

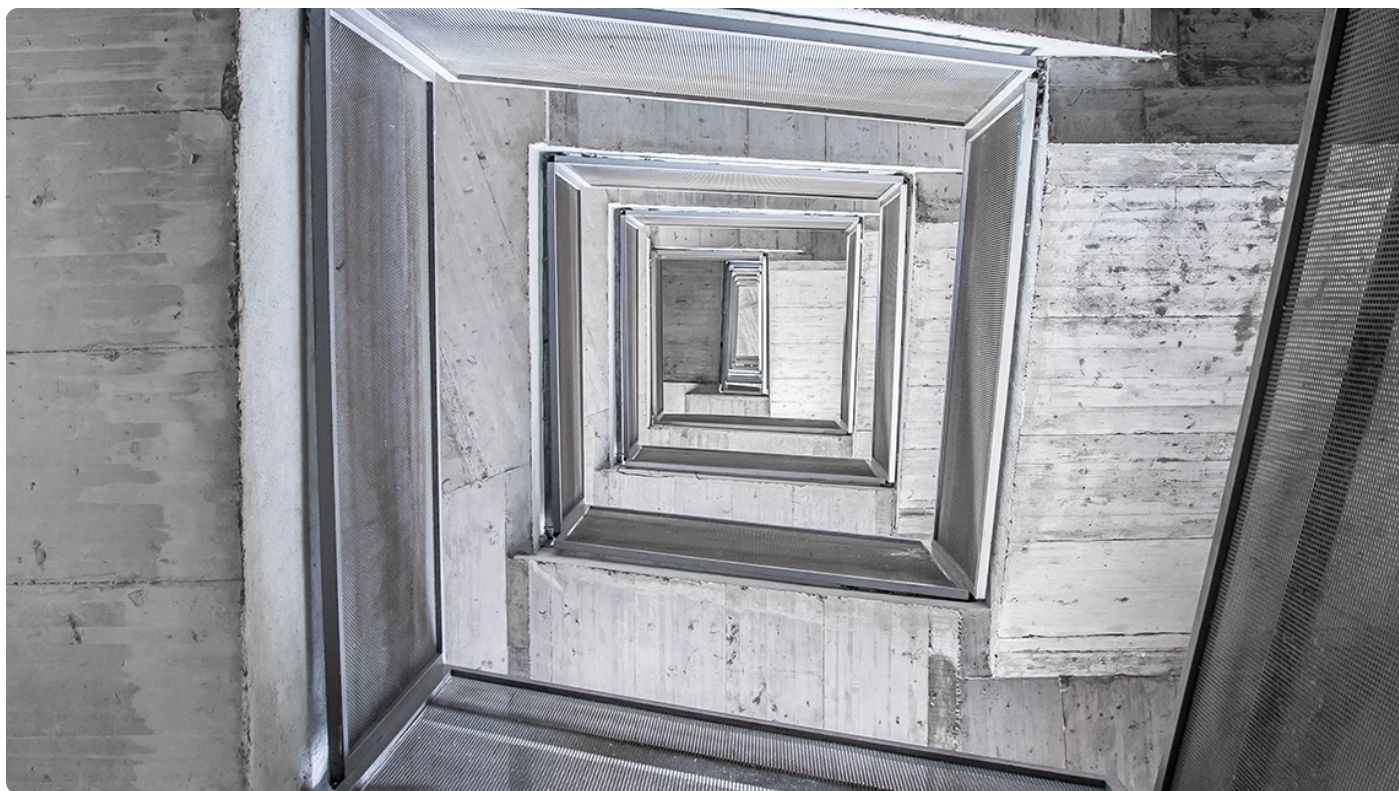
13 | 动态表单组件：怎么优雅地动态渲染表单？

2022-12-21 杨文坚 来自北京

天下无鱼
<https://shikey.com/>

《Vue 3 企业级项目实战课》

课程介绍 >



讲述：杨文坚

时长 12:17 大小 11.22M



你好，我是杨文坚。

上节课，我们学习 Vue.js 3.x 自研组件库的“受控表单组件”，开发了“表单字段组件”来辅助提高开发表单的效率。但是，在实际的企业级项目中，业务需求总是“紧急”且“多变”的，表单类的开发需求，不能只靠一个“受控表单组件”的能力来提效。我们看个常见例子。

