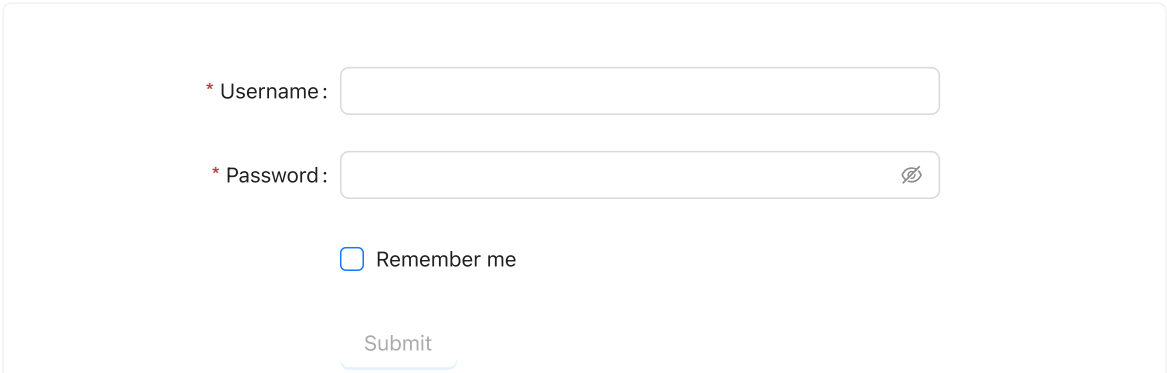
Form

High performance Form component with data scope management. Including data collection, verification, and styles.

When to use

- When you need to create an instance or collect information.
- When you need to validate fields in certain rules.

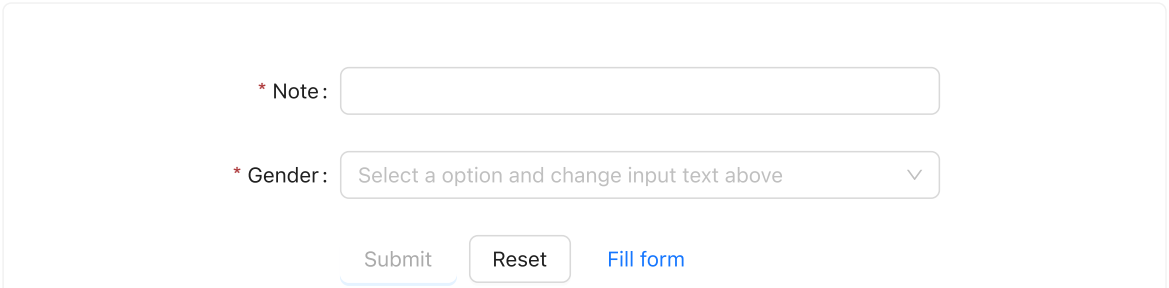
Examples



Basic Usage

Basic Form data control. Includes layout, initial values, validation and submit.

```
import React from 'react';
import { Button, Checkbox, Form, Input } from 'antd';

const onFinish = (values: any) => {
  console.log('Success:', values);
};

const onFinishFailed = (errorInfo: any) => {
  console.log('Failed:', errorInfo);
};

const App: React.FC = () => (
  <Form
    name="basic"
    labelCol={{ span: 8 }}
    wrapperCol={{ span: 16 }}
    style={{ maxWidth: 600 }}
    initialValues={{ remember: true }}
    onFinish={onFinish}
    onFinishFailed={onFinishFailed}
    autoComplete="off"
  >
    <Form.Item
      label="Username"
      name="username"
      rules={[{ required: true, message: 'Please input your username!' }]}
    >
      <Input />
    </Form.Item>

    <Form.Item
      label="Password"
      name="password"
      rules={[{ required: true, message: 'Please input your password!' }]}
    >
      <Input.Password />
    </Form.Item>

    <Form.Item name="remember" valuePropName="checked" wrapperCol={{ offset: 8, span: 16 }}>
      <Checkbox>Remember me</Checkbox>
    </Form.Item>

    <Form.Item wrapperCol={{ offset: 8, span: 16 }}>
      <Button type="primary" htmlType="submit">
        Submit
      </Button>
    </Form.Item>
  </Form>
);

export default App;
```

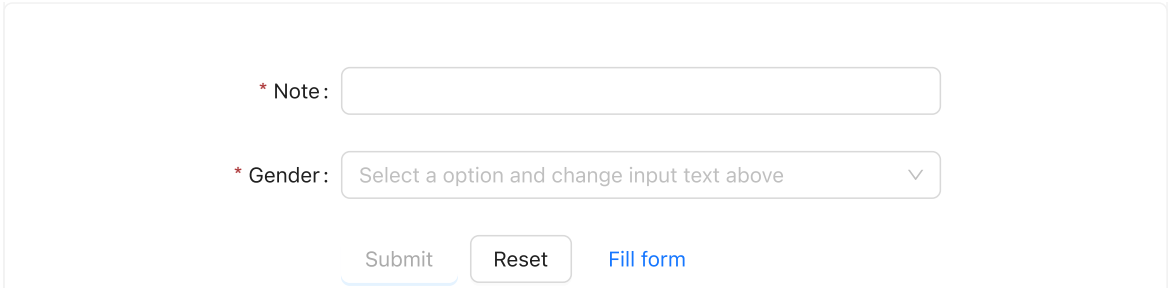


Form methods

Call form method with `Form.useForm`.

> Note that `useForm` is a React Hooks that only works in functional component.

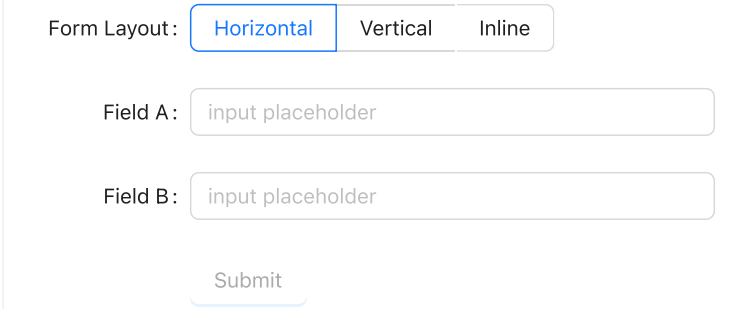
```
import { Button, Form, Input, Select } from 'antd';
import React from 'react';

const { Option } = Select;

const layout = {
  labelCol: { span: 8 },
  wrapperCol: { span: 16 },
};

const tailLayout = {
  wrapperCol: { offset: 8, span: 16 },
};

const App: React.FC = () => {
  const [form] = Form.useForm();

  const onGenderChange = (value: string) => {
    switch (value) {
      case 'male':
        form.setFieldsValue({ note: 'Hi, man!' });
        break;
      case 'female':
        form.setFieldsValue({ note: 'Hi, lady!' });
        break;
      case 'other':
        form.setFieldsValue({ note: 'Hi there!' });
        break;
      default:
    }
  };

  const onFinish = (values: any) => {
    console.log(values);
  };

  const onReset = () => {
    form.resetFields();
  };

  const onFill = () => {
    form.setFieldsValue({ note: 'Hello world!', gender: 'male' });
  };

  return (
    <Form
      {...layout}
      form={form}
      name="control-hooks"
      onFinish={onFinish}
      style={{ maxWidth: 600 }}
    >
      <Form.Item name="note" label="Note" rules={[{ required: true }]}>
        <Input />
      </Form.Item>
```



Form methods (Class component)

We recommend use `Form.useForm` to create data control. If you are using class component, you can get it by `ref`.

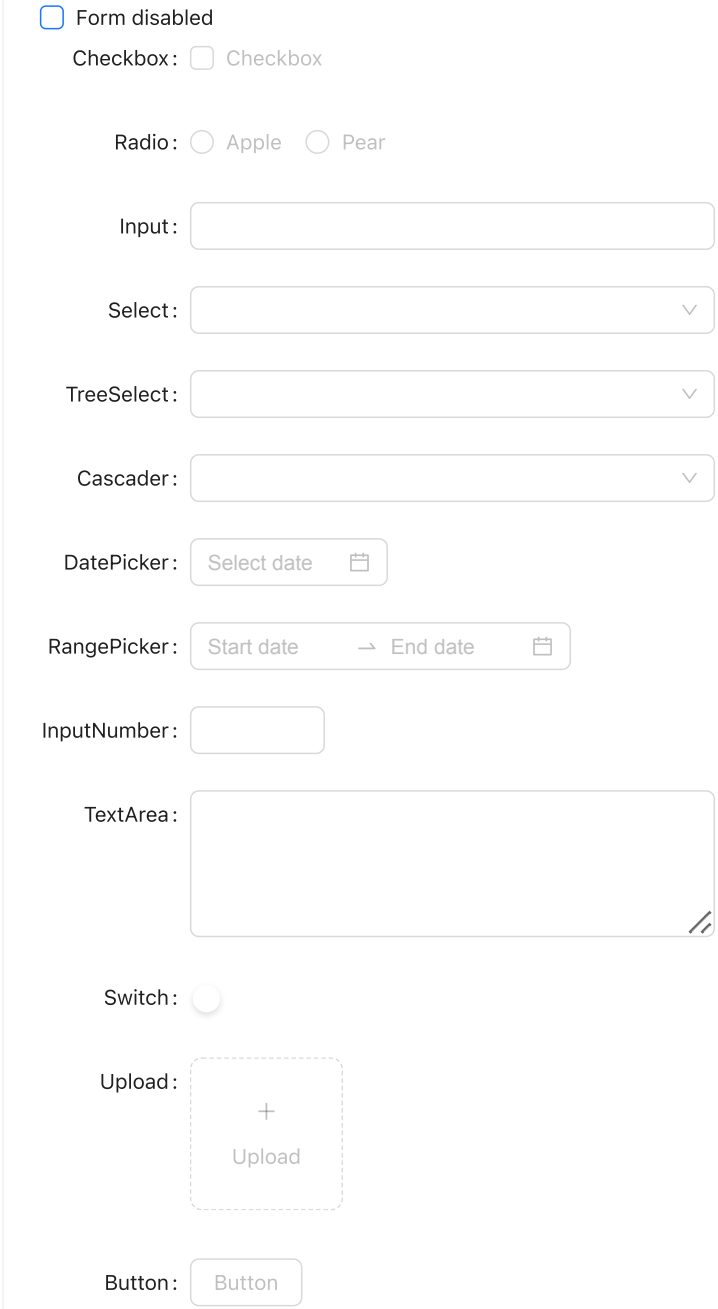
```
import React from 'react';
import { Button, Form, Input, Select } from 'antd';
import type { FormInstance } from 'antd/es/form';

const { Option } = Select;

const layout = {
  labelCol: { span: 8 },
  wrapperCol: { span: 16 },
};

const tailLayout = {
  wrapperCol: { offset: 8, span: 16 },
};

const App: React.FC = () => {
  const formRef = React.useRef<FormInstance>(null);

  const onGenderChange = (value: string) => {
    switch (value) {
      case 'male':
        formRef.current?.setFieldsValue({ note: 'Hi, man!' });
        break;
      case 'female':
        formRef.current?.setFieldsValue({ note: 'Hi, lady!' });
        break;
      case 'other':
        formRef.current?.setFieldsValue({ note: 'Hi there!' });
        break;
      default:
        break;
    }
  };

  const onFinish = (values: any) => {
    console.log(values);
  };

  const onReset = () => {
    formRef.current?.resetFields();
  };

  const onFill = () => {
    formRef.current?.setFieldsValue({ note: 'Hello world!', gender: 'male' });
  };

  return (
    <Form
      {...layout}
      ref={formRef}
      name="control-ref"
      onFinish={onFinish}
      style={{ maxWidth: 600 }}
    >
      <Form.Item name="note" label="Note" rules={[{ required: true }]}>
        <Input />
```



Form Layout

There are three layout for form: `horizontal`, `vertical`, `inline`.

