---

link: null
title: 珠峰架构师成长计划
description: mobx是一个简单可扩展的状态管理库
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=182 sentences=302, words=3091

---

# 1. mobx #

## 1.1 mobx #

mobx是一个简单可扩展的状态管理库

## 1.2. mobx vs redux #

mobx学习成本更低，性能更好的的状态解决方案

- 开发难度低
- 开发代码量少
- 渲染性能好

## 1.3. 核心思想 #

状态变化引起的副作用应该被自动触发

- 应用逻辑只需要修改状态数据即可,mobx回自动渲染UI，无需人工干预
- 数据变化只会渲染对应的组件
- MobX提供机制来存储和更新应用状态供 React 使用
- react 通过提供机制把应用状态转换为可渲染组件树并对其进行渲染

## 1.4. mobx #

observable

- MobX为现有的数据结构(如对象，数组和类实例)添加了可观察的功能。
- `observable`就是一种让数据的变化可以被观察的方法
- 先把数据转化成可以被观察的对象，那么对这些数据的修改就可以备监视

### 1.4.1 引用类型 (observable) #

```
import { observable } from 'mobx';
const pl = observable([1, 2, 3]);
pl.push(4);
pl.pop();
console.log(pl);
console.log(Array.isArray(pl));
```

### 1.4.2 基本类型(observable.box) #

类型 描述 String 字符串 Boolean 布尔值 Number 数字 Symbol 独一无二的值

```
import { observable } from 'mobx';
let num = observable.box(10);
let str = observable.box('hello');
let bool = observable.box(true);
console.log(num.get(), str.get(), bool.get());
num.set(100);
str.set('world');
bool.set(false);
console.log(num.get(), str.get(), bool.get());
```

# 2.formily #

- formily (https://formilyjs.org)是一款面向中后台复杂场景的数据+协议驱动的表单框架，也是阿里巴巴集团统一表单解决方案,可以完成复杂表单需求，而且提供了表单设计器让我们快速设计表单

## 2.1 核心优势 #

- 高性能 字段数据极多的情况下保持快速响应，可以实现高效联动逻辑
- 跨端能力 与框架无关，可以兼容 `react`和 `vue`等框架
- 生态完备 支持了业界主流的 `antd`和 `element`等组件库
- 协议驱动 可以通过JSON驱动表单渲染，可以成为领域视图模型驱动的低代码渲染引擎

## 2.2 分层架构 #

- @formily/core (https://core.formilyjs.org/zh-CN)负责管理表单的状态、校验和联动等
- @formily/react (https://react.formilyjs.org/zh-CN/guide)是UI桥接库，用来接入内核数据实现最终的表单交互效果，不同框架有不同的桥接库
- @formily/antd (https://antd.formilyjs.org/zh-CN/components)封装了场景化的组件

- 这张图主要将 Formily 分为了内核协议层，UI胶水桥接层，扩展组件层，和配置应用层
- 内核层是 UI 无关的，它保证了用户管理的逻辑和状态是不耦合任何一个框架
- JSON Schema 独立存在，给 UI 桥接层消费，保证了协议驱动在不同 UI 框架下的绝对一致性，不需要重复实现协议解析逻辑
- 扩展组件层，提供一系列表单场景化组件，保证用户开箱即用。无需花大量时间做二次开发

## 2.3 竞品对比 #

## 2.4 安装 #

```
npm init vite@latest
npm install @formily/reactive @formily/core @formily/reactive-react @formily/react @formily/antd ajv less --save
```

## 2.5 配置 #

- jsxRuntime (https://github.com/vitejs/vite/tree/main/packages/plugin-react#opting-out-of-the-automatic-jsx-runtime)
- 在 ``less``` 文件中引入 `antd` 的 `less` 文件会有一个 ~前置符，这种写法对于 ESM 构建工具是不兼容的
- `javascriptEnabled`这个参数在less3.0之后是默认为false

vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
+ plugins: [react({
+   jsxRuntime: 'classic'
+ })],
+ resolve: {
+   alias: [
+     { find: /^~/, replacement: '' }
+   ]
+ },
+ css: {
+   preprocessorOptions: {
+     less: {
+       // 支持内联 JavaScript
+       javascriptEnabled: true,
+     }
+   }
+ }
})
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "ESNext",
    "useDefineForClassFields": true,
    "lib": ["DOM", "DOM.Iterable", "ESNext"],
    "allowJs": false,
    "skipLibCheck": true,
    "esModuleInterop": false,
    "allowSyntheticDefaultImports": true,
+   "strict": false,
+   "noImplicitAny": false,
    "forceConsistentCasingInFileNames": true,
    "module": "ESNext",
    "moduleResolution": "Node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": ["src"],
  "references": [{ "path": "./tsconfig.node.json" }]
}
```

## 3. 字段数量多 #

### 3.1 问题 #

- 字段数量多，如何让性能不随字段数量增加而变差？

### 3.2 解决方案 #

- 依赖@formily/reactive (https://reactive.formilyjs.org/zh-CN)响应式解决方案，构建响应式表单的领域模型实现精确渲染

#### 3.2.1 MVVM #

- MVVM（Model–view–viewmodel）是一种 OOP 软件架构模式，它的核心是将我们的应用程序的逻辑与视图做分离，提升代码可维护性与应用健壮性
- View(视图层)负责维护 UI 结构与样式，同时负责与 ViewModel(视图模型)做数据绑定
- 这里的数据绑定关系是双向的，也就是，ViewModel(视图模型)的数据发生变化，会触发 View(视图层)的更新，同时视图层的数据变化又会触发 ViewModel(视图模型)的变化,Model 则更偏实际业务数据处理模型
- ViewModel 和 Model 都是充血模型，两者都注入了不同领域的业务逻辑，比如 ViewModel 的业务逻辑更偏视图交互层的领域逻辑，而 Model 的业务逻辑则更偏业务数据的处理逻辑
- Formily 它提供了 `View` 和 `ViewModel` 两层能力， `View` 则是 `@formily/react`，专门用来与 `@formily/core` 做桥接通讯的，所以， `@formily/core` 的定位就是 `ViewModel` 层

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MVVMtitle>
head>
<body>
    <input id="bookTitle" value="红楼梦"/>
    <script>
        class Book{
            constructor(title){
                this.title=title;
            }
        }

        let book = new Book('红楼梦');

        let viewModel = {display:'block'};
        Object.defineProperty(viewModel,'title',{
                get(){
                    return book.title;
                },
                set(newTitle){
                    bookTitle.value = book.title = newTitle;
                }
        });
        Object.defineProperty(viewModel,'display',{
                get(){
                    return bookTitle.style.display;
                },
                set(display){
                    bookTitle.style.display = display;
                }
        });

        viewModel.title='新红楼梦';
        setTimeout(()=>{
            viewModel.display= 'none';
        },3000);

        bookTitle.onchange = (event)=>{
            viewModel.title = event.target.value;
        }
    script>
body>
html>
```