假设你接到一个业务需求，要开发一个“用户信息设置”的页面，让注册用户可以编辑自己的个人信息，常规开发步骤，我们一般是用表单组件来封装这个用户信息配置的功能。

但接下来业务需求变了，业务方要做用户类型的区分，不同类型的用户显示“不同个人信息配置”，比如这里就有“普通用户”、“多种等级会员用户”的信息配置，后续可能会新增其他维度类型的用户信息配置。这时候，你还能用常规的表单开发思路，来开发这个需求吗？

我们简单分析一下，如果按常规开发思路，每新增一个用户类型，就要重新开发一个用户信息表单来支持业务需求，工作量就是无底洞。所以问题就来了，**是否有一种表单方案，能够通过简单的自定义配置，快速生成对应的表单功能呢？**



答案是有的，就是“动态表单组件”。这节课，我们就来学习如何基于 Vue.js 3.x，开发自研组件库里的“动态表单组件”。

什么是动态表单？

“动态表单”，顾名思义，就是能“动态”生产想要功能的“表单”。

在前端技术视角里，动态表单概念在十几年的 jQuery 时代就有了，可以用“简单配置”方式来“动态生成表单”，通过一个 JSON 的数据形式来描述表单格式，再通过 JavaScript 代码，根据数据描述渲染出表单。

可以看出，**动态表单，核心就是通过自定义数据来动态生成自定义表单**。也就是说，面对实际的开发需求时，每当新增一个表单类型的需求，我们只需要配置一下“数据”就能生成表单，不需要从零开始来开发。

这类技术方案在前端开发中经常用到，例如，开发后台页面场景时，不同类型用户身份，信息录入需要渲染多种表单；搭建页面场景时，动态生成调查问卷页面，让用户可以配置不同数据来生成不同问题内容的表单，这个过程无需投入额外的前端开发工作，用户可以自助配置数据生成问卷的表单。

那么，“动态表单”到底怎么实现呢？一步到位肯定不怎么现实，所以我们先从一个最简单的动态表单开始，先来看如何用 Vue.js3.x 实现一个最简单的动态表单。

如何用 Vue.js3.x 实现最简单的动态表单？

在动手实现之前，我们先分析一下，表单哪些内容可以统一进行动态管理。

上节课我们讲过，表单核心是由一个个表单字段组成，每个表单字段背后都是一个个表单数据组件。那么，动态表单，其实就是按照动态数据的内容，生成对应“表单数据组件”的各种“组合”的“结果”。**实现一个最简单的表单组件，其实就是根据自定义数据，来自定义生成表单数据组件的各种组合。**