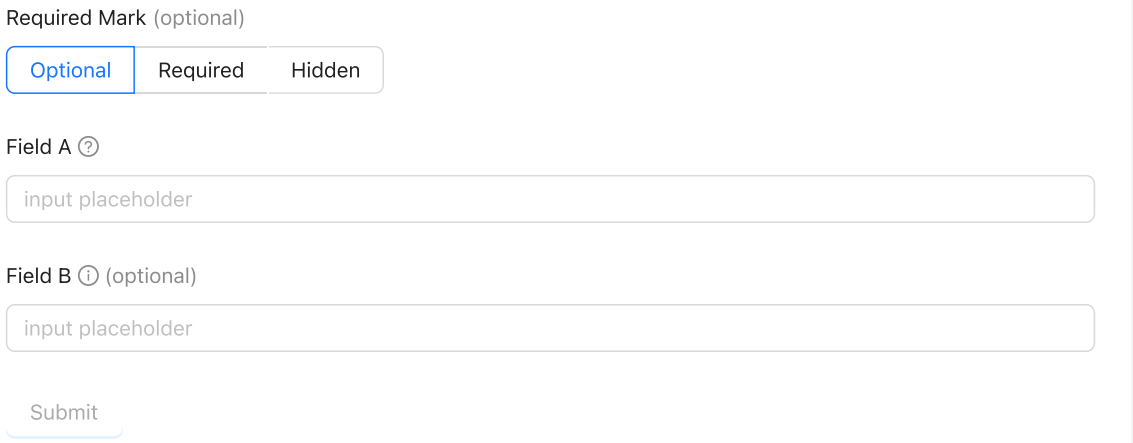
```
import React, { useState } from 'react';
import { Button, Form, Input, Radio } from 'antd';

type LayoutType = Parameters<typeof Form>[0]['layout'];

const App: React.FC = () => {
  const [form] = Form.useForm();
  const [formLayout, setFormLayout] = useState<LayoutType>('horizontal');

  const onFormLayoutChange = ({ layout }: { layout: LayoutType }) => {
    setFormLayout(layout);
  };

  const formItemLayout =
    formLayout === 'horizontal' ? { labelCol: { span: 4 }, wrapperCol: { span: 14 } } : null;

  const buttonItemLayout =
    formLayout === 'horizontal' ? { wrapperCol: { span: 14, offset: 4 } } : null;

  return (
    <Form
      {...formItemLayout}
      layout={formLayout}
      form={form}
      initialValues={{ layout: formLayout }}
      onValuesChange={onFormLayoutChange}
      style={{ maxWidth: 600 }}
    >
      <Form.Item label="Form Layout" name="layout">
        <Radio.Group value={formLayout}>
          <Radio.Button value="horizontal">Horizontal</Radio.Button>
          <Radio.Button value="vertical">Vertical</Radio.Button>
          <Radio.Button value="inline">Inline</Radio.Button>
        </Radio.Group>
      </Form.Item>
      <Form.Item label="Field A">
        <Input placeholder="input placeholder" />
      </Form.Item>
      <Form.Item label="Field B">
        <Input placeholder="input placeholder" />
      </Form.Item>
      <Form.Item {...buttonItemLayout}>
        <Button type="primary">Submit</Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```



Form disabled

Set component disabled, only works for antd components.

```
import { PlusOutlined } from '@ant-design/icons';
import {
  Button,
  Cascader,
  Checkbox,
  DatePicker,
  Form,
  Input,
  InputNumber,
  Radio,
  Select,
  Switch,
  TreeSelect,
  Upload,
} from 'antd';
import React, { useState } from 'react';

const { RangePicker } = DatePicker;
const { TextArea } = Input;
```

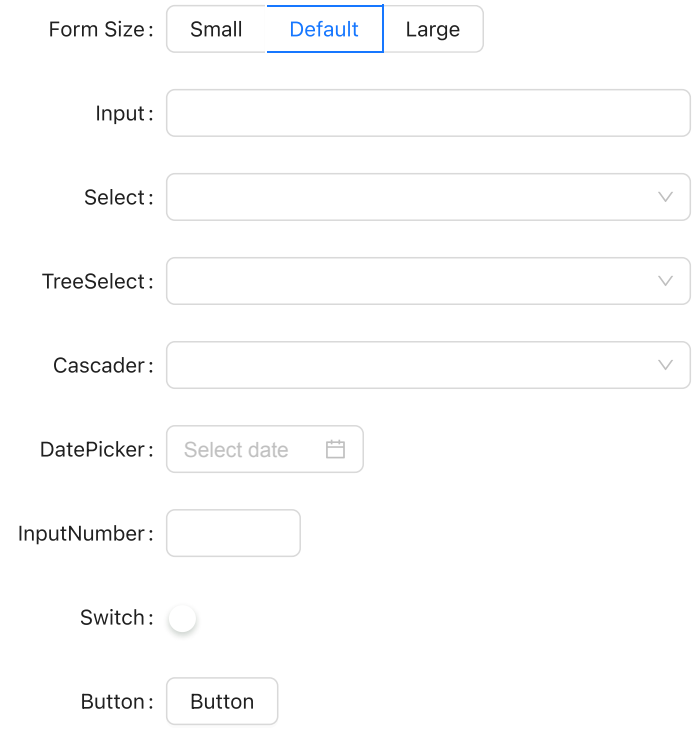


Required style

Switch required or optional style with `requiredMark`.

```
import React, { useState } from 'react';
import { InfoCircleOutlined } from '@ant-design/icons';
import { Button, Form, Input, Radio } from 'antd';

type RequiredMark = boolean | 'optional';

const App: React.FC = () => {
  const [form] = Form.useForm();
  const [requiredMark, setRequiredMarkType] = useState<RequiredMark>('optional');

  const onRequiredTypeChange = ({ requiredMarkValue }: { requiredMarkValue: RequiredMark }) => {
    setRequiredMarkType(requiredMarkValue);
  };

  return (
    <Form
      form={form}
      layout="vertical"
      initialValues={{ requiredMarkValue: requiredMark }}
      onValuesChange={onRequiredTypeChange}
      requiredMark={requiredMark}
    >
      <Form.Item label="Required Mark" name="requiredMarkValue">
        <Radio.Group>
          <Radio.Button value="optional">Optional</Radio.Button>
          <Radio.Button value>Required</Radio.Button>
          <Radio.Button value={false}>Hidden</Radio.Button>
        </Radio.Group>
      </Form.Item>
      <Form.Item label="Field A" required tooltip="This is a required field">
        <Input placeholder="input placeholder" />
      </Form.Item>
      <Form.Item
        label="Field B"
        tooltip={{ title: 'Tooltip with customize icon', icon: <InfoCircleOutlined /> }}
      >
        <Input placeholder="input placeholder" />
      </Form.Item>
      <Form.Item>
        <Button type="primary">Submit</Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

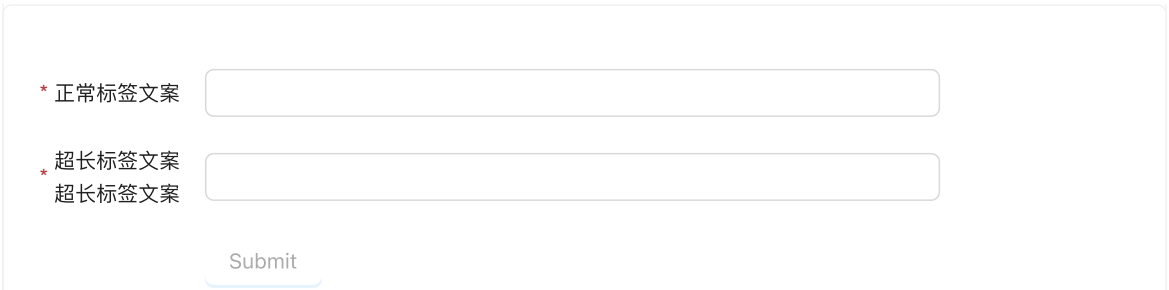
```
        <Form.Item label="InputNumber">
          <InputNumber />
        </Form.Item>
```



Form size

Set component size, only works for antd components.

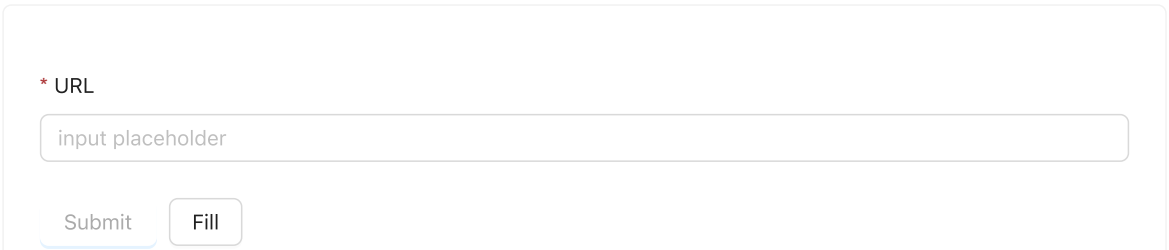
```
import React, { useState } from 'react';
import {
  Button,
  Cascader,
  DatePicker,
  Form,
  Input,
  InputNumber,
  Radio,
  Select,
  Switch,
  TreeSelect,
} from 'antd';

type SizeType = Parameters<typeof Form>[0]['size'];

const App: React.FC = () => {
  const [componentSize, setComponentSize] = useState<SizeType | 'default'>('default');

  const onFormLayoutChange = ({ size }: { size: SizeType }) => {
    setComponentSize(size);
  };

  return (
    <Form
      labelCol={{ span: 4 }}
      wrapperCol={{ span: 14 }}
      layout="horizontal"
      initialValues={{ size: componentSize }}
      onValuesChange={onFormLayoutChange}
      size={componentSize as SizeType}
      style={{ maxWidth: 600 }}
    >
      <Form.Item label="Form Size" name="size">
        <Radio.Group>
          <Radio.Button value="small">Small</Radio.Button>
          <Radio.Button value="default">Default</Radio.Button>
          <Radio.Button value="large">Large</Radio.Button>
        </Radio.Group>
      </Form.Item>
```



label can wrap

Turn on `labelWrap` to wrap label if text is long.

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

const App: React.FC = () => (
  <Form
    name="wrap"
    labelCol={{ flex: '110px' }}
    labelAlign="left"
    labelWrap
    wrapperCol={{ flex: 1 }}
    colon={false}
    style={{ maxWidth: 600 }}
  >
    <Form.Item label="正常标签文案" name="username" rules={[{ required: true }]}>
      <Input />
    </Form.Item>

    <Form.Item label="超长标签文案超长标签文案" name="password" rules={[{ required: true }]}>
      <Input />
    </Form.Item>

    <Form.Item label=" ">
      <Button type="primary" htmlType="submit">
        Submit
      </Button>
    </Form.Item>
  </Form>
);

export default App;
```



No block rule

`rule` with `warningOnly` will not block form submit.

```
import React from 'react';
import { Button, Form, Input, message, Space } from 'antd';

const App: React.FC = () => {
  const [form] = Form.useForm();

  const onFinish = () => {
    message.success('Submit success!');
  };

  const onFinishFailed = () => {
    message.error('Submit failed!');
  };

  const onFill = () => {
    form.setFieldsValue({
      url: 'https://taobao.com/',
    });
  };

  return (
    <Form
      form={form}
      layout="vertical"
      onFinish={onFinish}
      onFinishFailed={onFinishFailed}
      autoComplete="off"
    >
      <Form.Item
        name="url"
        label="URL"
        rules={[{ required: true }, { type: 'url', warningOnly: true }, { type: 'string', min: 6
      >
        <Input placeholder="input placeholder" />
      </Form.Item>
      <Form.Item>
        <Space>
          <Button type="primary" htmlType="submit">
            Submit
          </Button>
          <Button htmlType="button" onClick={onFill}>
            Fill
          </Button>
        </Space>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Name (Watch to trigger rerender)

Age (Not Watch)

```
Name Value:
```

Watch Hooks

`useWatch` helps watch the field change and only re-render for the value change. API Ref.

```
import React from 'react';
import { Form, Input, InputNumber, Typography } from 'antd';

const Demo: React.FC = () => {
  const [form] = Form.useForm<{ name: string; age: number }>();
  const nameValue = Form.useWatch('name', form);

  return (
    <>
      <Form form={form} layout="vertical" autoComplete="off">
        <Form.Item name="name" label="Name (Watch to trigger rerender)">
          <Input />
        </Form.Item>
        <Form.Item name="age" label="Age (Not Watch)">
          <InputNumber />
        </Form.Item>
      </Form>

      <Typography>
        <pre>Name Value: {nameValue}</pre>
      </Typography>
    </>
  );
};

export default Demo;
```

First Name

Last Name

Age

Submit

Path Prefix

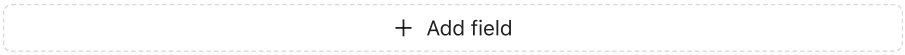
In some scenarios, you may want to set a prefix for some fields consistently. You can achieve this effect with HOC.

```
import React from 'react';
import { Form, Input, Button } from 'antd';
import type { FormItemProps } from 'antd';

const MyFormItemContext = React.createContext<(string | number)[]>([]);

interface MyFormItemGroupProps {
  prefix: string | number | (string | number)[];
  children: React.ReactNode;
}

function toArr(str: string | number | (string | number)[]): (string | number)[] {
  return Array.isArray(str) ? str : [str];
}

const MyFormItemGroup = ({ prefix, children }: MyFormItemGroupProps) => {
  const prefixPath = React.useContext(MyFormItemContext);
  const concatPath = React.useMemo(() => [...prefixPath, ...toArr(prefix)], [prefixPath, prefix]

  return <MyFormItemContext.Provider value={concatPath}>{children}</MyFormItemContext.Provider>;
};

const MyFormItem = ({ name, ...props }: FormItemProps) => {
  const prefixPath = React.useContext(MyFormItemContext);
  const concatName = name !== undefined ? [...prefixPath, ...toArr(name)] : undefined;

  return <Form.Item name={concatName} {...props} />;
};

const App: React.FC = () => {
  const onFinish = (value: object) => {
    console.log(value);
  };

  return (
    <Form name="form_item_path" layout="vertical" onFinish={onFinish}>
      <MyFormItemGroup prefix={['user']}>
        <MyFormItemGroup prefix={['name']}>
          <MyFormItem name="firstName" label="First Name">
            <Input />
          </MyFormItem>
          <MyFormItem name="lastName" label="Last Name">
            <Input />
          </MyFormItem>
        </MyFormItemGroup>

        <MyFormItem name="age" label="Age">
          <Input />
        </MyFormItem>
      </MyFormItemGroup>
```

Dynamic Form Item

Add or remove form items dynamically. `add` function support config initial value.

