When to use {#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

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

type FieldType = {
  username?: string;
  password?: string;
  remember?: string;
};

const onFinish: FormProps<FieldType>['onFinish'] = (values) => {
  console.log('Success:', values);
};

const onFinishFailed: FormProps<FieldType>['onFinishFailed'] = (errorInfo)
=> {
  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<FieldType>
      label="Username"
      name="username"
      rules={[{ required: true, message: 'Please input your username!' }]}
    >
      <Input />
    </Form.Item>
```

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

    <Form.Item<FieldType> name="remember" valuePropName="checked" label=
{null}>
      <Checkbox>Remember me</Checkbox>
    </Form.Item>

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

export default App;
```

Form methods

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

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.Item name="gender" label="Gender" rules={[{ required: true }]}>
      <Select
        placeholder="Select a option and change input text above"
        onChange={onGenderChange}
        allowClear
      >
        <Option value="male">male</Option>
        <Option value="female">female</Option>
        <Option value="other">other</Option>
      </Select>
    </Form.Item>
    <Form.Item
```

```
        noStyle
        shouldUpdate={(prevValues, currentValues) => prevValues.gender !==
currentValues.gender}
      >
        {({ getFieldValue }) =>
          getFieldValue('gender') === 'other' ? (
            <Form.Item name="customizeGender" label="Customize Gender"
rules={[{ required: true }]}>
              <Input />
            </Form.Item>
          ) : null
        }
      </Form.Item>
      <Form.Item {...tailLayout}>
        <Space>
          <Button type="primary" htmlType="submit">
            Submit
          </Button>
          <Button htmlType="button" onClick={onReset}>
            Reset
          </Button>
          <Button type="link" htmlType="button" onClick={onFill}>
            Fill form
          </Button>
        </Space>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Form Layout

```
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);
  };
```

```
  return (
    <Form
      layout={formLayout}
      form={form}
      initialValues={{ layout: formLayout }}
      onValuesChange={onFormLayoutChange}
      style={{ maxWidth: formLayout === 'inline' ? 'none' : 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>
        <Button type="primary">Submit</Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Form mix layout

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

const App: React.FC = () => (
  <>
    <Form
      name="layout-multiple-horizontal"
      layout="horizontal"
      labelCol={{ span: 4 }}
      wrapperCol={{ span: 20 }}
    >
      <Form.Item label="horizontal" name="horizontal" rules={[{ required:
true }]}>
```

```
        <Input />
      </Form.Item>
      <Form.Item
        layout="vertical"
        label="vertical"
        name="vertical"
        rules={[{ required: true }]}
        labelCol={{ span: 24 }}
        wrapperCol={{ span: 24 }}
      >
        <Input />
      </Form.Item>
    </Form>
    <br />
    <Form
      name="layout-multiple-vertical"
      layout="vertical"
      labelCol={{ span: 4 }}
      wrapperCol={{ span: 20 }}
    >
      <Form.Item label="vertical" name="vertical" rules={[{ required: true
}]}>
        <Input />
      </Form.Item>
      <Form.Item
        layout="horizontal"
        label="horizontal"
        name="horizontal"
        rules={[{ required: true }]}
      >
        <Input />
      </Form.Item>
    </Form>
  </>
);

export default App;
```