实现步骤现在就很清晰了：

- 第一步，列举要用到的表单数据组件，例如 input, radio 等；
- 第二步，定义描述动态表单的数据格式；
- 第三步，根据数据格式来渲染动态表单。

来看代码：

复制代码

```
1 <template>
2   <form class="dynamic-form" @submit="onSubmit">
3     <div class="dynamic-form-title">{{ schema.title }}</div>
4     <div
5       class="dynamic-form-field"
6       v-for="(field, index) in schema.fieldList"
7       v-bind:key="index"
8     >
9       <div class="dynamic-form-label">{{ field.label }}: </div>
10      <div v-if="field.fieldType === 'input'" class="dynamic-form-item">
11        <input v-model="model[field.name]" />
12      </div>
13      <div v-else-if="field.fieldType === 'radio'" class="dynamic-form-item">
14        <span
15          v-for="(option, index) in field.options"
16          v-bind:key="index"
17          class="dynamic-form-option"
18        >
19          <input
20            type="radio"
21            :id="option.value"
22            :name="field.name"
23            :value="option.value"
24            :checked="model[field.name] === option.value"
25            @change="
26              onRadioChange({ fieldName: field.name, value: option.value })
27            "
28          />
29          <label :for="option.value">{{ option.name }}</label>
30        </span>
31      </div>
32      <div v-else class="dynamic-form-item"></div>
33    </div>
34    <div>
35      <button class="dynamic-form-btn" type="submit">提交</button>
36    </div>
37  </form>
38 </template>
```

```

39 </script>
40 <script setup lang="ts">
41   import { reactive, toRaw } from 'vue';
42
43   const schema = {
44     title: '普通用户信息',
45     fileList: [
46       {
47         label: '用户名称',
48         name: 'username',
49         fieldType: 'input'
50       },
51       {
52         label: '手机号码',
53         name: 'phone',
54         fieldType: 'input'
55       },
56       {
57         label: '收货地址',
58         name: 'address',
59         fieldType: 'input'
60       }
61     ]
62   };
63
64   const model = reactive<{ [key: string]: unknown }>({});
65   schema.fieldList.forEach((field) => {
66     model[field.name] = '';
67   });
68
69   const onRadioChange = (data: { fieldName: string; value: string }) => {
70     model[data.fieldName] = data.value;
71   };
72
73   const onSubmit = (e: Event) => {
74     e.preventDefault();
75     window.alert(JSON.stringify(data));
76   };
77 </script>
78 <style> /* 样式代码省略，请看代码案例 */ </style>

```



天下无鱼

<https://shikey.com/>

我先定义了描述动态表单的数据格式 **schema**，然后，把动态表单的描述数据，按照数组形式进行管理，根据描述数据在数组里的排列形式，按需渲染出表单内容。

这里，**schema** 定义的动态表单数据，是一个“普通用户”的数据，表单的动态渲染结果如下图所示：



这个简单的动态表单，内部支持了两种表单数据组件，input 输入框和 radio 单项选择，现在我们把 schema 修改一下，改成“会员用户信息”的数据，代码如下：

复制代码

```
1 const schema = {
2   title: '会员用户信息',
3   fieldList: [
4     {
5       label: '用户名称',
6       name: 'username',
7       fieldType: 'input'
8     },
9     {
10      label: '手机号码',
11      name: 'phone',
12      fieldType: 'input'
13    },
14    {
15      label: '优惠服务',
16      name: 'service',
17      fieldType: 'radio',
18      options: [
19        { name: '免运费', value: 'service001' },
20        { name: '9折优惠', value: 'service002' },
21        { name: '满80减10', value: 'service003' }
22      ]
23    },
24    {
```

```
25     label: '收货地址',
26     name: 'address',
27     fieldType: 'input'
28   }
29 ]
30 }
```



这里的“会员用户”表单描述数据，比“普通用户”的多了一个“优惠服务”的表单描述字段，动态表单渲染效果如下：

会员用户信息

用户名称:

手机号码:

优惠服务: ☐ 免运费 ☒ 9折优惠 ☐ 满80减10

收货地址:

提交

这两个表单功能，其实出于同一个动态表单的代码，我们只是将表单描述数据 `schema` 做对应差异修改，就能直接渲染出对应不同的表单功能。

通过，这样一个简单的 `Vue.js 3.x` 动态表单实现，我们可以总结出**动态表单实现的三个核心要素**：

- 第一点，梳理要用到的表单数据组件；
- 第二点，根据表单数据组件种类制定数据格式；
- 第三点，根据数据格式的内容，来渲染表单数据组件的自定义组合，这个自定义组合就是所需要的表单结果。

我们用 **Vue.js** 实现了一个简单的动态表单，实际业务需要的动态表单功能可不只这些，我们还要考虑表单校验逻辑、扩展新的表单数据组件等等，这需要一个更完善的动态表单组件。