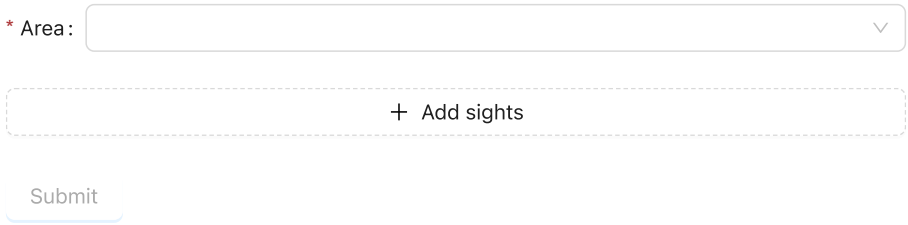
```
import React from 'react';
import { MinusCircleOutlined, PlusOutlined } from '@ant-design/icons';
import { Button, Form, Input } from 'antd';

const formItemLayout = {
  labelCol: {
    xs: { span: 24 },
    sm: { span: 4 },
  },
  wrapperCol: {
    xs: { span: 24 },
    sm: { span: 20 },
  },
};

const formItemLayoutWithOutLabel = {
  wrapperCol: {
    xs: { span: 24, offset: 0 },
    sm: { span: 20, offset: 4 },
  },
};

const App: React.FC = () => {
  const onFinish = (values: any) => {
    console.log('Received values of form:', values);
  };

  return (
    <Form
      name="dynamic_form_item"
      {...formItemLayoutWithOutLabel}
      onFinish={onFinish}
      style={{ maxWidth: 600 }}
    >
      <Form.List
        name="names"
        rules={[
          {
            validator: async (_, names) => {
              if (!names || names.length < 2) {
                return Promise.reject(new Error('At least 2 passengers'));
              }
            },
          },
        ]}
      >
        {(fields, { add, remove }, { errors }) => (
          <>
            {fields.map((field, index) => (
              <Form.Item
                {...(index === 0 ? formItemLayout : formItemLayoutWithOutLabel)}
                label={index === 0 ? 'Passengers' : ''}
                required={false}
                key={field.key}
              >
                <Form.Item
                  {...field}
```



Dynamic Form nest Items

Nest dynamic field need extends `field`. Pass `field.name` to nest item.

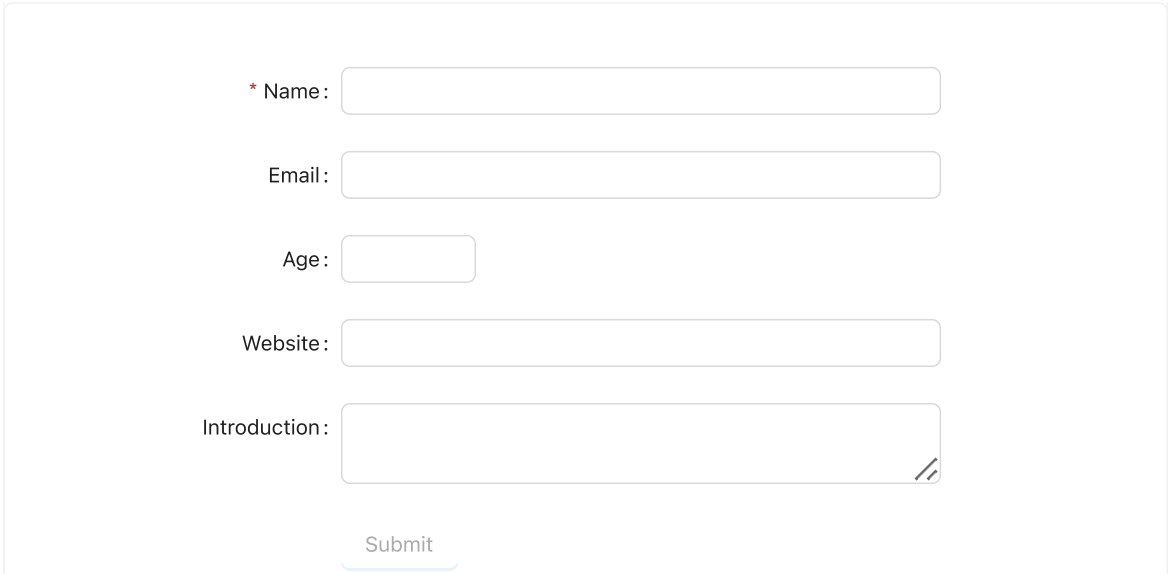
```
import React from 'react';
import { MinusCircleOutlined, PlusOutlined } from '@ant-design/icons';
import { Button, Form, Input, Space } from 'antd';

const onFinish = (values: any) => {
  console.log('Received values of form:', values);
};

const App: React.FC = () => (
  <Form
    name="dynamic_form_nest_item"
    onFinish={onFinish}
    style={{ maxWidth: 600 }}
    autoComplete="off"
  >
    <Form.List name="users">
      {(fields, { add, remove }) => (
        <>
          {fields.map(({ key, name, ...restField }) => (
            <Space key={key} style={{ display: 'flex', marginBottom: 8 }} align="baseline">
              <Form.Item
                {...restField}
                name={[name, 'first']}
                rules={[{ required: true, message: 'Missing first name' }]}
              >
                <Input placeholder="First Name" />
              </Form.Item>
              <Form.Item
                {...restField}
                name={[name, 'last']}
                rules={[{ required: true, message: 'Missing last name' }]}
              >
                <Input placeholder="Last Name" />
              </Form.Item>
              <MinusCircleOutlined onClick={() => remove(name)} />
            </Space>
          ))}
          <Form.Item>
            <Button type="dashed" onClick={() => add()} block icon={<PlusOutlined />}>
              Add field
            </Button>
          </Form.Item>
        </>
      )}
    </Form.List>
    <Form.Item>
      <Button type="primary" htmlType="submit">
        Submit
      </Button>
    </Form.Item>
  </Form>
);

export default App;
```



Complex Dynamic Form Item

This example demonstrates the case that a form contains multiple form controls.

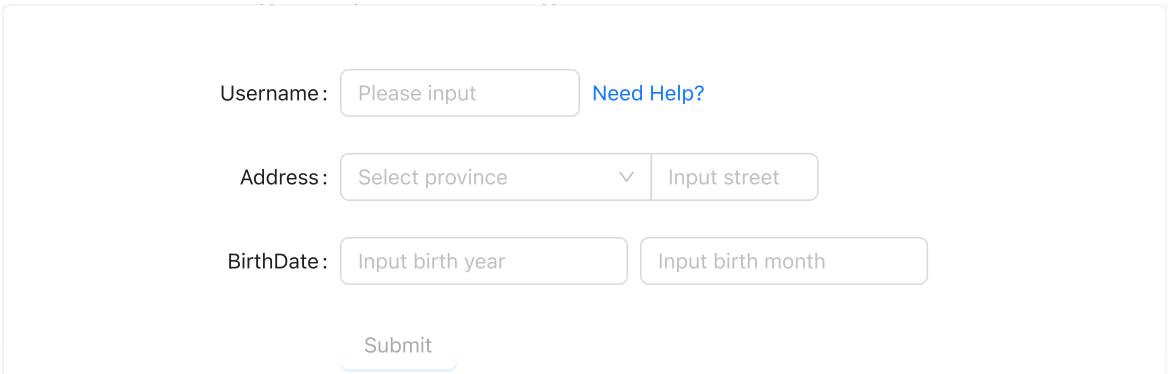
```
import React from 'react';
import { MinusCircleOutlined, PlusOutlined } from '@ant-design/icons';
import { Button, Form, Input, Select, Space } from 'antd';

const { Option } = Select;

const areas = [
  { label: 'Beijing', value: 'Beijing' },
  { label: 'Shanghai', value: 'Shanghai' },
];

const sights = {
  Beijing: ['Tiananmen', 'Great Wall'],
  Shanghai: ['Oriental Pearl', 'The Bund'],
};

type SightsKeys = keyof typeof sights;

const App: React.FC = () => {
  const [form] = Form.useForm();

  const onFinish = (values: any) => {
    console.log('Received values of form:', values);
  };

  const handleChange = () => {
    form.setFieldsValue({ sights: [] });
  };

  return (
    <Form
      form={form}
      name="dynamic_form_complex"
      onFinish={onFinish}
      style={{ maxWidth: 600 }}
      autoComplete="off"
    >
      <Form.Item name="area" label="Area" rules={[{ required: true, message: 'Missing area' }]}>
        <Select options={areas} onChange={handleChange} />
      </Form.Item>
      <Form.List name="sights">
        {(fields, { add, remove }) => (
          <>
            {fields.map((field) => (
              <Space key={field.key} align="baseline">
                <Form.Item
                  noStyle
                  shouldUpdate={(prevValues, curValues) =>
                    prevValues.area !== curValues.area || prevValues.sights !== curValues.sights
                  }
                >
                  {() => (
                    <Form.Item
                      {...field}
                      label="Sight"
                      name={[field.name, 'sight']}
                      rules={[{ required: true, message: 'Missing sight' }]}
```



Nest

`name` prop support nest data structure. Customize validate message template with `validateMessages` or `message`. Ref here about message template.

```
import React from 'react';
import { Button, Form, Input, InputNumber } from 'antd';

const layout = {
  labelCol: { span: 8 },
  wrapperCol: { span: 16 },
};

/* eslint-disable no-template-curly-in-string */
const validateMessages = {
  required: '${label} is required!',
  types: {
    email: '${label} is not a valid email!',
    number: '${label} is not a valid number!',
  },
  number: {
    range: '${label} must be between ${min} and ${max}',
  },
};
/* eslint-enable no-template-curly-in-string */

const onFinish = (values: any) => {
  console.log(values);
};

const App: React.FC = () => (
  <Form
    {...layout}
    name="nest-messages"
    onFinish={onFinish}
    style={{ maxWidth: 600 }}
    validateMessages={validateMessages}
  >
    <Form.Item name={['user', 'name']} label="Name" rules={[{ required: true }]}>
      <Input />
    </Form.Item>
    <Form.Item name={['user', 'email']} label="Email" rules={[{ type: 'email' }]}>
      <Input />
    </Form.Item>
    <Form.Item name={['user', 'age']} label="Age" rules={[{ type: 'number', min: 0, max: 99 }]}>
      <InputNumber />
    </Form.Item>
    <Form.Item name={['user', 'website']} label="Website">
      <Input />
    </Form.Item>
    <Form.Item name={['user', 'introduction']} label="Introduction">
```



complex form control

This demo shows how to use `Form.Item` with multiple controls. `<Form.Item name="field" />` will only bind the control(Input/Select) which is the only children of it. Imagine this case: you added some text description after the Input, then you have to wrap the Input by an extra `<Form.Item name="field">`. `style` property of `Form.Item` could be useful to modify the nested form item layout, or use `<Form.Item noStyle />` to turn it into a pure form-binded component(like `getFieldDecorator` in 3.x).

```
import React from 'react';
import { Button, Form, Input, Select, Space, Tooltip, Typography } from 'antd';

const { Option } = Select;

const onFinish = (values: any) => {
  console.log('Received values of form: ', values);
};

const App: React.FC = () => (
  <Form
    name="complex-form"
    onFinish={onFinish}
    labelCol={{ span: 8 }}
    wrapperCol={{ span: 16 }}
    style={{ maxWidth: 600 }}
  >
    <Form.Item label="Username">
      <Space>
        <Form.Item
          name="username"
          noStyle
          rules={[{ required: true, message: 'Username is required' }]}
        >
          <Input style={{ width: 160 }} placeholder="Please input" />
        </Form.Item>
        <Tooltip title="Useful information">
          <Typography.Link href="#API">Need Help?</Typography.Link>
        </Tooltip>
      </Space>
    </Form.Item>
    <Form.Item label="Address">
      <Space.Compact>
        <Form.Item
          name={['address', 'province']}
          noStyle
          rules={[{ required: true, message: 'Province is required' }]}
        >
          <Select placeholder="Select province">
            <Option value="Zhejiang">Zhejiang</Option>
            <Option value="Jiangsu">Jiangsu</Option>
          </Select>
        </Form.Item>
        <Form.Item
          name={['address', 'street']}
          noStyle
          rules={[{ required: true, message: 'Street is required' }]}
        >
```

Price: 0 RMB Submit

Customized Form Controls

Customized or third-party form controls can be used in Form, too. Controls must follow these conventions:

- It has a controlled property `value` or other name which is equal to the value of `valuePropName`.
- It has event `onChange` or an event which name is equal to the value of `trigger`.