#### 3.2.2 observable #

- observable (https://reactive.formilyjs.org/zh-CN/api/observable)主要用于创建不同响应式行为的 observable 对象
- 一个 observable对象，字面意思是可订阅对象，我们通过创建一个可订阅对象，在每次操作该对象的属性数据的过程中，会自动通知订阅者
- @formily/reactive (https://reactive.formilyjs.org/zh-CN) 创建observable (https://reactive.formilyjs.org/zh-CN/api/observable) 对象主要是通过 ES Proxy 来创建的，它可以做到完美劫持数据操作

#### 3.2.2 Reaction #

- `reaction (https://reactive.formilyjs.org/zh-CN/api/reaction)在响应式编程模型中，它就相当于是可订阅对象的订阅者
- 它接收一个 tracker 函数，这个函数在执行的时候，如果函数内部有对 observable 对象中的某个属性进行读操作会进行依赖收集，那当前 reaction 就会与该属性进行一个绑定(依赖追踪)，该属性在其它地方发生了写操作，就会触发 tracker 函数重复执行
- 从订阅到派发订阅，其实是一个封闭的循环状态机，每次 tracker 函数执行的时候都会重新收集依赖，依赖变化时又会重新触发 tracker执行

#### 3.2.3 autorun #

- autorun (https://reactive.formilyjs.org/zh-CN/api/autorun)可以创建一个自动执行的响应器
- 接收一个 tracker 函数，如果函数内部有消费 observable 数据，数据发生变化时， tracker 函数会重复执行

#### 3.2.4 实现observable #

##### 3.2.4.1 src\main.tsx #

src\main.tsx

```
import { observable, autorun } from './@formily/reactive'
const obs = observable({
  name: 'zhu',
})
const tracker = () => {
  console.log(obs.name);
}
autorun(tracker)
obs.name = 'feng';
```

```
import { observable, autorun } from '@formily/reactive'
const obs = observable({
  name: 'zhu',
+ age: 12
})
+let counter = 0;
const tracker = () => {
  console.log(obs.name);
+ if (counter++) {
+   console.log(obs.age);
+ }
}
autorun(tracker)
+obs.age = 13;
obs.name = 'feng';
+obs.age = 14;
/**
tracker第1次执行
zhu
tracker第2次执行
feng
13
tracker第3次执行
feng
14
 */
```

#### 3.2.4.2 reactive\index.ts #

src@formily\reactive\index.ts

```
const RawReactionsMap = new WeakMap()
let currentReaction;
export function observable(value) {
    return new Proxy(value, baseHandlers)
}
export const autorun = (tracker) => {
    const reaction = () => {
        currentReaction = reaction;
        tracker()
        currentReaction = null;
    }
    reaction()
}
const baseHandlers = {
    get(target, key) {
        const result = target[key]
        if (currentReaction) {
            addRawReactionsMap(target, key, currentReaction)
        }
        return result
    },
    set(target, key, value) {
        target[key] = value
        RawReactionsMap.get(target)?.get(key)?.forEach((reaction) => reaction())
        return true;
    }
}
const addRawReactionsMap = (target, key, reaction) => {
    const reactionsMap = RawReactionsMap.get(target)
    if (reactionsMap) {
        const reactions = reactionsMap.get(key)
        if (reactions) {
            reactions.push(reaction)
        } else {
            reactionsMap.set(key, [reaction])
        }
        return reactionsMap
    } else {
        const reactionsMap = new Map()
        reactionsMap.set(key, [reaction]);
        RawReactionsMap.set(target, reactionsMap)
        return reactionsMap
    }
}
```

### 3.2.5 Observer #

- Observer (https://reactive.formilyjs.org/zh-CN/api/react/observer)接收一个 `Function RenderProps`，只要在 `Function` 内部消费到的任何响应式数据，都会随数据变化而自动重新渲染，也更容易实现局部精确渲染

#### 3.2.5.1 src\main.tsx #

src\main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App';
ReactDOM.render(<App />, document.getElementById('root')!);
```

#### 3.2.5.2 src\App.tsx #

src\App.tsx