如何设计和实现完善的动态表单组件？

在具体实现之前，我们先设计动态表单组件的规范。这个规范除了能满足现有的功能需要，还需要有前瞻性的设计，保证能扩展“新的表单动态组件”，不能局限于一开始约定的表单数据组件。

按照要求，再结合实现最简单的动态表单的核心要素，我重新梳理了四点实现要素：

- 定义表单数据组件的统一 API；
- 定义默认支持的数据表单组件；
- 支持自定义表单字段的校验规则；
- 支持根据统一 API 扩展自定义表单数据组件。

我们来具体分析一下每一点要素。

第一点，定义表单数据组件的统一 API。由于动态表单核心是由各种“表单数据组件”的组成，但是，每个表单数据组件，都有各自原生的 API 使用方式，这些 API 的差异会降低兼容性。**所以，我们需要约定好统一的表单数据组件的 API，对不同表单数据组件做 API 统一封装。**

第二点，定义默认支持的数据表单组件。常用的表单数据组件要在动态表单内默认支持，所以我们要用统一的 API 进行二次封装，并内置到动态表单组件内。这里表单数据组件的 API 统一代码如下所示：

 复制代码

```
1  const props = defineProps<{
2    // 传入的数据值
3    value: string | any;
4    // 组件内部选项参数，例如多选框，单选框，下拉框的选项数据
5    options: Array<{ name: string; value: string }>;
6  }>();
7
8  const emits = defineEmits<{
9    // 监听组件内部数据变化事件
10    (e: 'change', value: string): void;
11  }>();
```

第三点，支持自定义表单字段和校验规则。上节课我们说了，抽象表单里最重要的复用逻辑就是“表单校验”，当时我演示了如何封装一个“表单字段组件”作为“表单数据组件”的外壳，统一管理字段校验规则。所以，**自定义表单校验规则，对动态表单来讲也很重要，我们可以把上节课的表单字段组件，沿用到我们的动态表单组件里，统一管理表单校验。**

第四点，支持根据统一 API 扩展自定义表单数据组件。

既然要实现动态表单组件，我们就不能只支持默认表单数据组件的表单生成。毕竟，如果不能扩展新的表单数据组件，以后有新的表单需求，要用到自定义的表单数据组件，动态表单组件就不能快速配置生成表单了，需要我们从零开发一个支持自定义数据组件的表单。这就失去动态表单原本提效的意义了。

所以，**我们这里需要支持可扩展自定义表单数据组件，但有个前提，就是自定义表单数据组件要按照第一点要素的内容，封装统一的 API。**

好了，那么我们现在来根据四点要素，实现动态表单组件，看代码：

[复制代码](#)

```
1 <template>
2   <div :class="{ [baseClassName]: true }">
3     <Form>
4       <Form
5         ref="formRef"
6         :model="internalModel"
7         @finish="onFinish"
8         @finishFail="onFinishFail"
9       >
10        <FormItem
11          v-for="(field, index) in fieldList"
12          :key="index"
13          :label="field.label"
14          :name="field.name"
15          :rule="field.rule"
16        >
17          <component
18            :is="registerComponentMap[field.fieldType]"
19            :value="internalModel[field.name]"
20            :options="field.options || []"
21            @change="(value: unknown) => { onFieldChange({ name: field.name, va
22          />
23        </FormItem>
```



```

24         <Row v-if="$slots.default">
25             <slot></slot>
26         </Row>
27     </Form>
28 </Form>
29 </div>
30 </template>
31
32 <script setup lang="ts">
33 import { reactive } from 'vue';
34 import { prefixName } from '../theme';
35 import Row from '../row';
36 import Form from '../form';
37 import Input from '../input';
38 import RadioList from '../radio-list';
39 import type { Component } from 'vue';
40 import type { DynamicFormField } from '../common';
41
42 // 内置支持的表单数据组件
43 const registerComponentMap: { [key: string]: Component } = {
44     Input: Input,
45     RadioList: RadioList
46 };
47
48 const props = withDefaults(
49     defineProps<{
50         fieldList?: DynamicFormField[];
51         model?: { [name: string]: unknown };
52     }>(),
53     {}
54 );
55
56 const internalModel = reactive<{ [name: string]: unknown }>(props?.model || {})
57 const FormItem = Form.FormItem;
58 const baseClassName = `${prefixName}-dynamic-form`;
59
60 const onFieldChange = (event: { name: string; value: string | unknown }) => {
61     internalModel[event.name] = event.value;
62 };
63
64 const emits = defineEmits<{
65     (event: 'finish', e: unknown): void;
66     (event: 'finishFail', e: unknown): void;
67 }>();
68
69 const onFinish = (e: unknown) => {
70     emits('finish', e);
71 };
72
73 const onFinishFail = (e: unknown) => {
74     emits('finishFail', e);
75 };