Form disabled

```
import React, { useState } from 'react';
import { PlusOutlined } from '@ant-design/icons';
import {
  Button,
  Cascader,
  Checkbox,
```

```
  ColorPicker,
  DatePicker,
  Form,
  Input,
  InputNumber,
  Radio,
  Rate,
  Select,
  Slider,
  Switch,
  TreeSelect,
  Upload,
} from 'antd';

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

const normFile = (e: any) => {
  if (Array.isArray(e)) {
    return e;
  }
  return e?.fileList;
};

const FormDisabledDemo: React.FC = () => {
  const [componentDisabled, setComponentDisabled] = useState<boolean>
(true);

  return (
    <>
      <Checkbox
        checked={componentDisabled}
        onChange={(e) => setComponentDisabled(e.target.checked)}
      >
        Form disabled
      </Checkbox>
      <Form
        labelCol={{ span: 4 }}
        wrapperCol={{ span: 14 }}
        layout="horizontal"
        disabled={componentDisabled}
        style={{ maxWidth: 600 }}
      >
        <Form.Item label="Checkbox" name="disabled"
valuePropName="checked">
          <Checkbox>Checkbox</Checkbox>
```

```
        </Form.Item>
        <Form.Item label="Radio">
          <Radio.Group>
            <Radio value="apple"> Apple </Radio>
            <Radio value="pear"> Pear </Radio>
          </Radio.Group>
        </Form.Item>
        <Form.Item label="Input">
          <Input />
        </Form.Item>
        <Form.Item label="Select">
          <Select>
            <Select.Option value="demo">Demo</Select.Option>
          </Select>
        </Form.Item>
        <Form.Item label="TreeSelect">
          <TreeSelect
            treeData={[
              { title: 'Light', value: 'light', children: [{ title:
'Bamboo', value: 'bamboo' }] },
            ]}
          />
        </Form.Item>
        <Form.Item label="Cascader">
          <Cascader
            options={[
              {
                value: 'zhejiang',
                label: 'Zhejiang',
                children: [
                  {
                    value: 'hangzhou',
                    label: 'Hangzhou',
                  },
                ],
              },
            ]}
          />
        </Form.Item>
        <Form.Item label="DatePicker">
          <DatePicker />
        </Form.Item>
        <Form.Item label="RangePicker">
          <RangePicker />
        </Form.Item>
        <Form.Item label="InputNumber">
```

```
          <InputNumber />
        </Form.Item>
        <Form.Item label="TextArea">
          <TextArea rows={4} />
        </Form.Item>
        <Form.Item label="Switch" valuePropName="checked">
          <Switch />
        </Form.Item>
        <Form.Item label="Upload" valuePropName="fileList"
getValueFromEvent={normFile}>
          <Upload action="/upload.do" listType="picture-card">
            <button
              style={{ color: 'inherit', cursor: 'inherit', border: 0,
background: 'none' }}
              type="button"
            >
              <PlusOutlined />
              <div style={{ marginTop: 8 }}>Upload</div>
            </button>
          </Upload>
        </Form.Item>
        <Form.Item label="Button">
          <Button>Button</Button>
        </Form.Item>
        <Form.Item label="Slider">
          <Slider />
        </Form.Item>
        <Form.Item label="ColorPicker">
          <ColorPicker />
        </Form.Item>
        <Form.Item label="Rate">
          <Rate />
        </Form.Item>
      </Form>
    </>
  );
};

export default () => <FormDisabledDemo />;
```

Form variants

v5.13.0

```
import React from 'react';
import {
```

```
  Button,
  Cascader,
  DatePicker,
  Form,
  Input,
  InputNumber,
  Mentions,
  Segmented,
  Select,
  TreeSelect,
} from 'antd';

const { RangePicker } = DatePicker;

const formItemLayout = {
  labelCol: {
    xs: { span: 24 },
    sm: { span: 6 },
  },
  wrapperCol: {
    xs: { span: 24 },
    sm: { span: 14 },
  },
};

const App: React.FC = () => {
  const [form] = Form.useForm();
  const variant = Form.useWatch('variant', form);
  return (
    <Form
      {...formItemLayout}
      form={form}
      variant={variant || 'filled'}
      style={{ maxWidth: 600 }}
      initialValues={{ variant: 'filled' }}
    >
      <Form.Item label="Form variant" name="variant">
        <Segmented options={['outlined', 'filled', 'borderless',
'underlined']} />
      </Form.Item>

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