```
import { observable } from './@formily/reactive'
import { Observer } from './@formily/reactive-react'
const username = observable({ value: 'zhangsan' })
const age = observable({ value: 14 })
export default () => {
  return (
    <>
      <Observer>
        {() => (
          <input
            value={username.value}
            onChange={(event) => {
              username.value = event.target.value
            }}
          />
        )}
      Observer>
      <Observer>{() => {
        console.log('username render');
        return <div>{username.value}div>;
      }}Observer>
      <Observer>
        {() => (
          <input
            value={age.value}
            onChange={(event) => {
              age.value = +event.target.value
            }}
          />
        )}
      Observer>
      <Observer>{() => {
        console.log('age render');
        return <div>{age.value}div>;
      }}Observer>
    </>
  )
}
```

#### 3.2.5.3 reactive-react\index.tsx #

src@formily\reactive-react\index.tsx

```
import React, { useReducer } from 'react';
import { Tracker } from '../../@formily/reactive'
export const Observer = (props) => {
    const [, forceUpdate] = useReducer(x => x + 1, 0)
    const trackerRef = React.useRef(null)
    if (!trackerRef.current)
        trackerRef.current = new Tracker(forceUpdate)
    return trackerRef.current.track(props.children)
}
```

#### 3.2.5.4 reactive\index.ts #

src@formily\reactive\index.ts

```
const RawReactionsMap = new WeakMap()
let currentReaction;
export function observable(value) {
    return new Proxy(value, baseHandlers)
}
export const autorun = (tracker) => {
    const reaction = () => {
        currentReaction = reaction;
        tracker()
        currentReaction = null;
    }
    reaction()
}
const baseHandlers = {
    get(target, key) {
        const result = target[key]
        if (currentReaction) {
            addRawReactionsMap(target, key, currentReaction)
        }
        return result
    },
    set(target, key, value) {
        target[key] = value
        RawReactionsMap.get(target)?.get(key)?.forEach((reaction) => {
+           if (typeof reaction._scheduler === 'function') {
+               reaction._scheduler()
+           } else {
+               reaction()
+           }
        })
        return true;
    }
}
const addRawReactionsMap = (target, key, reaction) => {
    const reactionsMap = RawReactionsMap.get(target)
    if (reactionsMap) {
        const reactions = reactionsMap.get(key)
        if (reactions) {
            reactions.push(reaction)
        } else {
            reactionsMap.set(key, [reaction])
        }
        return reactionsMap
    } else {
        const reactionsMap = new Map()
        reactionsMap.set(key, [reaction]);
        RawReactionsMap.set(target, reactionsMap)
        return reactionsMap
    }
}

+export class Tracker {
+    constructor(scheduler) {
+        this.track._scheduler = scheduler
+    }
+    track: any = (tracker) => {
+        currentReaction = this.track;
+        return tracker()
+    }
+}
```

## 4. 字段关联逻辑复杂 #

### 4.1 问题 #

- 字段关联逻辑复杂，如何更简单的实现复杂的联动逻辑？字段与字段关联时，如何保证不影响表单性能？
    - 一对多(异步)
    - 多对一(异步)
    - 多对多(异步)

### 4.2 领域模型 #

- 字段值的改变和应用状态、服务器返回数据等都可能会引发字段的联动
- 联动关系核心是将字段的某些状态属性与某些数据关联起来
- 可以定义针对表单领域的 `&#x9886;&#x57DF;&#x6A21;&#x578B;`
- Form (https://core.formilyjs.org/zh-CN/api/models/form)是调用 `createForm`所返回的核心表单模型
- Field (https://core.formilyjs.org/zh-CN/api/models/field)是调用 `createField`所返回的字段模型
- createForm (https://core.formilyjs.org/zh-CN/api/entry/create-form)用来创建表单核心领域模型，它是作为MVVM设计模式的标准 `ViewModel`

src\main.tsx

```
import { createForm } from '@formily/core'
const form = createForm()
const field = form.createField({ name: 'target' })
```

### 4.3 DDD(领域驱动) #

- DDD(Domain-Driven Design)即领域驱动设计是思考问题的方法论,用于对实际问题建模
- 它以一种领域专家、设计人员、开发人员都能理解的通用语言作为相互交流的工具，然后将这些概念设计成一个领域模型。由领域模型驱动软件设计，用代码来实现该领域模型

#### 4.3.1 表单 #

```
interface Form {
   values,
   visible,
   submit()
}
```

#### 4.3.2 字段 #

```
interface Field {
   value,
   visible,
   setValue()
}
```

### 4.4 路径系统 #

- 表单模型作为顶层模型管理着所有字段模型，每个字段都有着自己的路径
- 如何优雅的查找某个字段?
- Formily 独创的路径系统@formily/path (https://core.formilyjs.org/zh-CN/api/entry/form-path)让字段查找变得优雅
- FormPath 在 Formily 中核心是解决路径匹配问题和数据操作问题

src\main.tsx

```
import { FormPath } from '@formily/core'
const target = { array: [] }

FormPath.setIn(target, 'a.b.c', 'dotValue')
console.log(FormPath.getIn(target, 'a.b.c'))

FormPath.setIn(target, 'array.0.d', 'arrayValue')
console.log(FormPath.getIn(target, 'array.0.d'))