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

const { Option } = Select;

type Currency = 'rmb' | 'dollar';

interface PriceValue {
  number?: number;
  currency?: Currency;
}

interface PriceInputProps {
  value?: PriceValue;
  onChange?: (value: PriceValue) => void;
}

const PriceInput: React.FC<PriceInputProps> = ({ value = {}, onChange }) => {
  const [number, setNumber] = useState(0);
  const [currency, setCurrency] = useState<Currency>('rmb');

  const triggerChange = (changedValue: { number?: number; currency?: Currency }) => {
    onChange?.({ number, currency, ...value, ...changedValue });
  };

  const onNumberChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const newNumber = parseInt(e.target.value || '0', 10);
    if (Number.isNaN(number)) {
      return;
    }
    if (!('number' in value)) {
      setNumber(newNumber);
    }
    triggerChange({ number: newNumber });
  };

  const onCurrencyChange = (newCurrency: Currency) => {
    if (!('currency' in value)) {
      setCurrency(newCurrency);
    }
    triggerChange({ currency: newCurrency });
  };

  return (
    <span>
      <Input
        type="text"
        value={value.number || number}
        onChange={onNumberChange}
        style={{ width: 100 }}
      />
      <Select
        value={value.currency || currency}
        style={{ width: 80, margin: '0 8px' }}
        onChange={onCurrencyChange}
      >
        <Option value="rmb">RMB</Option>
        <Option value="dollar">Dollar</Option>
      </Select>
    </span>
  );
```

* Username:

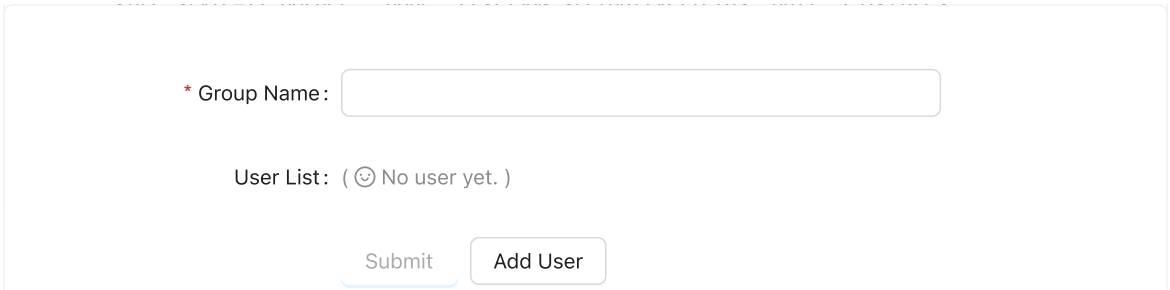
```
[
  {
    "name": [
      "username"
    ],
    "value": "Ant Design"
  }
]
```

Store Form Data into Upper Component

We can store form data into upper component or Redux or dva by using `onFieldsChange` and `fields`, see more at this rc-field-form demo.

Note: Save Form data globally is not a good practice. You should avoid this if not necessary.

```tsx
import React, { useState } from 'react';
import { Form, Input, Typography } from 'antd';

const { Paragraph } = Typography;

interface FieldData {
  name: string | number | (string | number)[];
  value?: any;
  touched?: boolean;
  validating?: boolean;
  errors?: string[];
}

interface CustomizedFormProps {
  onChange: (fields: FieldData[]) => void;
  fields: FieldData[];
}

const CustomizedForm: React.FC<CustomizedFormProps> = ({ onChange, fields }) => (
  <Form
    name="global_state"
    layout="inline"
    fields={fields}
    onFieldsChange={(_, allFields) => {
      onChange(allFields);
    }}
  >
    <Form.Item
      name="username"
      label="Username"
      rules={[{ required: true, message: 'Username is required!' }]}
    >
      <Input />
    </Form.Item>
  </Form>
);

const App: React.FC = () => {
  const [fields, setFields] = useState<FieldData[]>([{ name: ['username'], value: 'Ant Design' }

  return (
    <>
      <CustomizedForm
        fields={fields}
        onChange={(newFields) => {
          setFields(newFields);
        }}
      />
      <Paragraph style={{ maxWidth: 440, marginTop: 24 }}>
        <pre style={{ border: 'none' }}>{JSON.stringify(fields, null, 2)}</pre>
```



Control between forms

Use `Form.Provider` to process data between forms. In this case, submit button is in the Modal which is out of Form. You can use `form.submit` to submit form. Besides, we recommend native `<Button htmlType="submit" />` to submit a form.

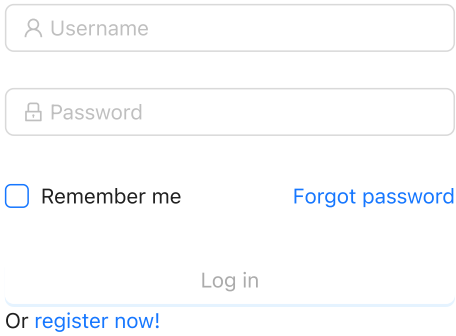
```
import React, { useEffect, useRef, useState } from 'react';
import { SmileOutlined, UserOutlined } from '@ant-design/icons';
import { Avatar, Button, Form, Input, InputNumber, Modal, Typography } from 'antd';
import type { FormInstance } from 'antd/es/form';

const layout = {
  labelCol: { span: 8 },
  wrapperCol: { span: 16 },
};

const tailLayout = {
  wrapperCol: { offset: 8, span: 16 },
};

interface UserType {
  name: string;
  age: string;
}

interface ModalFormProps {
  open: boolean;
  onCancel: () => void;
}

// reset form fields when modal is form, closed
const useResetFormOnCloseModal = ({ form, open }: { form: FormInstance; open: boolean }) => {
  const prevOpenRef = useRef<boolean>();
  useEffect(() => {
    prevOpenRef.current = open;
  }, [open]);
  const prevOpen = prevOpenRef.current;

  useEffect(() => {
    if (!open && prevOpen) {
      form.resetFields();
    }
  }, [form, prevOpen, open]);
};

const ModalForm: React.FC<ModalFormProps> = ({ open, onCancel }) => {
  const [form] = Form.useForm();

  useResetFormOnCloseModal({
    form,
    open,
  });

  const onOk = () => {
    form.submit();
  };

  return (
    <Modal title="Basic Drawer" open={open} onOk={onOk} onCancel={onCancel}>
      <Form form={form} layout="vertical" name="userForm">
```

Inline Login Form

Inline login form is often used in navigation bar.

```
import React, { useEffect, useState } from 'react';
import { LockOutlined, UserOutlined } from '@ant-design/icons';
import { Button, Form, Input } from 'antd';

const App: React.FC = () => {
  const [form] = Form.useForm();
  const [, forceUpdate] = useState({});

  // To disable submit button at the beginning.
  useEffect(() => {
    forceUpdate({});
  }, []);

  const onFinish = (values: any) => {
    console.log('Finish:', values);
  };

  return (
    <Form form={form} name="horizontal_login" layout="inline" onFinish={onFinish}>
      <Form.Item
        name="username"
        rules={[{ required: true, message: 'Please input your username!' }]}
      >
        <Input prefix={<UserOutlined className="site-form-item-icon" />} placeholder="Username" .
      </Form.Item>
      <Form.Item
        name="password"
        rules={[{ required: true, message: 'Please input your password!' }]}
      >
        <Input
          prefix={<LockOutlined className="site-form-item-icon" />}
          type="password"
          placeholder="Password"
        />
      </Form.Item>
      <Form.Item shouldUpdate>
        {() => (
          <Button
            type="primary"
            htmlType="submit"
            disabled={
              !form.isFieldsTouched(true) ||
              !!form.getFieldsError().filter(({ errors }) => errors.length).length
            }
          >
            Log in
          </Button>
        )}
      </Form.Item>
    </Form>
  );
};

export default App;
```

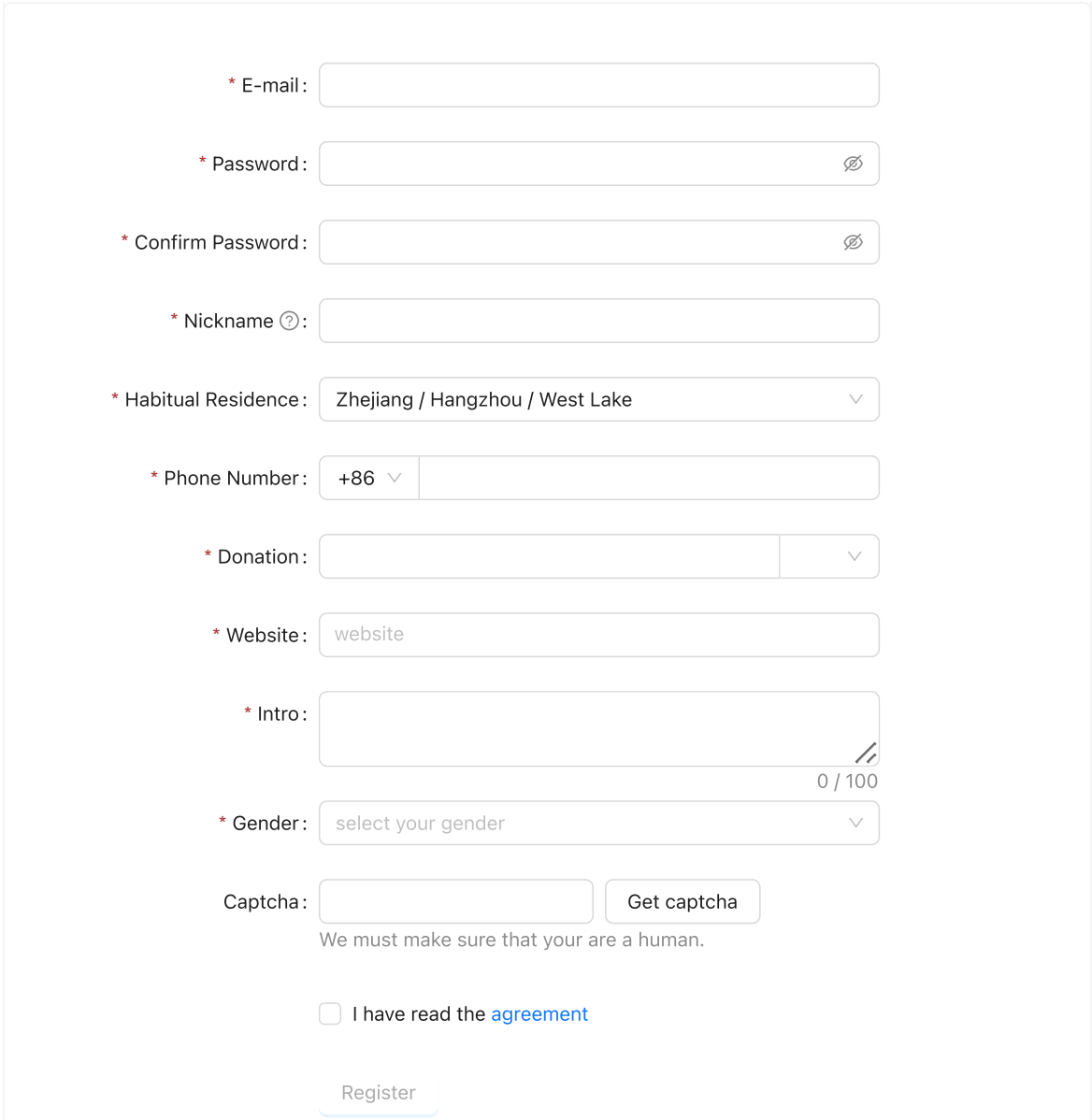
```
            Submit
          </Button>
          <Button htmlType="button" style={{ margin: '0 8px' }} onClick={showUserModal}>
            Add User
          </Button>
        </Form.Item>
      </Form>