```
<Form.Item
  label="InputNumber"
  name="InputNumber"
  rules={[{ required: true, message: 'Please input!' }]}
>
  <InputNumber style={{ width: '100%' }} />
</Form.Item>

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

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

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

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

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

```
      </Form.Item>

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

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

      <Form.Item wrapperCol={{ offset: 6, span: 16 }}>
        <Button type="primary" htmlType="submit">
          Submit
        </Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Required style

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

type RequiredMark = boolean | 'optional' | 'customize';

const customizeRequiredMark = (label: React.ReactNode, { required }: {
required: boolean }) => (
  <>
    {required ? <Tag color="error">Required</Tag> : <Tag
color="warning">optional</Tag>}
    {label}
  </>
);
```

```tsx
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 === 'customize' ? customizeRequiredMark :
requiredMark}
    >
      <Form.Item label="Required Mark" name="requiredMarkValue">
        <Radio.Group>
          <Radio.Button value>Default</Radio.Button>
          <Radio.Button value="optional">Optional</Radio.Button>
          <Radio.Button value={false}>Hidden</Radio.Button>
          <Radio.Button value="customize">Customize</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 size

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

```
      <Form.Item label="Select">
        <Select>
          <Select.Option value="demo">Demo</Select.Option>
        </Select>
      </Form.Item>
      <Form.Item label="TreeSelect">
        <TreeSelect
          treeData={[
            { title: 'Light', value: 'light', children: [{ title: 'Bamboo',
value: 'bamboo' }] },
          ]}
        />
      </Form.Item>
      <Form.Item label="Cascader">
        <Cascader
          options={[
            {
              value: 'zhejiang',
              label: 'Zhejiang',
              children: [{ value: 'hangzhou', label: 'Hangzhou' }],
            },
          ]}
        />
      </Form.Item>
      <Form.Item label="DatePicker">
        <DatePicker />
      </Form.Item>
      <Form.Item label="InputNumber">
        <InputNumber />
      </Form.Item>
      <Form.Item label="Switch" valuePropName="checked">
        <Switch />
      </Form.Item>
      <Form.Item label="Button">
        <Button>Button</Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

label can wrap

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

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

    <Form.Item label="A super long label text" 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

```tsx
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;
```

Watch Hooks

```
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);
  // The selector is static and does not support closures.
  const customValue = Form.useWatch((values) => `name: ${values.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>
        <pre>Custom Value: {customValue}</pre>
      </Typography>
    </>
  );
};

export default Demo;
```

Validate Trigger

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

const App: React.FC = () => (
  <Form name="trigger" style={{ maxWidth: 600 }} layout="vertical"
autoComplete="off">
    <Alert message="Use 'max' rule, continue type chars to see it" />

    <Form.Item
      hasFeedback
      label="Field A"
      name="field_a"
      validateTrigger="onBlur"
      rules={[{ max: 3 }]}
    >
      <Input placeholder="Validate required onBlur" />
    </Form.Item>
```

```
    <Form.Item
      hasFeedback
      label="Field B"
      name="field_b"
      validateDebounce={1000}
      rules={[{ max: 3 }]}
    >
      <Input placeholder="Validate required debounce after 1s" />
    </Form.Item>

    <Form.Item
      hasFeedback
      label="Field C"
      name="field_c"
      validateFirst
      rules={[{ max: 6 }, { max: 3, message: 'Continue input to exceed 6
chars' }]}
    >
      <Input placeholder="Validate one by one" />
    </Form.Item>
  </Form>
);