FormPath.setIn(target, 'parent.[f,g]', [1, 2])

console.log(JSON.stringify(target))
```

## 5. 生命周期 #

### 5.1 问题 #

- 响应式和路径系统组成一个较为完备的表单方案,但是一个黑盒
- 想要在某个过程阶段内实现一些自定义逻辑如何实现?

### 5.2 解决方案 #

- Form Effect Hooks (https://core.formilyjs.org/zh-CN/api/entry/form-effect-hooks)可以将整个表单生命周期作为事件钩子暴露给外界，这样就能做到了既有抽象，但又灵活的表单方案
- onFormInit (https://core.formilyjs.org/zh-CN/api/entry/form-effect-hooks#onforminit)用于监听某个表单初始化的副作用钩子，我们在调用 createForm 的时候就会触发初始化事件
- onFormReact (https://core.formilyjs.org/zh-CN/api/entry/form-effect-hooks#%E6%8F%8F%E8%BF%B0-3)用于实现表单响应式逻辑的副作用钩子，它的核心原理就是表单初始化的时候会执行回调函数，同时自动追踪依赖，依赖数据发生变化时回调函数会重复执行

```
import { useMemo, useState } from 'react'
import { createForm, onFormInit, onFormReact } from '@formily/core'
export default () => {
  const [state, setState] = useState('init')
  const form = useMemo(
    () =>
      createForm({
        effects() {
          onFormInit(() => {
            setState('表单已初始化')
          })
          onFormReact((form) => {
            if (form.values.input == 'Hello') {
              setState('响应Hello')
            } else if (form.values.input == 'World') {
              setState('响应World')
            }
          })
        },

      }),
    []
  )
  return (
    <div>
      <p>{state}p>
      <button
        onClick={() => {
          form.setValuesIn('input', 'Hello')
        }}
      >
        Hello
      button>
      <button
        onClick={() => {
          form.setValuesIn('input', 'World')
        }}
      >
        World
      button>
    div>
  )
}
```

## 6 协议驱动 #

### 6.1 问题 #

- 动态渲染述求很强烈
    - 字段配置化，让非专业前端也能快速搭建复杂表单
    - 跨端渲染，一份 JSON Schema，多端适配
    - 如何在表单协议中描述布局?
        - 纵向布局
        - 横向布局
        - 网格布局
        - 弹性布局
        - 自由布局

- 如何在表单协议中描述逻辑？

### 6.2 解决方案 #

- 表单场景的数据协议最流行就是JSON-Schema (https://json-schema.org/)
- 定义一套通用协议，简单高效的描述表单逻辑，适合开发低代码

### 6.3 JSON-Schema #

- JSON-Schema (https://json-schema.org)以数据描述视角驱动UI渲染，不好描述UI
- ajv (https://ajv.js.org)是一个JSON Schema验证器

```
import Ajv from 'ajv';
const ajv = new Ajv()

const schema = {
  type: "object",
  properties: {
    foo: { type: "integer" },
    bar: { type: "string" }
  },
  required: ["foo"],
  additionalProperties: false
}

const validate = ajv.compile(schema)

const data = {
  foo: 1,
  bar: "abc",
  age: 1
}

const valid = validate(data)
if (!valid)
  console.log(validate.errors)
```

### 6.4 扩展的JSON-Schema #

- Formily扩展了 JSON-Schema 属性，统一以 x-*格式来表达扩展属性以描述数据无关的布局容器和控件，实现UI协议与数据协议混合在一起
- JSON Schema 引入 void，代表一个虚数据节点，表示该节点并不占用实际数据结构
- DSL(领域特定语言)(Domain Specific Language)是针对某一领域，具有受限表达性的一种计算机程序设计语言

```
{
  "type": "string",
  "title": "字符串",
  "description": "这是一个字符串",
  "x-component": "Input",
  "x-component-props": {
    "placeholder": "请输入"
  }
}
```