```



```
76 </script>
77
```



上述代码中，我设计了动态表单的数据格式，通过数组一一对应来描述表单字段内容的。每个表单字段的数据描述有：表单的标签名称、字段名称、字段类型、字段使用数据组件的类型和校验规则。

我们根据实现的动态表单组件，来生成一个可校验的“普通用户”信息编辑表单，代码如下所示：

复制代码

```
1 <template>
2   <div class="example">
3     <DynamicForm
4       :model="model"
5       :fieldList="fieldList"
6       @finish="onFinish"
7       @finishFail="onFinishFail"
8     >
9       <Button type="primary">提交信息</Button>
10    </DynamicForm>
11  </div>
12 </template>
13
14 <script setup lang="ts">
15 import { Button, DynamicForm } from '../src';
16 import type { DynamicFormField } from '../src';
17
18 const model = {
19   username: 'Hello',
20   phone: '123456',
21   address: '',
22   service: ''
23 };
24
25 const fieldList: DynamicFormField[] = [
26   {
27     label: '用户名称',
28     name: 'username',
29     fieldType: 'Input',
30     rule: {
31       validator: (val: unknown) => {
32         const hasError = /^[a-z]{1,}$/gi.test(`${val} || ''`) !== true;
33         return {
34           hasError,
35           message: hasError ? '仅支持a-z的大小写字母' : ''
36         };
37       }
38     }
39   }
40 ]
```

```
38   }
39 },
40 {
41   label: '手机号码',
42   name: 'phone',
43   fieldType: 'Input',
44   rule: {
45     validator: (val: unknown) => {
46       const hasError = /^[0-9]{1,}$/gi.test(`${val} || ''`) !== true;
47       return {
48         hasError,
49         message: hasError ? '仅支持0-9的数字' : ''
50       };
51     }
52   }
53 },
54 {
55   label: '快递地址',
56   name: 'address',
57   fieldType: 'Input',
58   rule: {
59     validator: (val: unknown) => {
60       const hasError = `${val}`?.length === 0;
61       return {
62         hasError,
63         message: hasError ? '地址不能为空' : ''
64       };
65     }
66   }
67 }
68 ];
69
70 const onFinish = (e: any) => {
71   // eslint-disable-next-line no-console
72   console.log('success =', e);
73 };
74
75 const onFinishFail = (e: any) => {
76   // eslint-disable-next-line no-console
77   console.log('fail =', e);
78 };
79 </script>
80
81 <style>
82 html,
83 body {
84   height: 100%;
85   width: 100%;
86 }
87 .example {
88   width: 400px;
89   padding: 16px;
```

```

90     margin: 20px auto;
91     box-sizing: border-box;
92     border-radius: 4px;
93     border: 1px solid #999999;
94     font-size: 14px;
95 }
96
97 .btn {
98     height: 32px;
99     padding: 0 20px;
100    min-width: 100px;
101 }
102 </style>
103