      <ModalForm open={open} onCancel={hideUserModal} />
```



Login Form

Normal login form which can contain more elements.

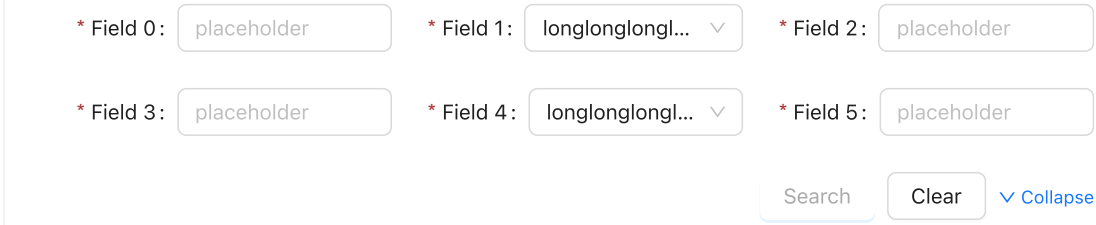
```
import React from 'react';
import { LockOutlined, UserOutlined } from '@ant-design/icons';
import { Button, Checkbox, Form, Input } from 'antd';

const App: React.FC = () => {
  const onFinish = (values: any) => {
    console.log('Received values of form: ', values);
  };

  return (
    <Form
      name="normal_login"
      className="login-form"
      initialValues={{ remember: true }}
      onFinish={onFinish}
    >
      <Form.Item
        name="username"
        rules={[{ required: true, message: 'Please input your Username!' }]}
      >
        <Input prefix={<UserOutlined className="site-form-item-icon" />} placeholder="Username" 
      </Form.Item>
      <Form.Item
        name="password"
        rules={[{ required: true, message: 'Please input your Password!' }]}
      >
        <Input
          prefix={<LockOutlined className="site-form-item-icon" />}
          type="password"
          placeholder="Password"
        />
      </Form.Item>
      <Form.Item>
        <Form.Item name="remember" valuePropName="checked" noStyle>
          <Checkbox>Remember me</Checkbox>
        </Form.Item>

        <a className="login-form-forgot" href="">
          Forgot password
        </a>
      </Form.Item>

      <Form.Item>
        <Button type="primary" htmlType="submit" className="login-form-button">
          Log in
        </Button>
        Or <a href="">register now!</a>
      </Form.Item>
    </Form>
  );
};

export default App;
```



Registration

Fill in this form to create a new account for you.

```
import type { CascaderProps } from 'antd';
import {
  AutoComplete,
  Button,
  Cascader,
  Checkbox,
  Col,
  Form,
  Input,
  InputNumber,
  Row,
  Select,
} from 'antd';
import React, { useState } from 'react';

const { Option } = Select;

interface DataNodeType {
  value: string;
  label: string;
  children?: DataNodeType[];
}

const residences: CascaderProps<DataNodeType>['options'] = [
  {
    value: 'zhejiang',
```



Search Result List

Advanced search

Three columns layout is often used for advanced searching of data table.

Because the width of label is not fixed, you may need to adjust it by customizing its style.

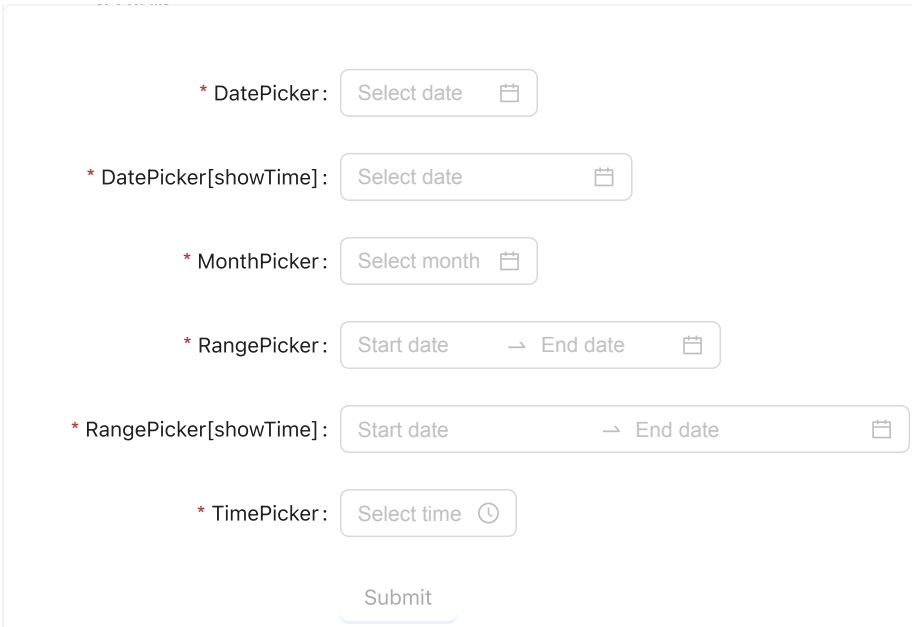
```
import React, { useState } from 'react';
import { DownOutlined } from '@ant-design/icons';
import { Button, Col, Form, Input, Row, Select, Space, theme } from 'antd';

const { Option } = Select;

const AdvancedSearchForm = () => {
  const { token } = theme.useToken();
  const [form] = Form.useForm();
  const [expand, setExpand] = useState(false);

  const formStyle = {
    maxWidth: 'none',
    background: token.colorFillAlter,
    borderRadius: token.borderRadiusLG,
    padding: 24,
  };

  const getFields = () => {
    const count = expand ? 10 : 6;
    const children = [];
    for (let i = 0; i < count; i++) {
      children.push(
        <Col span={8} key={i}>
          {i % 3 !== 1 ? (
            <Form.Item
              name={`field-${i}`}
              label={`Field ${i}`}
              rules={[
                {
                  required: true,
                  message: 'Input something!',
                },
              ]}
            >
              <Input placeholder="placeholder" />
            </Form.Item>
          ) : (
            <Form.Item
              name={`field-${i}`}
              label={`Field ${i}`}
              rules={[
                {
```

New Collection

Form in Modal to Create

When user visit a page with a list of items, and want to create a new item. The page can popup a form in Modal, then let user fill in the form to create an item.

```
import React, { useState } from 'react';
import { Button, Form, Input, Modal, Radio } from 'antd';

interface Values {
  title: string;
  description: string;
  modifier: string;
}

interface CollectionCreateFormProps {
  open: boolean;
  onCreate: (values: Values) => void;
  onCancel: () => void;
}

const CollectionCreateForm: React.FC<CollectionCreateFormProps> = ({
  open,
  onCreate,
  onCancel,
}) => {
  const [form] = Form.useForm();
  return (
    <Modal
      open={open}
      title="Create a new collection"
      okText="Create"
      cancelText="Cancel"
      onCancel={onCancel}
      onOk={() => {
        form
          .validateFields()
          .then((values) => {
            form.resetFields();
            onCreate(values);
          })
          .catch((info) => {
            console.log('Validate Failed:', info);
          });
      }}
    >
      <Form
        form={form}
        layout="vertical"
        name="form_in_modal"
        initialValues={{ modifier: 'public' }}
      >
        <Form.Item
          name="title"
          label="Title"
          rules={[{ required: true, message: 'Please input the title of collection!' }]}
        >
          <Input />
        </Form.Item>
        <Form.Item name="description" label="Description">
          <Input type="textarea" />
        </Form.Item>
        <Form.Item name="modifier" className="collection-create-form_last-form-item">
          <Radio.Group>
            <Radio value="public">Public</Radio>
            <Radio value="private">Private</Radio>
          </Radio.Group>
        </Form.Item>
      </Form>
```



Time-related Controls

The `value` of time-related components is a `dayjs` object, which we need to pre-process it before we submit to server.

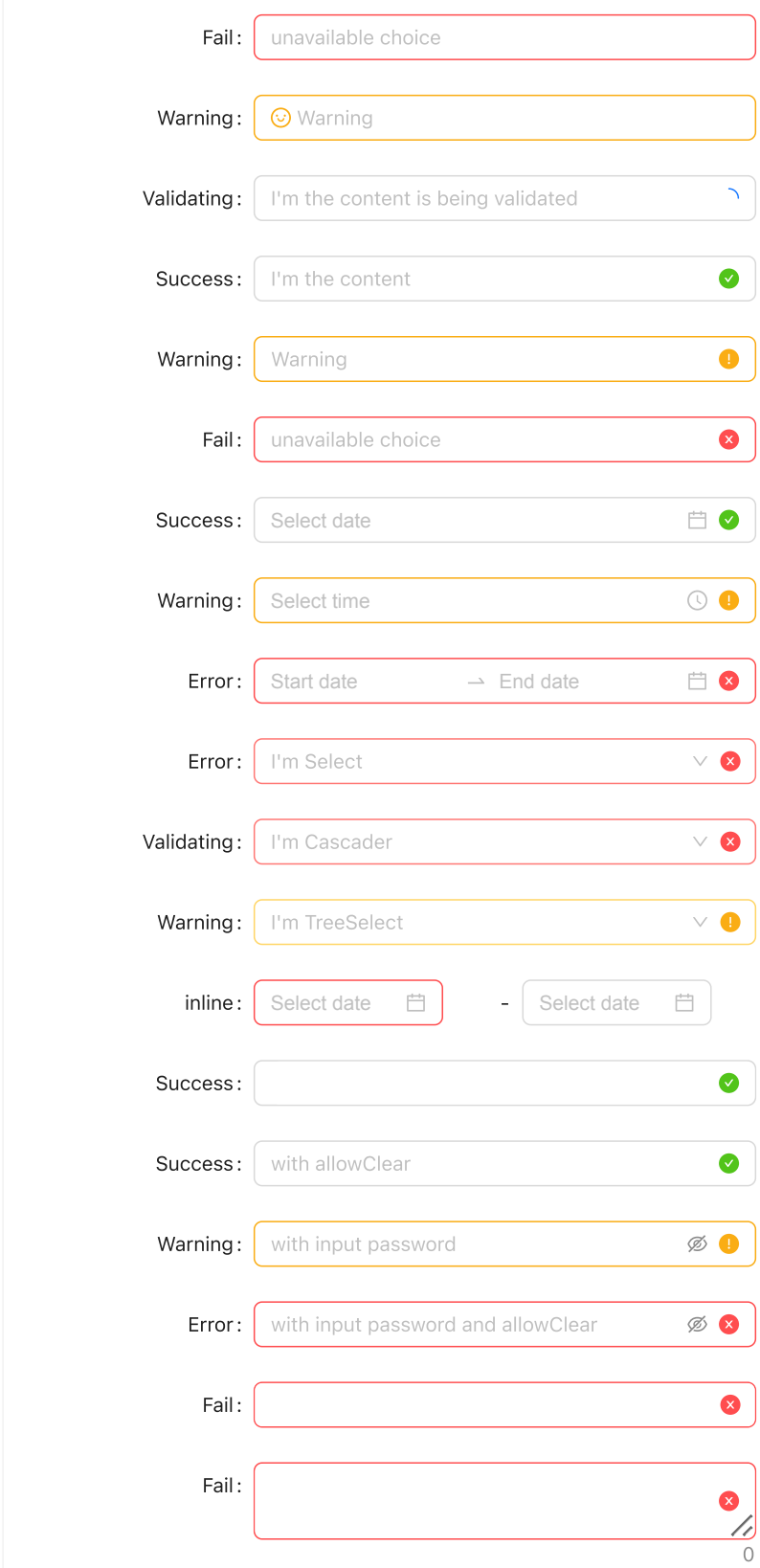
```
import React from 'react';
import { Button, DatePicker, Form, TimePicker } from 'antd';

const { RangePicker } = DatePicker;

const formItemLayout = {
  labelCol: {
    xs: { span: 24 },
    sm: { span: 8 },
  },
  wrapperCol: {
    xs: { span: 24 },
    sm: { span: 16 },
  },
};

const config = {
  rules: [{ type: 'object' as const, required: true, message: 'Please select time!' }],
};

const rangeConfig = {
  rules: [{ type: 'array' as const, required: true, message: 'Please select time!' }],
};

const onFinish = (fieldsValue: any) => {
  // Should format date value before submit.
  const rangeValue = fieldsValue['range-picker'];
  const rangeTimeValue = fieldsValue['range-time-picker'];
  const values = {
    ...fieldsValue,
    'date-picker': fieldsValue['date-picker'].format('YYYY-MM-DD'),
    'date-time-picker': fieldsValue['date-time-picker'].format('YYYY-MM-DD HH:mm:ss'),
    'month-picker': fieldsValue['month-picker'].format('YYYY-MM'),
    'range-picker': [rangeValue[0].format('YYYY-MM-DD'), rangeValue[1].format('YYYY-MM-DD')],
    'range-time-picker': [
      rangeTimeValue[0].format('YYYY-MM-DD HH:mm:ss'),
      rangeTimeValue[1].format('YYYY-MM-DD HH:mm:ss'),
    ],
    'time-picker': fieldsValue['time-picker'].format('HH:mm:ss'),
  };
  console.log('Received values of form: ', values);
};

const App: React.FC = () => (
```

Prime between 8 & 12: 11

Handle Form Data Manually

`Form` will collect and validate form data automatically. But if you don't need this feature or the default behavior cannot satisfy your business, you can handle form data manually.