```
{
  "type": "void",
  "title": "卡片",
  "description": "这是一个卡片",
  "x-component": "Card",
  "properties": {
    "name": {
      "type": "string",
      "title": "字符串",
      "description": "这是一个字符串",
      "x-component": "Input",
      "x-component-props": {
        "placeholder": "请输入"
      }
    }
  }
}
```

### 6.5 API #

- createForm (https://core.formilyjs.org/zh-CN/api/entry/create-form)创建一个 Form 实例，作为 ViewModel 给 UI 框架层消费
  - effects 副作用逻辑，用于实现各种联动逻辑
  - onFieldMount (https://core.formilyjs.org/zh-CN/api/entry/field-effect-hooks#onfieldmount)用于监听某个字段已挂载的副作用钩子
  - onFieldValueChange (https://core.formilyjs.org/zh-CN/api/entry/field-effect-hooks#onfieldvaluechange)用于监听某个字段值变化的副作用钩子
  - setFieldState (https://core.formilyjs.org/zh-CN/api/models/form#setfieldstate)可以设置字段状态
- core/Field (https://core.formilyjs.org/zh-CN/api/models/field)组件是用来承接普通字段的组件
- react/Field (https://react.formilyjs.org/zh-CN/api/components/field)作为 @formily/core 的 createField React 实现，它是专门用于将 ViewModel 与输入控件做绑定的桥接组件
  - title (https://core.formilyjs.org/zh-CN/api/models/field#%E5%B1%9E%E6%80%A7)字段标题
  - required (https://core.formilyjs.org/zh-CN/api/models/field#%E5%B1%9E%E6%80%A7)字段是否必填,如果 decorator 指定为 FormItem，那么会自动出现星号提示
  - component (https://core.formilyjs.org/zh-CN/api/models/field#%E5%B1%9E%E6%80%A7)字段组件,注意 component 属性传递的是数组形式，第一个参数代表指定组件类型，第二个参数代表指定组件属性
  - decorator (https://core.formilyjs.org/zh-CN/api/models/field#%E5%B1%9E%E6%80%A7)字段装饰器,通常我们都会指定为 FormItem,注意 decorator 属性传递的是数组形式，第一个参数代表指定组件类型，第二个参数代表指定组件属性
- SchemaField组件是专门用于解析 JSON-Schema动态渲染表单的组件。 在使用 SchemaField组件的时候，需要通过createSchemaField (https://react.formilyjs.org/zh-CN/api/components/schema-field)工厂函数创建一个 ``SchemaField``` 组件
- Schema (https://react.formilyjs.org/zh-CN/api/shared/schema)是 @formily/react协议驱动最核心的部分
  - 解析 json-schema 的能力
  - 将 json-schema 转换成 Field Model 的能力
  - 编译 json-schema 表达式的能力
  - x-component 的组件标识与 createSchemaField传入的组件集合的 Key 匹配
  - x-decorator 的组件标识与 createSchemaField传入的组件集合的 Key 匹配
  - Schema 的每个属性都能使用字符串表达式 {{expression}}，表达式变量可以从 createSchemaField 中传入，也可以从 SchemaField 组件中传入
- Schema (https://react.formilyjs.org/zh-CN/api/shared/schema)属性
  - type (https://react.formilyjs.org/zh-CN/api/shared/schema)类型
  - properties (https://react.formilyjs.org/zh-CN/api/shared/schema)属性描述
  - title (https://react.formilyjs.org/zh-CN/api/shared/schema)标题
  - required (https://react.formilyjs.org/zh-CN/api/shared/schema)必填

- x-decorator (https://react.formilyjs.org/zh-CN/api/shared/schema)字段 UI 包装器组件
- x-component (https://react.formilyjs.org/zh-CN/api/shared/schema)字段 UI 组件属性
- x-component-props (https://react.formilyjs.org/zh-CN/api/shared/schema)字段 UI 组件属性
- x-reactions (https://react.formilyjs.org/zh-CN/api/shared/schema#schemareactions)字段联动协议
- $deps (https://react.formilyjs.org/zh-CN/api/shared/schema#deps)只能在 x-reactions中的表达式消费，与 x-reactions``` 定义的dependencies``` 对应，数组顺序一致
- $self (https://react.formilyjs.org/zh-CN/api/shared/schema#self)代表当前字段实例，可以在普通属性表达式中使用，也能在 x-reactions 中使用

### 6.5 表单渲染 #

- Formily 的表单校验使用了极其强大且灵活的FieldValidator (https://core.formilyjs.org/zh-CN/api/models/field#fieldvalidator)校验引擎，校验主要分两种场景：
    - 纯 JSX 场景校验属性，使用 validator 属性实现校验
    - Markup(JSON) Schema场景协议校验属性校验，使用 JSON Schema 本身的校验属性与 x-validator 属性实现校验

#### 6.5.1 JSX 案例 #

src\App.tsx

```
import { createForm } from '@formily/core'
import { Field } from '@formily/react'
import 'antd/dist/antd.css'
import { Form, FormItem, Input, NumberPicker } from '@formily/antd'

const form = createForm()
function App() {
  return (
    <Form form={form} labelCol={6} wrapperCol={10}>
      <Field
        name="name"
        title="姓名"
        required
        component={[Input]}
        decorator={[FormItem]}
      />
      <Field
        name="age"
        title="年龄"
        validator={{ maximum: 5 }}
        component={[NumberPicker]}
        decorator={[FormItem]}
      />
    Form>
  )
}
export default App;
```

#### 6.5.2 JSON Schema案例 #

- schema (https://react.formilyjs.org/zh-CN/api/shared/schema)是 @formily/react协议驱动最核心的部分
    - 解析 json-schema 的能力
    - 将 json-schema 转换成 Field Model 的能力
    - 编译 json-schema 表达式的能力

src\App.tsx

```
import { createForm } from '@formily/core'
import { createSchemaField } from '@formily/react'
import 'antd/dist/antd.css'
import { Form, FormItem, Input } from '@formily/antd'
const form = createForm()
const SchemaField = createSchemaField({
  components: {
    FormItem,
    Input
  },
})
const schema = {
  type: 'object',
  properties: {
    name: {
      title: `姓名`,
      type: 'string',
      required: true,
      'x-decorator': 'FormItem',
      'x-component': 'Input',
    },
    age: {
      title: `邮箱`,
      type: 'string',
      required: true,
      'x-validator': 'email',
      'x-decorator': 'FormItem',
      'x-component': 'Input',
    },
  },
}
function App() {
  return (
    <Form form={form} labelCol={6} wrapperCol={10}>
      <SchemaField schema={schema} />
    Form>
  )
}
export default App;
```

#### 6.5.3 Markup Schema 案例 #

src\App.tsx