```



代码在浏览器里的演示效果如图：

我再改变一下自定义数据，生成一个可校验的“会员用户”信息编辑表单，代码如下所示：

复制代码

```

1
2 const fieldList: DynamicFormField[] = [
3   {
4     label: '用户名称',
5     name: 'username',
6     fieldType: 'Input',
7     rule: {
8       validator: (val: unknown) => {
9         const hasError = /^[a-z]{1,}$/gi.test(`${val} || ''`) !== true;
10        return {
11          hasError,

```

```
12     message: hasError ? '仅支持a-z的大小写字母' : ''
13   };
14   }
15 }
16 },
17 {
18   label: '手机号码',
19   name: 'phone',
20   fieldType: 'Input',
21   rule: {
22     validator: (val: unknown) => {
23       const hasError = /^[0-9]{1,}$/gi.test(`${val} || ''`) !== true;
24       return {
25         hasError,
26         message: hasError ? '仅支持0-9的数字' : ''
27       };
28     }
29   }
30 },
31 {
32   label: '快递地址',
33   name: 'address',
34   fieldType: 'Input',
35   rule: {
36     validator: (val: unknown) => {
37       const hasError = `${val}`?.length === 0;
38       return {
39         hasError,
40         message: hasError ? '地址不能为空' : ''
41       };
42     }
43   }
44 },
45 {
46   label: '会员服务',
47   name: 'service',
48   fieldType: 'RadioList',
49   options: [
50     { name: '免运费', value: 'service001' },
51     { name: '9折优惠', value: 'service002' },
52     { name: '满80减10', value: 'service003' }
53   ],
54   rule: {
55     validator: (val: unknown) => {
56       const hasError = `${val}`?.length === 0;
57       return {
58         hasError,
59         message: hasError ? '优惠不能为空' : ''
60       };
61     }
62   }
63 }
```

上述使用代码的动态表单渲染如下图所示：



这两个不同的表单内容，是通过输入不同的表单配置得来的。

第一个表单是在动态表单组件里输入“普通用户”的表单配置数据（用户名称、手机号码和快递地址），渲染了普通用户的表单，也实现了对应表单的校验功能。

第二个表单，输入“会员用户”的表单配置数据（用户名称、手机号码、快递地址和会员服务），其中“会员服务”的配置数据是一个“单选表单数据组件”，附带了可选择的数据，渲染了一个与输入框不一样的表单数据组件。同时，所有表单的字段也配置了校验数据，自动地实现动态校验功能。

好，到这里，我们已经通过动态数据，大致实现了动态渲染表单和动态校验的功能。

但是，在完善动态表单的要素中，我们说的最后一点就是，要支持自定义表单数据组件的扩展，那么要怎么基于现在完善的动态表单组件，实现可扩展的动态表单组件呢？

如何实现可扩展的动态表单

从前面完善的表单组件可以看出，默认支持的表单数据组件，都是内部定义好的，存放在内部的一个对象里，这就意味着，要扩展其他自定义表单数据组件，把相应的组件配置进去就可以了。



这时候，我们需要一个“配置”的过程，一般称为“注册”，首先就是自定义表单数据组件的注册。而表单数据组件需要统一使用的 API，也就是说，我们还需要先封装好自定义组件，再把自定义表单数据组件，给注册到统一的动态表单里。