```
import React, { useState } from 'react';
import { Form, InputNumber } from 'antd';

type ValidateStatus = Parameters<typeof Form.Item>[0]['validateStatus'];

const validatePrimeNumber = (
  number: number,
): {
  validateStatus: ValidateStatus;
  errorMsg: string | null;
} => {
  if (number === 11) {
    return {
      validateStatus: 'success',
      errorMsg: null,
    };
  }
  return {
    validateStatus: 'error',
    errorMsg: 'The prime between 8 and 12 is 11!',
  };
};

const formItemLayout = {
  labelCol: { span: 7 },
  wrapperCol: { span: 12 },
};

const tips =
  'A prime is a natural number greater than 1 that has no positive divisors other than 1 and its

const App: React.FC = () => {
  const [number, setNumber] = useState<{
    value: number;
    validateStatus?: ValidateStatus;
    errorMsg?: string | null;
  }>({ value: 11 });

  const onNumberChange = (value: number) => {
    setNumber({
      ...validatePrimeNumber(value),
      value,
    });
  };

  return (
    <Form style={{ maxWidth: 600 }}>
      <Form.Item
        {...formItemLayout}
        label="Prime between 8 & 12"
        validateStatus={number.validateStatus}
        help={number.errorMsg || tips}
      >
        <InputNumber min={8} max={12} value={number.value} onChange={onNumberChange} />
      </Form.Item>
    </Form>
  );
};

export default App;
```

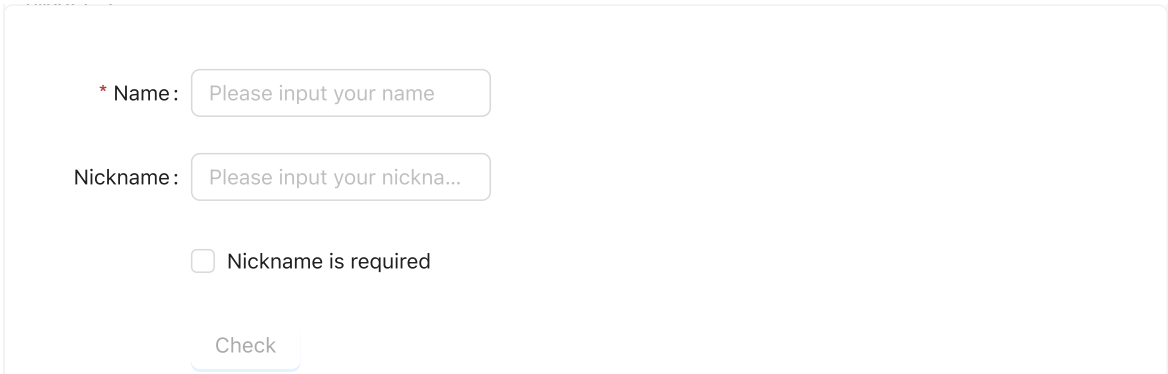


Customized Validation

We provide properties like `validateStatus` `help` `hasFeedback` to customize your own validate status and message, without using Form.

1. `validateStatus` : validate status of form components which could be 'success', 'warning', 'error', 'validating'.
2. `hasFeedback` : display feed icon of input control
3. `help` : display validate message.

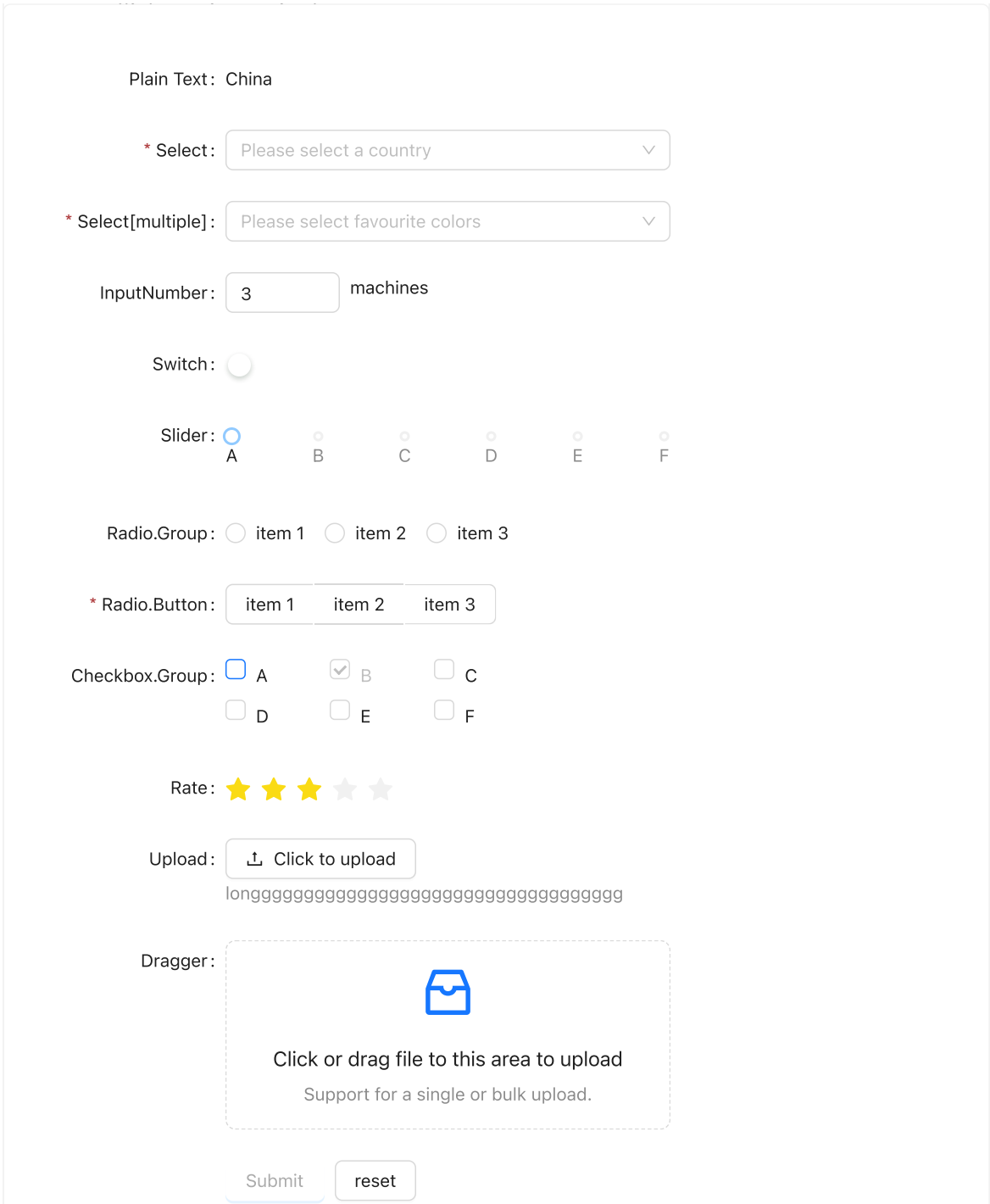
```
import React from 'react';
import { SmileOutlined } from '@ant-design/icons';
import {
```



Dynamic Rules

Perform different check rules according to different situations.

```
import React, { useEffect, useState } from 'react';
import { Button, Checkbox, Form, Input } from 'antd';

const formItemLayout = {
  labelCol: { span: 4 },
  wrapperCol: { span: 8 },
};

const formTailLayout = {
  labelCol: { span: 4 },
  wrapperCol: { span: 8, offset: 4 },
};

const App: React.FC = () => {
  const [form] = Form.useForm();
  const [checkNick, setCheckNick] = useState(false);

  useEffect(() => {
    form.validateFields(['nickname']);
  }, [checkNick, form]);

  const onCheckboxChange = (e: { target: { checked: boolean } }) => {
    setCheckNick(e.target.checked);
  };

  const onCheck = async () => {
    try {
      const values = await form.validateFields();
      console.log('Success:', values);
    } catch (errorInfo) {
      console.log('Failed:', errorInfo);
    }
  };

  return (
    <Form form={form} name="dynamic_rule" style={{ maxWidth: 600 }}>
      <Form.Item
        {...formItemLayout}
        name="username"
        label="Name"
        rules={[{ required: true, message: 'Please input your name' }]}
      >
        <Input placeholder="Please input your name" />
      </Form.Item>
      <Form.Item
        {...formItemLayout}
        name="nickname"
        label="Nickname"
        rules={[{ required: checkNick, message: 'Please input your nickname' }]}
      >
        <Input placeholder="Please input your nickname" />
      </Form.Item>
      <Form.Item {...formTailLayout}>
        <Checkbox checked={checkNick} onChange={onCheckboxChange}>
```



Other Form Controls

Demonstration of validation configuration for form controls which are not shown in the demos above.

```
import { InboxOutlined, UploadOutlined } from '@ant-design/icons';
import {
  Button,
  Checkbox,
  Col,
  Form,
  InputNumber,
  Radio,
  Rate,
  Row,
  Select,
  Slider,
  Space,
  Switch,
  Upload,
} from 'antd';
import React from 'react';

const { Option } = Select;
```

```
const { Option } = Select;

const formItemLayout = {
  labelCol: { span: 6 },
  wrapperCol: { span: 14 },
};

const normFile = (e: any) => {
  console.log('Upload event:', e);
  if (Array.isArray(e)) {
    return e;
  }
  return e?.fileList;
};

const onFinish = (values: any) => {
  console.log('Received values of form: ', values);
};

const App: React.FC = () => (
  <Form
    name="validate_other"
    {...formItemLayout}
    onFinish={onFinish}
    initialValues={{ 'input-number': 3, 'checkbox-group': ['A', 'B'], rate: 3.5 }}
    style={{ maxWidth: 600 }}
  >
    <Form.Item label="Plain Text">
      <span className="ant-form-text">China</span>
    </Form.Item>
    <Form.Item
      name="select"
      label="Select"
      hasFeedback
      rules={[{ required: true, message: 'Please select your country!' }]}
    >
      <Select placeholder="Please select a country">
        <Option value="china">China</Option>
        <Option value="usa">U.S.A</Option>
      </Select>
    </Form.Item>

    <Form.Item
      name="select-multiple"
      label="Select[multiple]"
      rules={[{ required: true, message: 'Please select your favourite colors!', type: 'array' }
    >
      <Select mode="multiple" placeholder="Please select favourite colors">
        <Option value="red">Red</Option>
        <Option value="green">Green</Option>
        <Option value="blue">Blue</Option>
      </Select>
    </Form.Item>

    <Form.Item label="InputNumber">
      <Form.Item name="input-number" noStyle>
        <InputNumber min={1} max={10} />
      </Form.Item>
      <span className="ant-form-text" style={{ marginLeft: 8 }}>
        machines
      </span>
    </Form.Item>

    <Form.Item name="switch" label="Switch" valuePropName="checked">
      <Switch />
    </Form.Item>

    <Form.Item name="slider" label="Slider">
      <Slider
        marks={{
          0: 'A',
          20: 'B',
          40: 'C',
          60: 'D',
          80: 'E'
```

```
      80: 'E',
      100: 'F',
    }}
  />
</Form.Item>

<Form.Item name="radio-group" label="Radio.Group">
  <Radio.Group>
    <Radio value="a">item 1</Radio>
    <Radio value="b">item 2</Radio>
    <Radio value="c">item 3</Radio>
  </Radio.Group>
</Form.Item>

<Form.Item
  name="radio-button"
  label="Radio.Button"
  rules={[{ required: true, message: 'Please pick an item!' }]}
>
  <Radio.Group>
    <Radio.Button value="a">item 1</Radio.Button>
    <Radio.Button value="b">item 2</Radio.Button>
    <Radio.Button value="c">item 3</Radio.Button>
  </Radio.Group>
</Form.Item>