```
import { createForm } from '@formily/core'
import { createSchemaField } from '@formily/react'
import 'antd/dist/antd.css'
import { Form, FormItem, Input, NumberPicker } from '@formily/antd'
const form = createForm()
const SchemaField = createSchemaField({
  components: {
    Input,
    FormItem,
    NumberPicker
  },
})

function App() {
  return (
    <Form form={form} labelCol={6} wrapperCol={10}>
      <SchemaField>
        <SchemaField.String
          name="name"
          title="姓名"
          required
          x-component="Input"//字段 UI 组件属性
          x-decorator="FormItem"//字段 UI 包装器组件
        />
        <SchemaField.Number
          name="age"
          title="年龄"
          maximum={120}
          x-component="NumberPicker"//字段 UI 组件属性
          x-decorator="FormItem"//字段 UI 包装器组件
        />
      SchemaField>
    Form>
  )
}
export default App;
```

### 6.6 联动校验 #

- 同时我们还能在 `effects` 或者 `x-reactions` 中实现联动校验

#### 6.6.1 主动联动 #

- Schema 联动协议 (https://react.formilyjs.org/zh-CN/api/shared/schema#schemareactions)，如果 `reaction`对象里包含 `target`，则代表 `&#x4E3B;&#x52A8;`联动模式，否则代表 `&#x88AB;&#x52A8;`联动模式

src\App.tsx

```
import { createForm } from '@formily/core'
import { createSchemaField } from '@formily/react'
import 'antd/dist/antd.css'
import { Form, FormItem, Input } from '@formily/antd'
const form = createForm()
const SchemaField = createSchemaField({
  components: {
    FormItem,
    Input
  },
})
const schema = {
  type: 'object',
  properties: {
    source: {
      title: `来源`,
      type: 'string',
      required: true,
      'x-decorator': 'FormItem',
      'x-component': 'Input',
      "x-component-props": {
        "placeholder": "请输入"
      },
      "x-reactions": [
        {
          "target": "target",

          "when": "{{$self.value == '123'}}",
          "fulfill": {
            "state": {
              "visible": true
            }
          },
          "otherwise": {
            "state": {
              "visible": false
            }
          }
        }
      ]
    },
    target: {
      "title": "目标",
      "type": "string",
      "x-component": "Input",
      "x-component-props": {
        "placeholder": "请输入"
      },
      'x-decorator': 'FormItem'
    }
  },
}
function App() {
  return (
    <Form form={form} labelCol={6} wrapperCol={10}>
      <SchemaField schema={schema} />
    Form>
  )
}
export default App;
```

#### 6.6.2 被动联动 #

src\App.tsx

```
import { createForm } from '@formily/core'
import { createSchemaField } from '@formily/react'
import 'antd/dist/antd.css'
import { Form, FormItem, Input } from '@formily/antd'
const form = createForm()
const SchemaField = createSchemaField({
  components: {
    FormItem,
    Input
  },
})
const schema = {
  type: 'object',
  properties: {
    source: {
      title: `来源`,
      type: 'string',
      required: true,
      'x-decorator': 'FormItem',
      'x-component': 'Input',
      "x-component-props": {
        "placeholder": "请输入"
      }
    },
    target: {
      "title": "目标",
      "type": "string",
      "x-component": "Input",
      "x-component-props": {
        "placeholder": "请输入"
      },
      'x-decorator': 'FormItem',
      "x-reactions": [
        {
          "dependencies": ["source"],

          "when": "{{$deps[0] == '123'}}",
          "fulfill": {
            "state": {
              "visible": true
            }
          },
          "otherwise": {
            "state": {
              "visible": false
            }
          }
        }
      ]
    }
  },
}
function App() {
  return (
    <Form form={form} labelCol={6} wrapperCol={10}>
      <SchemaField schema={schema} />
    Form>
  )
}
export default App;
```

#### 6.6.3 effects #

src\App.tsx

```
import { createForm, onFieldMount, onFieldValueChange } from '@formily/core'
import { createSchemaField } from '@formily/react'
import 'antd/dist/antd.css'
import { Form, FormItem, Input } from '@formily/antd'
const form = createForm({
  effects() {

    onFieldMount('target', (field: any) => {

      form.setFieldState(field.query('target'), (state) => {
        if (field.value === '123') {
          state.visible = true;
        } else {
          state.visible = false;
        }
      })
    })

    onFieldValueChange('source', (field: any) => {
      form.setFieldState(field.query('target'), (state) => {
        if (field.value === '123') {
          state.visible = true;
        } else {
          state.visible = false;
        }
      })
    })
  },
})
const SchemaField = createSchemaField({
  components: {
    FormItem,
    Input
  },
})
const schema = {
  type: 'object',
  properties: {
    source: {
      title: `来源`,
      type: 'string',
      required: true,
      'x-decorator': 'FormItem',
      'x-component': 'Input',
      "x-component-props": {
        "placeholder": "请输入"
      }
    },
    target: {
      "title": "目标",
      "type": "string",
      'x-decorator': 'FormItem',
      "x-component": "Input",
      "x-component-props": {
        "placeholder": "请输入"
      }
    }
  }
}
function App() {
  return (
    <Form form={form} labelCol={6} wrapperCol={10}>
      <SchemaField schema={schema} />
    Form>
  )
}
export default App;
```

## 7.案例 #

### 7.1.注册 #