实现的代码如下：

复制代码

```
1 <template>
2   <div :class="{ [baseClassName]: true }">
3     <Form>
4       <Form
5         ref="formRef"
6         :model="internalModel"
7         @finish="onFinish"
8         @finishFail="onFinishFail"
9       >
10        <FormItem
11          v-for="(field, index) in fieldList"
12          :key="index"
13          :label="field.label"
14          :name="field.name"
15          :rule="field.rule"
16        >
17          <component
18            :is="registerComponentMap[field.fieldType]"
19            :value="internalModel[field.name]"
20            :options="field.options || []"
21            @change="(value: unknown) => { onFieldChange({ name: field.name, va
22          />
23        </FormItem>
24        <Row v-if="$slots.default">
25          <slot></slot>
26        </Row>
27      </Form>
28    </Form>
29  </div>
30 </template>
31
32 <script setup lang="ts">
33
34 // 中间省略上述演示过的代码 ....
35
36 // 内置支持的表单数据组件
```

```

37 const registerComponentMap: { [key: string]: Component } = {
38   Input: Input,
39   RadioList: RadioList
40 };
41
42 // 中间省略上述演示过的代码 ....
43
44 // 注册自定义表单数据组件方法
45 const registerFieldComponent = (name: string, component: Component) => {
46   registerComponentMap[name] = component;
47 };
48
49 // 暴露可以注册自定义表单数据组件
50 defineExpose<{
51   registerFieldComponent: typeof registerFieldComponent;
52 }>({
53   registerFieldComponent
54 });
55 </script>
56

```



代码中，我给动态表单组件添加了一个“外用方法”`registerFieldComponent`，把子自定义表单组件，注册到动态表单里。你可以从代码的注释中看出，`registerFieldComponent` 注册组件方法和内置组件缓存变量 `registerComponentMap` 的关系。

通过 `registerFieldComponent` 方法，我们可以把自定义组件缓存到 `registerComponentMap` 变量里，等待后续渲染表单时候使用。也就是说，外部组件，可以直接通过这个方法操控动态表单，将自定义组件注入到表单中。后续，只要动态的配置数据里用到了这个自定义组件，就会自动渲染出来。

至此，我们就从 API 设计到组件扩展层面，实现了一个完整的动态表单组件。

总结

通过这节课的学习，相信你应该已经理解了什么是动态表单，以及如何基于 `Vue.js 3.x` 开发自研组件库里的动态表单组件。我们总结几个重要的概念和技术实现。

动态表单的要素：

- 能通过自定义数据来配置生成表单渲染；
- 能支持扩展其他表单数据组件来扩展表单能力；

“动态表单组件”的核心技术实现就是，通过数据来动态渲染所需的表单数据组件，以及可以自定义其他数据组件。具体的实现步骤：



- 第一步，需要定义好统一的表单数据组件的 API，封装好默认支持的表单数据组件；
- 第二步，定义动态表单的配置数据格式，并且做好可以扩展的数据格式设计；
- 第三步，根据配置数据来渲染描述的表单数据组件，以及用表单字段组件进行统一管理；
- 第四步，开发自定义表单数据组件的注册能力；


思考题

动态表单能实现多种表单数据组件的渲染，那么不同表单数据组件能否做联动操作的功能配置？

欢迎积极参与讨论，如果有疑问，也欢迎在留言区留言，我会尽快回复。下一讲见。

[🔗 完整的代码在这里](#)

分享给需要的人，Ta购买本课程，你将得 **18** 元

 生成海报并分享

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

[上一篇](#) 12 | 受控表单组件：如何开发受控的表单组件？



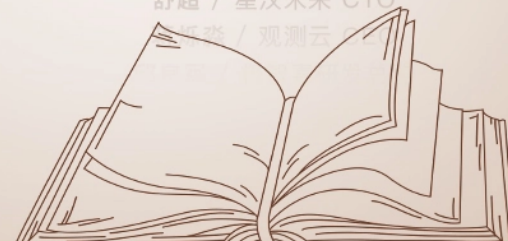
技术领导力实战笔记 2022

从实操中提升你的领导力

TGO 鲲鹏会

数十位优秀管理者的真知灼见

肖军 / 苏宁金科 CTO
王璞 / DatenLord 联合创始人
郭炜 / 前易观数据 CTO
肖德时 / 前数人云 CTO
林晓峰 / GrowingIO 副总裁
于游 / 马泷医疗集团 CTO
王植萌 / 去哪儿网高级技术总监
胡广寰 / 酷家乐技术 VP
舒超 / 星汉未来 CTO



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。