<Form.Item name="checkbox-group" label="Checkbox.Group">
  <Checkbox.Group>
    <Row>
      <Col span={8}>
        <Checkbox value="A" style={{ lineHeight: '32px' }}>
          A
        </Checkbox>
      </Col>
      <Col span={8}>
        <Checkbox value="B" style={{ lineHeight: '32px' }} disabled>
          B
        </Checkbox>
      </Col>
      <Col span={8}>
        <Checkbox value="C" style={{ lineHeight: '32px' }}>
          C
        </Checkbox>
      </Col>
      <Col span={8}>
        <Checkbox value="D" style={{ lineHeight: '32px' }}>
          D
        </Checkbox>
      </Col>
      <Col span={8}>
        <Checkbox value="E" style={{ lineHeight: '32px' }}>
          E
        </Checkbox>
      </Col>
      <Col span={8}>
        <Checkbox value="F" style={{ lineHeight: '32px' }}>
          F
        </Checkbox>
      </Col>
    </Row>
  </Checkbox.Group>
</Form.Item>

<Form.Item name="rate" label="Rate">
  <Rate />
</Form.Item>

<Form.Item
  name="upload"
  label="Upload"
  valuePropName="fileList"
  getValueFromEvent={normFile}
  extra="longgggggggggggggggggggggggggggggggggg"
>
  <Upload name="logo" action="/upload.do" listType="picture">
```

```
        <Button icon={<UploadOutlined />}>Click to upload</Button>
      </Upload>
    </Form.Item>

    <Form.Item label="Dragger">
      <Form.Item name="dragger" valuePropName="fileList" getValueFromEvent={normFile} noStyle>
        <Upload.Dragger name="files" action="/upload.do">
          <p className="ant-upload-drag-icon">
            <InboxOutlined />
          </p>
          <p className="ant-upload-text">Click or drag file to this area to upload</p>
          <p className="ant-upload-hint">Support for a single or bulk upload.</p>
        </Upload.Dragger>
      </Form.Item>
    </Form.Item>

    <Form.Item wrapperCol={{ span: 12, offset: 6 }}>
      <Space>
        <Button type="primary" htmlType="submit">
          Submit
        </Button>
        <Button htmlType="reset">reset</Button>
      </Space>
    </Form.Item>
  </Form>
);

export default App;
```

API

Form

Property	Description	Type	Default
colon	Configure the default value of `colon` for Form.Item. Indicates whether the colon after the label is displayed (only effective when prop layout is horizontal)	boolean	true
disabled	Set form component disable, only available for antd components	boolean	false
component	Set the Form rendering element.	ComponentType | false	form

Property	Description	Type	Default
	Do not create a DOM node for `false`		
fields	Control of form fields through state management (such as redux). Not recommended for non-strong demand. View example	FieldData[]	-
form	Form control instance created by `Form.useForm()`. Automatically created when not provided	FormInstance	-
initialValues	Set value by Form initialization or reset	object	-
labelAlign	The text align of label of all items	`left` | `right`	`right`
labelWrap	whether label can be wrap	boolean	false
labelCol	Label layout, like `<Col>` component. Set `span` `offset` value like `{span: 3, offset: 12}` or `sm: {span: 3, offset: 12}`	object	-
layout	Form layout	`horizontal` | `vertical` | `inline`	`horizontal`
name	Form name. Will be the prefix of Field `id`	string	-
preserve	Keep field value even when field removed	boolean	true
requiredMark	Required mark style. Can use required mark or optional mark. You can not config to single Form.Item since this is a Form level config	boolean | `optional`	true
scrollToFirstError	Auto scroll to first failed field when submit	boolean | Options	false
size	Set field component size (antd components only)	`small` | `middle` | `large`	-
validateMessages	Validation prompt template,	ValidateMessages	-

Property	Description	Type	Default
	description see below		
validateTrigger	Config field validate trigger	string | string[]	onChange
wrapperCol	The layout for input controls, same as labelCol	object	-
onFieldsChange	Trigger when field updated	function(changedFields, allFields)	-
onFinish	Trigger after submitting the form and verifying data successfully	function(values)	-
onFinishFailed	Trigger after submitting the form and verifying data failed	function({ values, errorFields, outOfDate })	-
onValuesChange	Trigger when value updated	function(changedValues, allValues)	-

validateMessages

Form provides default verification error messages. You can modify the template by configuring `validateMessages` property. A common usage is to configure localization:

```
const validateMessages = {
  required: "'${name}' is required!",
  // ...
};

<Form validateMessages={validateMessages} />;
```

Besides, ConfigProvider also provides a global configuration scheme that allows for uniform configuration error notification templates:

```
const validateMessages = {
  required: "'${name}' is Required!",
  // ...
};

<ConfigProvider form={{ validateMessages }}>
  <Form />
</ConfigProvider>;
```

Form.Item

Form field component for data bidirectional binding, validation, layout, and so on.

Property	Description	Type	Default	Version
colon	Used with `label`, whether to display `:` after label text.	boolean	true	
dependencies	Set the dependency field. See below	NamePath[]	-	
extra	The extra prompt message. It is similar to help. Usage example: to display error message and prompt message at the same time	ReactNode	-	
getValueFromEvent	Specify how to get value from event or other onChange arguments	(..args: any[]) => any	-	
getValueProps	Additional props with sub component	(value: any) => any	-	4.2.0
hasFeedback	Used with `validateStatus`, this option specifies the validation status icon. Recommended to be used only with `Input`	boolean	false	
help	The prompt message. If not provided, the prompt message will be generated by the validation rule.	ReactNode	-	
hidden	Whether to hide Form.Item (still collect and validate value)	boolean	false	4.4.0
htmlFor	Set sub label `htmlFor`	string	-	
initialValue	Config sub default value. Form `initialValues` get higher priority when conflict	string	-	4.2.0
label	Label text	ReactNode	-	
labelAlign	The text align of label	`left` | `right`	`right`	
labelCol	The layout of label. You can set `span` `offset` to something like `{span: 3, offset:`	object	-	

Property	Description	Type	Default	Version
	`12}` or `sm: {span: 3, offset: 12}` same as with `<Col>`. You can set `labelCol` on Form which will not affect nest Item. If both exists, use Item first			
messageVariables	The default validate field info	Record<string, string>	-	4.7.0
name	Field name, support array	NamePath	-	
normalize	Normalize value from component value before passing to Form instance. Do not support async	(value, prevValue, prevValues) => any	-	
noStyle	No style for `true`, used as a pure field control	boolean	false	
preserve	Keep field value even when field removed	boolean	true	4.4.0
required	Display required style. It will be generated by the validation rule	boolean	false	
rules	Rules for field validation. Click here to see an example	Rule[]	-	
shouldUpdate	Custom field update logic. See below	boolean | (prevValue, curValue) => boolean	false	
tooltip	Config tooltip info	ReactNode | TooltipProps & { icon: ReactNode }	-	4.7.0
trigger	When to collect the value of children node. Click here to see an example	string	`onChange`	
validateFirst	Whether stop validate on first rule of error for this field. Will parallel validate when `parallel` configured	boolean | `parallel`	false	`parallel`: 4.5.0
validateStatus	The validation status. If not	string	-	

Property	Description	Type	Default	Version
	provided, it will be generated by validation rule. options: `success` `warning` `error` `validating`			
validateTrigger	When to validate the value of children node	string | string[]	`onChange`	
valuePropName	Props of children node, for example, the prop of Switch is 'checked'. This prop is an encapsulation of `getValueProps`, which will be invalid after customizing `getValueProps`	string	`value`	
wrapperCol	The layout for input controls, same as `labelCol`. You can set `wrapperCol` on Form which will not affect nest Item. If both exists, use Item first	object	-	

After wrapped by `Form.Item` with `name` property, `value` (or other property defined by `valuePropName`) `onChange` (or other property defined by `trigger`) props will be added to form controls, the flow of form data will be handled by Form which will cause:

1. You shouldn't use `onChange` on each form control to **collect data**(use `onValuesChange` of Form), but you can still listen to `onChange`.
2. You cannot set value for each form control via `value` or `defaultValue` prop, you should set default value with `initialValues` of Form. Note that `initialValues` cannot be updated by `setState` dynamically, you should use `setFieldsValue` in that situation.
3. You shouldn't call `setState` manually, please use `form.setFieldsValue` to change value programmatically.

dependencies

Used when there are dependencies between fields. If a field has the `dependencies` prop, this field will automatically trigger updates and validations when upstream is updated. A common scenario is a user registration form with "password" and "confirm password" fields. The "Confirm Password" validation depends on the "Password" field. After setting `dependencies`, the "Password" field update will re-trigger the validation of "Check Password". You can refer examples.

`dependencies` shouldn't be used together with `shouldUpdate`, since it may result in conflicting update logic.

`dependencies` supports `Form.Item` with render props children since `4.5.0`.

shouldUpdate

Form updates only the modified field-related components for performance optimization purposes by incremental update. In most cases, you only need to write code or do validation with the `dependencies` property. In some specific cases, such as when a new field option appears with a field value changed, or you just want to keep some area updating by form update, you can modify the update logic of Form.Item via the `shouldUpdate`.

When `shouldUpdate` is `true`, any Form update will cause the Form.Item to be re-rendered. This is very helpful for custom rendering some areas:

```
<Form.Item shouldUpdate>
  {() => {
    return <pre>{JSON.stringify(form.getFieldsValue(), null, 2)}</pre>;
  }}
</Form.Item>
```

You can ref example to see detail.

When `shouldUpdate` is a function, it will be called by form values update. Providing original values and current value to compare. This is very helpful for rendering additional fields based on values:

```
<Form.Item shouldUpdate={(prevValues, curValues) => prevValues.additional !== curValues.additionc
  {() => {
    return (
      <Form.Item name="other">
        <Input />
      </Form.Item>
    );
  }}
</Form.Item>
```

You can ref example to see detail.

messageVariables

You can modify the default verification information of Form.Item through `messageVariables`.

```
<Form>
  <Form.Item
    messageVariables={{ another: 'good' }}
    label="user"
    rules={[{ required: true, message: '${another} is required' }]}
  >
    <Input />
  </Form.Item>
  <Form.Item
    messageVariables={{ label: 'good' }}
    label={<span>user</span>}
    rules={[{ required: true, message: '${label} is required' }]}
  >
    <Input />
  </Form.Item>
</Form>
```

Form.List

Provides array management for fields.

Property	Description	Type	Default	Version
children	Render function	(fields: Field[], operation: { add, remove, move }, meta: { errors }) => React.ReactNode	-	
initialValue	Config sub default value. Form `initialValues` get higher priority when conflict	any[]	-	4.9.0
name	Field name, support array	NamePath	-	
rules	Validate rules, only support customize validator. Should work with ErrorList	{ validator, message }[]	-	4.7.0

```
<Form.List>
  {(fields) => (
    <div>
      {fields.map((field) => (
        <Form.Item {...field}>
          <Input />
        </Form.Item>
      ))}
    </div>
  )}
</Form.List>
```

Note: You should not configure Form.Item `initialValue` under Form.List. It always should be configured by Form.List `initialValue` or Form `initialValues`.

operation

Some operator functions in render form of Form.List.

Property	Description	Type	Default	Version
add	add form item	(defaultValue?: any, insertIndex?: number) => void	insertIndex	4.6.0
move	move form item	(from: number, to: number) => void	-	
remove	remove form item	(index: number | number[]) => void	number[]	4.5.0

Form.ErrorList

New in 4.7.0. Show error messages, should only work with `rules` of Form.List. See example.

Property	Description	Type	Default
errors	Error list	ReactNode[]	-

Form.Provider

Provide linkage between forms. If a sub form with `name` prop update, it will auto trigger Provider related events. See example.

Property	Description	Type	Default
onFormChange	Triggered when a sub form field updates	function(formName: string, info: { changedFields, forms })	-
onFormFinish	Triggered when a sub form submits	function(formName: string, info: { values, forms })	-