```
import React from 'react'
import { createForm } from '@formily/core'
import { createSchemaField } from '@formily/react'
import {
  Form,
  FormItem,
  FormLayout,
  Input,
  Select,
  Password,
  Cascader,
  DatePicker,
  Submit,
  Space,
  FormGrid,
  Upload,
  ArrayItems,
  Editable,
  FormButtonGroup,
} from '@formily/antd'
import { action } from '@formily/reactive'
import { Card, Button } from 'antd'
import { UploadOutlined } from '@ant-design/icons'

const form = createForm({
  validateFirst: true,
})

const IDUpload = (props) => {
  return (
    <Upload
```

```
      {...props}
      action="https://www.mocky.io/v2/5cc8019d300000980a055e76"
      headers={{
        authorization: 'authorization-text',
      }}
    >
      <Button icon={<UploadOutlined />}>上传复印件Button>
    Upload>
  )
}

const SchemaField = createSchemaField({
  components: {
    FormItem,
    FormGrid,
    FormLayout,
    Input,
    DatePicker,
    Cascader,
    Select,
    Password,
    IDUpload,
    Space,
    ArrayItems,
    Editable,
  },
  scope: {
    fetchAddress: (field) => {
      const transform = (data = {}) => {
        return Object.entries(data).reduce((buf, [key, value]) => {
          if (typeof value === 'string')
            return buf.concat({
              label: value,
              value: key,
            })
          const { name, code, cities, districts } = value
          const _cities = transform(cities)
          const _districts = transform(districts)
          return buf.concat({
            label: name,
            value: code,
            children: _cities.length
              ? _cities
              : _districts.length
              ? _districts
              : undefined,
          })
        }, [])
      }

      field.loading = true
      fetch('//unpkg.com/china-location/dist/location.json')
        .then((res) => res.json())
        .then(
          action.bound((data) => {
            field.dataSource = transform(data)
            field.loading = false
          })
        )
    },
  },
})

export default () => {
  return (
    <div
      style={{
        display: 'flex',
        justifyContent: 'center',
        background: '#eee',
        padding: '40px 0',
      }}
    >
      <Card title="新用户注册" style={{ width: 620 }}>
        <Form
          form={form}
          labelCol={5}
          wrapperCol={16}
          onAutoSubmit={console.log}
        >
          <SchemaField>
            <SchemaField.String
              name="username"
              title="用户名"
              required
              x-decorator="FormItem"
              x-component="Input"
            />
            <SchemaField.String
              name="password"
              title="密码"
              required
              x-decorator="FormItem"
              x-component="Password"
              x-component-props={{
                checkStrength: true,
              }}
              x-reactions={[
                {
                  dependencies: ['.confirm_password'],
                  fulfill: {
                    state: {
                      selfErrors:
                        '{{$deps[0] && $self.value && $self.value !== $deps[0] ? "确认密码不匹配" : ""}}',
```

```
        },
      },
    },
  ]}
/>
<SchemaField.String
  name="confirm_password"
  title="确认密码"
  required
  x-decorator="FormItem"
  x-component="Password"
  x-component-props={{
    checkStrength: true,
  }}
  x-reactions={[
    {
      dependencies: ['.password'],
      fulfill: {
        state: {
          selfErrors:
            '{{$deps[0] && $self.value && $self.value !== $deps[0] ? "确认密码不匹配" : ""}}',
        },
      },
    },
  ]}
/>
<SchemaField.Void
  title="姓名"
  x-decorator="FormItem"
  x-decorator-props={{
    asterisk: true,
    feedbackLayout: 'none',
  }}
  x-component="FormGrid"
>
  <SchemaField.String
    name="firstName"
    x-decorator="FormItem"
    x-component="Input"
    x-component-props={{
      placeholder: '姓',
    }}
    required
  />
  <SchemaField.String
    name="lastName"
    x-decorator="FormItem"
    x-component="Input"
    x-component-props={{
      placeholder: '名',
    }}
    required
  />
SchemaField.Void>
<SchemaField.String
  name="email"
  title="邮箱"
  required
  x-validator="email"
  x-decorator="FormItem"
  x-component="Input"
/>
<SchemaField.String
  name="gender"
  title="性别"
  x-decorator="FormItem"
  x-component="Select"
  enum={[
    {
      label: '男',
      value: 1,
    },
    {
      label: '女',
      value: 2,
    },
    {
      label: '第三性别',
      value: 3,
    },
  ]}
  required
/>
<SchemaField.String
  name="birthday"
  title="生日"
  required
  x-decorator="FormItem"
  x-component="DatePicker"
/>
<SchemaField.String
  name="address"
  title="地址"
  required
  x-decorator="FormItem"
  x-component="Cascader"
  x-reactions="{{fetchAddress}}"
/>
<SchemaField.String
  name="idCard"
  title="身份证复印件"
  required
  x-decorator="FormItem"
  x-component="IDUpload"
```

```
          />
          <SchemaField.Array
            name="contacts"
            title="联系人信息"
            required
            x-decorator="FormItem"
            x-component="ArrayItems"
          >
            <SchemaField.Object x-component="ArrayItems.Item">
              <SchemaField.Void
                x-decorator="FormItem"
                x-component="ArrayItems.SortHandle"
              />
              <SchemaField.Void
                name="popover"
                title="维护联系人信息"
                x-decorator="Editable.Popover"
                x-component="FormLayout"
                x-component-props={{
                  layout: 'vertical',
                }}
                x-reactions={[
                  {
                    dependencies: ['.popover.name'],
                    fulfill: {
                      schema: {
                        title: '{{$deps[0]}}',
                      },
                    },
                  },
                ]}
              >
                <SchemaField.String
                  name="name"
                  required
                  title="姓名"
                  x-decorator="FormItem"
                  x-component="Input"
                  x-component-props={{
                    style: {
                      width: 300,
                    },
                  }}
                />
                <SchemaField.String
                  name="email"
                  title="邮箱"
                  x-validator={[{ required: true }, 'email']}
                  x-decorator="FormItem"
                  x-component="Input"
                  x-component-props={{
                    style: {
                      width: 300,
                    },
                  }}
                />
                <SchemaField.String
                  name="phone"
                  required
                  title="手机号"
                  x-validator="phone"
                  x-decorator="FormItem"
                  x-component="Input"
                  x-component-props={{
                    style: {
                      width: 300,
                    },
                  }}
                />
              SchemaField.Void>
              <SchemaField.Void
                x-decorator="FormItem"
                x-component="ArrayItems.Remove"
              />
            SchemaField.Object>
            <SchemaField.Void
              x-component="ArrayItems.Addition"
              title="新增联系人"
            />
          SchemaField.Array>
        SchemaField>
        <FormButtonGroup.FormItem>
          <Submit block size="large">
            注册
          Submit>
        FormButtonGroup.FormItem>
      Form>
    Card>
  div>
 )
}
```

### 7.2 登录 #