export default App;
```

Validate Only

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

interface SubmitButtonProps {
  form: FormInstance;
}

const SubmitButton: React.FC<React.PropsWithChildren<SubmitButtonProps>> =
({ form, children }) => {
  const [submittable, setSubmittable] = React.useState<boolean>(false);

  // Watch all values
  const values = Form.useWatch([], form);

  React.useEffect(() => {
    form
      .validateFields({ validateOnly: true })
      .then(() => setSubmittable(true))
```

```
      .catch(() => setSubmittable(false));
  }, [form, values]);

  return (
    <Button type="primary" htmlType="submit" disabled={!submittable}>
      {children}
    </Button>
  );
};

const App: React.FC = () => {
  const [form] = Form.useForm();
  return (
    <Form form={form} name="validateOnly" layout="vertical"
autoComplete="off">
      <Form.Item name="name" label="Name" rules={[{ required: true }]}>
        <Input />
      </Form.Item>
      <Form.Item name="age" label="Age" rules={[{ required: true }]}>
        <Input />
      </Form.Item>
      <Form.Item>
        <Space>
          <SubmitButton form={form}>Submit</SubmitButton>
          <Button htmlType="reset">Reset</Button>
        </Space>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Path Prefix

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

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

interface MyFormItemGroupProps {
  prefix: string | number | (string | number)[];
}

function toArr(str: string | number | (string | number)[]): (string |
```

```
number)[] {
  return Array.isArray(str) ? str : [str];
}

const MyFormItemGroup:
React.FC<React.PropsWithChildren<MyFormItemGroupProps>> = ({
  prefix,
  children,
}) => {
  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>
```

```
      <Button type="primary" htmlType="submit">
        Submit
      </Button>
    </Form>
  );
};

export default App;
```

Dynamic Form Item

```
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}
                  validateTrigger={['onChange', 'onBlur']}
                  rules={[
                    {
                      required: true,
                      whitespace: true,
                      message: "Please input passenger's name or delete
this field.",
                    },
                  ]}
                  noStyle
                >
                  <Input placeholder="passenger name" style={{ width: '60%'
}} />
                </Form.Item>
                {fields.length > 1 ? (
                  <MinusCircleOutlined
                    className="dynamic-delete-button"
                    onClick={() => remove(field.name)}
                  />
                ) : null}
              </Form.Item>
            ))}
```

```
            <Form.Item>
              <Button
                type="dashed"
                onClick={() => add()}
                style={{ width: '60%' }}
                icon={<PlusOutlined />}
              >
                Add field
              </Button>
              <Button
                type="dashed"
                onClick={() => {
                  add('The head item', 0);
                }}
                style={{ width: '60%', marginTop: '20px' }}
                icon={<PlusOutlined />}
              >
                Add field at head
              </Button>
              <Form.ErrorList errors={errors} />
            </Form.Item>
          </>
        )}
      </Form.List>
      <Form.Item>
        <Button type="primary" htmlType="submit">
          Submit
        </Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Dynamic Form nest Items

```
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;
```

Dynamic Form nest pure Items

Debug

```
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_no_style"
    onFinish={onFinish}
    style={{ maxWidth: 600 }}
    autoComplete="off"
  >
    <Form.Item label="Users">
      <Form.List name="users">
        {(fields, { add, remove }) => (
          <>
            {fields.map((field) => (
              <Space key={field.key} style={{ marginBottom: 16 }}>
                <Form.Item noStyle name={[field.name, 'lastName']} rules=
{[{ required: true }]}>
                  <Input placeholder="Last Name" />
                </Form.Item>
                <Form.Item noStyle name={[field.name, 'firstName']} rules=
{[{ required: true }]}>
                  <Input placeholder="First Name" />
                </Form.Item>
                <MinusCircleOutlined
                  onClick={() => {
                    remove(field.name);
                  }}
                />
              </Space>
            ))}
            <Form.Item>
              <Button type="dashed" onClick={() => add()} block icon=
{<PlusOutlined />}>
```