```
<Form.Provider
  onFormFinish={(name) => {
    if (name === 'form1') {
      // Do something...
    }
  }}
>
  <Form name="form1">...</Form>
  <Form name="form2">...</Form>
</Form.Provider>
```

FormInstance

Name	Description	Type	Version
getFieldError	Get the error messages by the field name	(name: NamePath) => string[]	
getFieldInstance	Get field instance	(name: NamePath) => any	4.4.0
getFieldsError	Get the error messages by the fields name. Return as an array	(nameList?: NamePath[]) => FieldError[]	
getFieldsValue	Get values by a set of field names. Return according to the corresponding structure. Default return mounted field value, but you can use `getFieldsValue(true)` to get all values	(nameList?: NamePath[], filterFunc?: (meta: { touched: boolean, validating:	

Name	Description	Type	Version
		boolean }) => boolean) => any	
getFieldValue	Get the value by the field name	(name: NamePath) => any	
isFieldsTouched	Check if fields have been operated. Check if all fields is touched when `allTouched` is `true`	(nameList?: NamePath[], allTouched?: boolean) => boolean	
isFieldTouched	Check if a field has been operated	(name: NamePath) => boolean	
isFieldValidating	Check field if is in validating	(name: NamePath) => boolean	
resetFields	Reset fields to `initialValues`	(fields?: NamePath[]) => void	
scrollToField	Scroll to field position	(name: NamePath, options: [ScrollOptions]) => void	
setFields	Set fields status	(fields: FieldData[]) => void	
setFieldValue	Set fields value(Will directly pass to form store. If you do not want to modify passed object, please clone first)	(name: NamePath, value: any) => void	4.22.0
setFieldsValue	Set fields value(Will directly pass to form store. If you do not want to modify passed object, please clone first). Use `setFieldValue` instead if you want to only config single value in Form.List	(values) => void	
submit	Submit the form. It's same as click `submit` button	() => void	
validateFields	Validate fields	(nameList?: NamePath[]) => Promise	

validateFields return sample

```
validateFields()
  .then((values) => {
    /*
  values:
    {
      username: 'username',
      password: 'password',
    }
```

```
  */
})
.catch((errorInfo) => {
  /*
  errorInfo:
    {
      values: {
        username: 'username',
        password: 'password',
      },
      errorFields: [
        { name: ['password'], errors: ['Please input your Password!'] },
      ],
      outOfDate: false,
    }
  */
});
```

Hooks

Form.useForm

`type Form.useForm = (): [FormInstance]`

Create Form instance to maintain data store.

Form.useFormInstance

`type Form.useFormInstance = (): FormInstance`

Added in `4.20.0`. Get current context form instance to avoid pass as props between components:

```
const Sub = () => {
  const form = Form.useFormInstance();

  return <Button onClick={() => form.setFieldsValue({})} />;
};

export default () => {
  const [form] = Form.useForm();

  return (
    <Form form={form}>
      <Sub />
    </Form>
  );
};
```

Form.useWatch

`type Form.useWatch = (namePath: NamePath, formInstance?: FormInstance | WatchOptions): Value`

Watch the value of a field. You can use this to interact with other hooks like `useSWR` to reduce development costs:

```tsx
const Demo = () => {
  const [form] = Form.useForm();
  const userName = Form.useWatch('username', form);

  const { data: options } = useSWR(`/api/user/${userName}`, fetcher);

  return (
    <Form form={form}>
      <Form.Item name="username">
        <AutoComplete options={options} />
      </Form.Item>
    </Form>
  );
};
```

If your component is wrapped by `Form.Item`, you can omit the second argument, `Form.useWatch` will find the nearest `FormInstance` automatically.

By default `useWatch` only watches the registered field. If you want to watch the unregistered field, please use `preserve`:

```tsx
const Demo = () => {
  const [form] = Form.useForm();

  const age = Form.useWatch('age', { form, preserve: true });
  console.log(age);

  return (
    <div>
      <Button onClick={() => form.setFieldValue('age', 2)}>Update</Button>
      <Form form={form}>
        <Form.Item name="name">
          <Input />
        </Form.Item>
      </Form>
    </div>
  );
};
```

Form.Item.useStatus

`type Form.Item.useStatus = (): { status: ValidateStatus | undefined, errors: ReactNode[], warnings: ReactNode[] }`

Added in `4.22.0`. Could be used to get validate status of Form.Item. If this hook is not used under Form.Item, `status` would be `undefined`. Added `error` and `warnings` in `5.4.0`, Could be used to get error messages and warning messages of Form.Item:

```tsx
const CustomInput = ({ value, onChange }) => {
  const { status, errors } = Form.Item.useStatus();
  return (
    <input
      value={value}
      onChange={onChange}
```

```
        className={`custom-input-${status}`}
        placeholder={(errors.length && errors[0]) || ''}
      />
    );
  };

  export default () => (
    <Form>
      <Form.Item name="username">
        <CustomInput />
      </Form.Item>
    </Form>
  );
```

Difference between other data fetching method

Form only update the Field which changed to avoid full refresh perf issue. Thus you can not get real time value with `getFieldsValue` in render. And `useWatch` will rerender current component to sync with latest value. You can also use Field renderProps to get better performance if only want to do conditional render. If component no need care field value change, you can use `onValuesChange` to give to parent component to avoid current one rerender.

Interface

NamePath

`string | number | (string | number)[]`

FieldData

Name	Description	Type
errors	Error messages	string[]
warnings	Warning messages	string[]
name	Field name path	NamePath[]
touched	Whether is operated	boolean
validating	Whether is in validating	boolean
value	Field value	any

Rule

Rule supports a config object, or a function returning config object:

```
type Rule = RuleConfig | ((form: FormInstance) => RuleConfig);
```

Name	Description	Type	Version
defaultField	Validate rule for all array elements, valid when `type` is `array`	rule	

Name	Description	Type	Version
enum	Match enum value. You need to set `type` to `enum` to enable this	any[]	
fields	Validate rule for child elements, valid when `type` is `array` or `object`	Record<string, rule>	
len	Length of string, number, array	number	
max	`type` required: max length of `string`, `number`, `array`	number	
message	Error message. Will auto generate by template if not provided	string	
min	`type` required: min length of `string`, `number`, `array`	number	
pattern	Regex pattern	RegExp	
required	Required field	boolean	
transform	Transform value to the rule before validation	(value) => any	
type	Normally `string` | `number` | `boolean` | `url` | `email`. More type to ref here	string	
validateTrigger	Set validate trigger event. Must be the sub set of `validateTrigger` in Form.Item	string | string[]	
validator	Customize validation rule. Accept Promise as return. See example	(rule, value) => Promise	
warningOnly	Warning only. Not block form submit	boolean	4.17.0
whitespace	Failed if only has whitespace, only work with `type: 'string'` rule	boolean	

WatchOptions

Name	Description	Type	Default	Version
form	Form instance	FormInstance	Current form in context	5.4.0
preserve	Whether to watch the field which has no matched `Form.Item`	boolean	false	5.4.0

Design Token

▾ Global Token

Token Name	Description	Type	Default Value
colorBorder	Default border color, used to separate different elements, such as: form separator, card separator, etc.	`string`	`#d9d9d9`
colorError	Used to represent the visual elements of the operation failure, such as the error Button, error Result component, etc.	`string`	`#ff4d4f`
colorPrimary	Brand color is one of the most direct visual elements to reflect the characteristics and communication of the product. After you have selected the brand color, we will automatically generate a complete color palette and assign it effective design semantics.	`string`	`#1677ff`
colorSuccess	Used to represent the token sequence of operation success, such as Result, Progress and other components will use these map tokens.	`string`	`#52c41a`
colorText	Default text color which comply with W3C standards, and this color is also the darkest neutral color.	`string`	`rgba(0, 0, 0, 0.88)`
colorTextDescription	Control the font color of text description.	`string`	`rgba(0, 0, 0, 0.45)`
colorTextHeading	Control the font color of heading.	`string`	`rgba(0, 0, 0, 0.88)`
colorWarning	Used to represent the warning map token, such as Notification, Alert, etc. Alert or Control component(like Input) will use these map tokens.	`string`	`#faad14`
controlHeight	The height of the basic controls such as buttons and input boxes in Ant Design	`number`	32
controlHeightLG	LG component height	`number`	40
controlHeightSM	SM component height	`number`	24
controlOutline	Control the outline color of input component.	`string`	`rgba(5, 145, 255, 0.1)`
controlOutlineWidth	Control the outline width of input component.	`number`	2
fontFamily	The font family of Ant Design prioritizes the default interface font of the system, and provides a set of alternative font libraries that are suitable for screen display to maintain the readability and readability of the font under different platforms and browsers, reflecting the friendly, stable and professional characteristics.	`string`	-apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, 'Helvetica Neue', Arial, 'Noto Sans', sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol', 'Noto Color Emoji'
fontSize	The most widely used font size in the design system, from which the text gradient will be derived.	`number`	14

Token Name	Description	Type	Default Value
fontSizeLG	Large font size	`number`	16
lineHeight	Line height of text.	`number`	1.5714285714285714
lineType	Border style of base components	`string`	solid
lineWidth	Border width of base components	`number`	1
margin	Control the margin of an element, with a medium size.	`number`	16
marginLG	Control the margin of an element, with a large size.	`number`	24
marginXS	Control the margin of an element, with a small size.	`number`	8
marginXXS	Control the margin of an element, with the smallest size.	`number`	4
motionDurationMid	Motion speed, medium speed. Used for medium element animation interaction.	`string`	0.2s
motionDurationSlow	Motion speed, slow speed. Used for large element animation interaction.	`string`	0.3s
motionEaseInOut	Preset motion curve.	`string`	cubic-bezier(0.645, 0.045, 0.355, 1)
motionEaseOut	Preset motion curve.	`string`	cubic-bezier(0.215, 0.61, 0.355, 1)
motionEaseOutBack	Preset motion curve.	`string`	cubic-bezier(0.12, 0.4, 0.29, 1.46)
paddingSM	Control the small padding of the element.	`number`	12
paddingXS	Control the extra small padding of the element.	`number`	8
screenLGMax	Control the maximum width of large screens.	`number`	1199
screenMDMax	Control the maximum width of medium screens.	`number`	991
screenSMMax	Control the maximum width of small screens.	`number`	767
screenXSMax	Control the maximum width of extra small screens.	`number`	575

FAQ

Custom validator not working

It may be caused by your `validator` if it has some errors that prevents `callback` to be called. You can use `async` instead or use `try...catch` to catch the error:

```
validator: async (rule, value) => {
  throw new Error('Something wrong!');
}

// or

validator(rule, value, callback) => {
  try {
    throw new Error('Something wrong!');
  } catch (err) {
    callback(err);
  }
}
```

How does `name` fill value when it's an array?

`name` will fill value by array order. When there exists number in it and no related field in form store, it will auto convert field to array. If you want to keep it as object, use string like: `['1', 'name']`.

Why is there a form warning when used in Modal?

> Warning: Instance created by `useForm` is not connect to any Form element. Forget to pass `form` prop?

Before Modal opens, children elements do not exist in the view. You can set `forceRender` on Modal to pre-render its children. Click here to view an example.

Why is component `defaultValue` not working when inside Form.Item?

Components inside Form.Item with name property will turn into controlled mode, which makes `defaultValue` not work anymore. Please try `initialValues` of Form to set default value.

Why can not call `ref` of Form at first time?

`ref` only receives the mounted instance. please ref React official doc: https://reactjs.org/docs/refs-and-the-dom.html#accessing-refs

Why will `resetFields` re-mount component?

`resetFields` will re-mount component under Field to clean up customize component side effects (like async data, cached state, etc.). It's by design.

Difference between Form initialValues and Item initialValue?

In most case, we always recommend to use Form `initialValues`. Use Item `initialValue` only with dynamic field usage. Priority follows the rules:

1. Form `initialValues` is the first priority
2. Field `initialValue` is secondary *. Does not work when multiple Item with same `name` setting the `initialValue`

Why does `onFieldsChange` trigger three times on change when field sets `rules`?

Validating is also part of the value updating. It pass follow steps:

1. Trigger value change
2. Rule validating
3. Rule validated

In each `onFieldsChange`, you will get `false` > `true` > `false` with `isFieldValidating`.

Why doesn't Form.List support `label` and need ErrorList to show errors?

Form.List use renderProps which mean internal structure is flexible. Thus `label` and `error` can not have best place. If you want to use antd `label`, you can wrap with Form.Item instead.

Why can't Form.Item `dependencies` work on Form.List field?

Your name path should also contain Form.List `name`:

```jsx
<Form.List name="users">
  {(fields) =>
    fields.map((field) => (
      <React.Fragment key={field.key}>
        <Form.Item name={[field.name, 'name']} {...someRest1} />
        <Form.Item name={[field.name, 'age']} {...someRest1} />
      </React.Fragment>
    ))
  }
</Form.List>
```

dependencies should be `['users', 0, 'name']`

Why doesn't `normalize` support async?

React can not get correct interaction of controlled component with async value update. When user trigger `onChange`, component will do no response since `value` update is async. If you want to trigger value update async, you should use customize component to handle value state internal and pass sync value control to Form instead.

`scrollToFirstError` and `scrollToField` not working on custom form control?

See similar issues: #28370 #27994

`scrollToFirstError` and `scrollToField` deps on `id` attribute passed to form control, please make sure that it hasn't been ignored in your custom form control. Check codesandbox for solution.

`setFieldsValue` do not trigger `onFieldsChange` or `onValuesChange`?

It's by design. Only user interactive can trigger the change event. This design is aim to avoid call `setFieldsValue` in change event which may makes loop calling.