY 与宿主项目中的相同

```

vant-kit-engineering...mo_index...
vant-kit Vue h function: true
vant-kit-engineering...mo_index...
Field cpn: true
vant-kit-engineering...mo_index...
Vue h function: true
vant-kit-engineering...mo_index...
Field cpn: true
vant-kit-engineering...mo_index...
FORM_KEY: true van-form van-form

```

在 vant-kit 中的 TestField 组件的打印结果中，inject 到的 children 也变成了 3 个(包含了 vant-kit 的二开 field，与宿主项目中的原生 field)

```

packages_vant-kit_di...ooks_es...
vanForm inject result: Symbol(v
an-form)
  {children: Proxy(Array), int
ernalChildren: Proxy(Array),
  ▼ props: Proxy(Object), link:
    f, unlink: f} ⓘ
    ▼ children: Proxy(Array)
      ▶ [[Handler]]: MutableReac
      ▶ [[Target]]: Array(3)
        [[IsRevoked]]: false
      ▶ internalChildren: Proxy(Ar
      ▶ link: (child) => {...}
      ▶ props: Proxy(Object) {col
      ▶ unlink: (child) => {...}
      ▶ [[Prototype]]: Object

```

用例1：不输入任何字段，点击验证按钮触发校验

TestField

modelValue:

\* testField

请输入

\* testField

prefix----suffix

\* vantField

请选择

验证

- 所有字段的必填校验全部正常触发，出现必填提示信息
- 控制台报错信息中能够打印所有未通过验证的字段

```

Uncaught :3000/mobile.html#/field-date-picker:1
ht (in promise)
  (2)  [{...}, {...}] i
    0: {name: 'testField', message: '请输入'}
    1: {name: 'vantField', message: '请选择'}
    length: 2
    [[Prototype]]: Array(0)
  
```

## 用例2：输入所有字段，点击验证按钮触发校验

TestField

modelValue:testField value

\* testField

testField value

\* testField

prefix--testField value--suffix

\* vantField

vantField value

验证

表单验证通过，能够正常打印所有字段

```

vant-kit-engineering...mo_index_v...
  {testField: 'testField value',
    vantField: 'vantField value'}
  
```