```
              Add field
            </Button>
          </Form.Item>
        </>
      )}
    </Form.List>
  </Form.Item>
  <Form.Item>
    <Button type="primary" htmlType="submit">
      Submit
    </Button>
  </Form.Item>
</Form>
);

export default App;
```

Complex Dynamic Form Item

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

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

  return (
    <Form
      labelCol={{ span: 6 }}
      wrapperCol={{ span: 18 }}
      form={form}
      name="dynamic_form_complex"
      style={{ maxWidth: 600 }}
      autoComplete="off"
      initialValues={{ items: [{}] }}
    >
      <Form.List name="items">
        {(fields, { add, remove }) => (
          <div style={{ display: 'flex', rowGap: 16, flexDirection:
'column' }}>
            {fields.map((field) => (
              <Card
                size="small"
                title={`Item ${field.name + 1}`}
                key={field.key}
                extra={
```

```
                  <CloseOutlined
                    onClick={() => {
                      remove(field.name);
                    }}
                  />
                }
              >
                <Form.Item label="Name" name={[field.name, 'name']}>
                  <Input />
                </Form.Item>

                {/* Nest Form.List */}
                <Form.Item label="List">
                  <Form.List name={[field.name, 'list']}>
                    {(subFields, subOpt) => (
                      <div style={{ display: 'flex', flexDirection:
'column', rowGap: 16 }}>
                        {subFields.map((subField) => (
                          <Space key={subField.key}>
                            <Form.Item noStyle name={[subField.name,
'first']}>
                              <Input placeholder="first" />
                            </Form.Item>
                            <Form.Item noStyle name={[subField.name,
'second']}>
                              <Input placeholder="second" />
                            </Form.Item>
                            <CloseOutlined
                              onClick={() => {
                                subOpt.remove(subField.name);
                              }}
                            />
                          </Space>
                        ))}
                        <Button type="dashed" onClick={() => subOpt.add()}
block>
                          + Add Sub Item
                        </Button>
                      </div>
                    )}
                  </Form.List>
                </Form.Item>
              </Card>
            ))}

            <Button type="dashed" onClick={() => add()} block>
```

```
              + Add Item
            </Button>
          </div>
        )}
      </Form.List>

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

export default App;
```

Nest

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

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

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}',
  },
};

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">
      <Input.TextArea />
    </Form.Item>
    <Form.Item label={null}>
      <Button type="primary" htmlType="submit">
        Submit
      </Button>
    </Form.Item>
  </Form>
);

export default App;
```

complex form control

```
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);
};
```

```tsx
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' }]}
        >
          <Input style={{ width: '50%' }} placeholder="Input street" />
        </Form.Item>
      </Space.Compact>
    </Form.Item>
    <Form.Item label="BirthDate" style={{ marginBottom: 0 }}>
      <Form.Item
        name="year"
```

```
        rules={[{ required: true }]}
        style={{ display: 'inline-block', width: 'calc(50% - 8px)' }}
      >
        <Input placeholder="Input birth year" />
      </Form.Item>
      <Form.Item
        name="month"
        rules={[{ required: true }]}
        style={{ display: 'inline-block', width: 'calc(50% - 8px)', margin:
'0 8px' }}
      >
        <Input placeholder="Input birth month" />
      </Form.Item>
    </Form.Item>
    <Form.Item label={null}>
      <Button type="primary" htmlType="submit">
        Submit
      </Button>
    </Form.Item>
  </Form>
);

export default App;
```

Customized Form Controls

```
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 {
  id?: string;
  value?: PriceValue;
  onChange?: (value: PriceValue) => void;
}

const PriceInput: React.FC<PriceInputProps> = (props) => {
  const { id, value = {}, onChange } = props;
```

```tsx
  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 id={id}>
      <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>
  );
};
```

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

  const checkPrice = (_: any, value: { number: number }) => {
    if (value.number > 0) {
      return Promise.resolve();
    }
    return Promise.reject(new Error('Price must be greater than zero!'));
  };

  return (
    <Form
      name="customized_form_controls"
      layout="inline"
      onFinish={onFinish}
      initialValues={{
        price: {
          number: 0,
          currency: 'rmb',
        },
      }}
    >
      <Form.Item name="price" label="Price" rules={[{ validator: checkPrice
}]}>
        <PriceInput />
      </Form.Item>
      <Form.Item>
        <Button type="primary" htmlType="submit">
          Submit
        </Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Store Form Data into Upper Component

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

const { Paragraph } = Typography;
```

```tsx
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>
      </Paragraph>
    </>
  );
};

export default App;
```