```
import React from 'react'
import { createForm } from '@formily/core'
import { createSchemaField } from '@formily/react'
import { Form, FormItem, Input, Password, Submit } from '@formily/antd'
import { Tabs, Card } from 'antd'
import * as ICONS from '@ant-design/icons'
import { VerifyCode } from './VerifyCode'

const normalForm = createForm({
  validateFirst: true,
```

```
})

const phoneForm = createForm({
  validateFirst: true,
})

const SchemaField = createSchemaField({
  components: {
    FormItem,
    Input,
    Password,
    VerifyCode,
  },
  scope: {
    icon(name) {
      return React.createElement(ICONS[name])
    },
  },
})

export default () => {
  return (
    <div
      style={{
        display: 'flex',
        justifyContent: 'center',
        background: '#eee',
        padding: '40px 0',
      }}
    >
      <Card style={{ width: 400 }}>
        <Tabs style={{ overflow: 'visible', marginTop: -10 }}>
          <Tabs.TabPane key="1" tab="账密登录">
            <Form
              form={normalForm}
              layout="vertical"
              size="large"
              onAutoSubmit={console.log}
            >
              <SchemaField>
                <SchemaField.String
                  name="username"
                  title="用户名"
                  required
                  x-decorator="FormItem"
                  x-component="Input"
                  x-validator={{
                    required: true,
                  }}
                  x-component-props={{
                    prefix: "{{icon('UserOutlined')}}",
                  }}
                />
                <SchemaField.String
                  name="password"
                  title="密码"
                  required
                  x-decorator="FormItem"
                  x-component="Input"
                  x-component-props={{
                    prefix: "{{icon('LockOutlined')}}",
                  }}
                />
              SchemaField>
              <Submit block size="large">
                登录
              Submit>
            Form>
          Tabs.TabPane>
          <Tabs.TabPane key="2" tab="手机登录">
            <Form
              form={phoneForm}
              layout="vertical"
              size="large"
              onAutoSubmit={console.log}
            >
              <SchemaField>
                <SchemaField.String
                  name="phone"
                  title="手机号"
                  required
                  x-validator="phone"
                  x-decorator="FormItem"
                  x-component="Input"
                  x-component-props={{
                    prefix: "{{icon('PhoneOutlined')}}",
                  }}
                />
                <SchemaField.String
                  name="verifyCode"
                  title="验证码"
                  required
                  x-decorator="FormItem"
                  x-component="VerifyCode"
                  x-component-props={{
                    prefix: "{{icon('LockOutlined')}}",
                  }}
                  x-reactions={[
                    {
                      dependencies: ['.phone#value', '.phone#valid'],
                      fulfill: {
                        state: {
                          'component[1].readyPost': '{{$deps[0] && $deps[1]}}',
                          'component[1].phoneNumber': '{{$deps[0]}}',
```

```
                    },
                  },
                },
              ]}
            />
            SchemaField>
            <Submit block size="large">
              登录
            Submit>
          Form>
        Tabs.TabPane>
      Tabs>
      <div
        style={{
          display: 'flex',
          justifyContent: 'space-between',
        }}
      >
        <a href="#新用户注册">新用户注册a>
        <a href="#忘记密码">忘记密码?a>
      div>
    Card>
  div>
 )
}
```

```
import React, { useState } from 'react'
import { Input, Button } from 'antd'

interface IVerifyCodeProps {
  value?: any
  onChange?: (value: any) => void
  readyPost?: boolean
  phoneNumber?: number
  style?: React.CSSProperties
}

export const VerifyCode: React.FC> =
  ({ value, onChange, readyPost, phoneNumber, ...props }) => {
    const [lastTime, setLastTime] = useState(0)

    const counting = (time = 20) => {
      if (time < 0) return
      setLastTime(time)
      setTimeout(() => {
        counting(time - 1)
      }, 1000)
    }

    return (
      <div
        style={{ display: 'inline-flex', width: '100%', alignItems: 'center' }}
      >
        <Input
          {...props}
          style={{ marginRight: 5, ...props.style }}
          value={value}
          onChange={onChange}
        />
        <div
          style={{
            flexShrink: 0,
            color: '#999',
            width: 100,
            height: 35,
            display: 'flex',
            alignItems: 'center',
            justifyContent: 'center',
          }}
        >
          {lastTime === 0 && (
            <Button
              disabled={!readyPost}
              block
              onClick={() => {
                if (phoneNumber) {
                  console.log(`post code by phone number ${phoneNumber}`)
                }
                counting()
              }}
            >
              发送验证码
            Button>
          )}
          {lastTime > 0 && <span>剩余{lastTime}秒span>}
        div>
      div>
    )
  }
```