Control between forms

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

type FormInstance = GetRef<typeof 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>(null);
  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">
        <Form.Item name="name" label="User Name" rules={[{ required: true
}]}>
          <Input />
        </Form.Item>
        <Form.Item name="age" label="User Age" rules={[{ required: true
}]}>
          <InputNumber />
        </Form.Item>
      </Form>
    </Modal>
  );
};

const App: React.FC = () => {
  const [open, setOpen] = useState(false);

  const showUserModal = () => {
    setOpen(true);
  };

  const hideUserModal = () => {
    setOpen(false);
```

```
  };

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

  return (
    <Form.Provider
      onFormFinish={(name, { values, forms }) => {
        if (name === 'userForm') {
          const { basicForm } = forms;
          const users = basicForm.getFieldValue('users') || [];
          basicForm.setFieldsValue({ users: [...users, values] });
          setOpen(false);
        }
      }}
    >
      <Form {...layout} name="basicForm" onFinish={onFinish} style={{
maxWidth: 600 }}>
        <Form.Item name="group" label="Group Name" rules={[{ required: true
}]}>
          <Input />
        </Form.Item>

        {/* Create a hidden field to make Form instance record this */}
        <Form.Item name="users" noStyle />

        <Form.Item
          label="User List"
          shouldUpdate={(prevValues, curValues) => prevValues.users !==
curValues.users}
        >
          {({ getFieldValue }) => {
            const users: UserType[] = getFieldValue('users') || [];
            return users.length ? (
              <Flex vertical gap={8}>
                {users.map((user) => (
                  <Space key={user.name}>
                    <Avatar icon={<UserOutlined />} />
                    {`${user.name} - ${user.age}`}
                  </Space>
                ))}
              </Flex>
            ) : (
              <Typography.Text className="ant-form-text" type="secondary">
                ( <SmileOutlined /> No user yet. )
```

```
              </Typography.Text>
            );
          }}
        </Form.Item>
        <Form.Item {...tailLayout}>
          <Button htmlType="submit" type="primary">
            Submit
          </Button>
          <Button htmlType="button" style={{ margin: '0 8px' }} onClick=
{showUserModal}>
            Add User
          </Button>
        </Form.Item>
      </Form>

      <ModalForm open={open} onCancel={hideUserModal} />
    </Form.Provider>
  );
};

export default App;
```

Inline Login Form

```
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 [clientReady, setClientReady] = useState<boolean>(false);

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

  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 />} placeholder="Username" />
      </Form.Item>
      <Form.Item
        name="password"
        rules={[{ required: true, message: 'Please input your password!'
}]}
      >
        <Input prefix={<LockOutlined />} type="password"
placeholder="Password" />
      </Form.Item>
      <Form.Item shouldUpdate>
        {() => (
          <Button
            type="primary"
            htmlType="submit"
            disabled={
              !clientReady ||
              !form.isFieldsTouched(true) ||
              !!form.getFieldsError().filter(({ errors }) =>
errors.length).length
            }
          >
            Log in
          </Button>
        )}
      </Form.Item>
    </Form>
  );
};

export default App;
```

Login Form

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

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

```
  return (
    <Form
      name="login"
      initialValues={{ remember: true }}
      style={{ maxWidth: 360 }}
      onFinish={onFinish}
    >
      <Form.Item
        name="username"
        rules={[{ required: true, message: 'Please input your Username!'
}]}
      >
        <Input prefix={<UserOutlined />} placeholder="Username" />
      </Form.Item>
      <Form.Item
        name="password"
        rules={[{ required: true, message: 'Please input your Password!'
}]}
      >
        <Input prefix={<LockOutlined />} type="password"
placeholder="Password" />
      </Form.Item>
      <Form.Item>
        <Flex justify="space-between" align="center">
          <Form.Item name="remember" valuePropName="checked" noStyle>
            <Checkbox>Remember me</Checkbox>
          </Form.Item>
          <a href="">Forgot password</a>
        </Flex>
      </Form.Item>

      <Form.Item>
        <Button block type="primary" htmlType="submit">
          Log in
        </Button>
        or <a href="">Register now!</a>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Registration

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

const { Option } = Select;

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

const residences: CascaderProps<DataNodeType>['options'] = [
  {
    value: 'zhejiang',
    label: 'Zhejiang',
    children: [
      {
        value: 'hangzhou',
        label: 'Hangzhou',
        children: [
          {
            value: 'xihu',
            label: 'West Lake',
          },
        ],
      },
    ],
  },
  {
    value: 'jiangsu',
    label: 'Jiangsu',
    children: [
      {
        value: 'nanjing',
```

```
        label: 'Nanjing',
        children: [
          {
            value: 'zhonghuamen',
            label: 'Zhong Hua Men',
          },
        ],
      },
    ],
  },
];

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

const tailFormItemLayout = {
  wrapperCol: {
    xs: {
      span: 24,
      offset: 0,
    },
    sm: {
      span: 16,
      offset: 8,
    },
  },
};

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

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

  const prefixSelector = (
    <Form.Item name="prefix" noStyle>
      <Select style={{ width: 70 }}>
```

```
        <Option value="86">+86</Option>
        <Option value="87">+87</Option>
      </Select>
    </Form.Item>
  );

  const suffixSelector = (
    <Form.Item name="suffix" noStyle>
      <Select style={{ width: 70 }}>
        <Option value="USD">$</Option>
        <Option value="CNY">¥</Option>
      </Select>
    </Form.Item>
  );

  const [autoCompleteResult, setAutoCompleteResult] = useState<string[]>
([]);

  const onWebsiteChange = (value: string) => {
    if (!value) {
      setAutoCompleteResult([]);
    } else {
      setAutoCompleteResult(['.com', '.org', '.net'].map((domain) =>
`${value}${domain}`));
    }
  };

  const websiteOptions = autoCompleteResult.map((website) => ({
    label: website,
    value: website,
  }));

  return (
    <Form
      {...formItemLayout}
      form={form}
      name="register"
      onFinish={onFinish}
      initialValues={{ residence: ['zhejiang', 'hangzhou', 'xihu'], prefix:
'86' }}
      style={{ maxWidth: 600 }}
      scrollToFirstError
    >
      <Form.Item
        name="email"
        label="E-mail"
```

```
        rules={[
          {
            type: 'email',
            message: 'The input is not valid E-mail!',
          },
          {
            required: true,
            message: 'Please input your E-mail!',
          },
        ]}
      >
        <Input />
      </Form.Item>

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

      <Form.Item
        name="confirm"
        label="Confirm Password"
        dependencies={['password']}
        hasFeedback
        rules={[
          {
            required: true,
            message: 'Please confirm your password!',
          },
          ({ getFieldValue }) => ({
            validator(_, value) {
              if (!value || getFieldValue('password') === value) {
                return Promise.resolve();
              }
              return Promise.reject(new Error('The new password that you
entered do not match!'));
            },
```

```
          }),
        ]}
      >
        <Input.Password />
      </Form.Item>

      <Form.Item
        name="nickname"
        label="Nickname"
        tooltip="What do you want others to call you?"
        rules={[{ required: true, message: 'Please input your nickname!',
whitespace: true }]}
      >
        <Input />
      </Form.Item>

      <Form.Item
        name="residence"
        label="Habitual Residence"
        rules={[
          { type: 'array', required: true, message: 'Please select your
habitual residence!' },
        ]}
      >
        <Cascader options={residences} />
      </Form.Item>

      <Form.Item
        name="phone"
        label="Phone Number"
        rules={[{ required: true, message: 'Please input your phone
number!' }]}
      >
        <Input addonBefore={prefixSelector} style={{ width: '100%' }} />
      </Form.Item>

      <Form.Item
        name="donation"
        label="Donation"
        rules={[{ required: true, message: 'Please input donation amount!'
}]}
      >
        <InputNumber addonAfter={suffixSelector} style={{ width: '100%' }}
/>
      </Form.Item>
```

```
      <Form.Item
        name="website"
        label="Website"
        rules={[{ required: true, message: 'Please input website!' }]}
      >
        <AutoComplete options={websiteOptions} onChange={onWebsiteChange}
placeholder="website">
          <Input />
        </AutoComplete>
      </Form.Item>

      <Form.Item
        name="intro"
        label="Intro"
        rules={[{ required: true, message: 'Please input Intro' }]}
      >
        <Input.TextArea showCount maxLength={100} />
      </Form.Item>

      <Form.Item
        name="gender"
        label="Gender"
        rules={[{ required: true, message: 'Please select gender!' }]}
      >
        <Select placeholder="select your gender">
          <Option value="male">Male</Option>
          <Option value="female">Female</Option>
          <Option value="other">Other</Option>
        </Select>
      </Form.Item>

      <Form.Item label="Captcha" extra="We must make sure that your are a
human.">
        <Row gutter={8}>
          <Col span={12}>
            <Form.Item
              name="captcha"
              noStyle
              rules={[{ required: true, message: 'Please input the captcha
you got!' }]}
            >
              <Input />
            </Form.Item>
          </Col>
          <Col span={12}>
            <Button>Get captcha</Button>
```

```
          </Col>
        </Row>
      </Form.Item>

      <Form.Item
        name="agreement"
        valuePropName="checked"
        rules={[
          {
            validator: (_, value) =>
              value ? Promise.resolve() : Promise.reject(new Error('Should
accept agreement')),
          },
        ]}
        {...tailFormItemLayout}
      >
        <Checkbox>
          I have read the <a href="">agreement</a>
        </Checkbox>
      </Form.Item>
      <Form.Item {...tailFormItemLayout}>
        <Button type="primary" htmlType="submit">
          Register
        </Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Advanced search

```
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: React.CSSProperties = {
    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={[
                {
                  required: true,
                  message: 'Select something!',
                },
              ]}
              initialValue="1"
            >
              <Select>
                <Option value="1">

longlonglonglonglonglonglonglonglonglonglonglonglonglonglonglonglonglonglonglongl
                </Option>
                <Option value="2">222</Option>
              </Select>
            </Form.Item>
          )}
        </Col>,
```

```
      );
    }
    return children;
  };

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

  return (
    <Form form={form} name="advanced_search" style={formStyle} onFinish=
{onFinish}>
      <Row gutter={24}>{getFields()}</Row>
      <div style={{ textAlign: 'right' }}>
        <Space size="small">
          <Button type="primary" htmlType="submit">
            Search
          </Button>
          <Button
            onClick={() => {
              form.resetFields();
            }}
          >
            Clear
          </Button>
          <a
            style={{ fontSize: 12 }}
            onClick={() => {
              setExpand(!expand);
            }}
          >
            <DownOutlined rotate={expand ? 180 : 0} /> Collapse
          </a>
        </Space>
      </div>
    </Form>
  );
};

const App: React.FC = () => {
  const { token } = theme.useToken();

  const listStyle: React.CSSProperties = {
    lineHeight: '200px',
    textAlign: 'center',
    background: token.colorFillAlter,
```

```
    borderRadius: token.borderRadiusLG,
    marginTop: 16,
  };

  return (
    <>
      <AdvancedSearchForm />
      <div style={listStyle}>Search Result List</div>
    </>
  );
};

export default App;
```

Form in Modal to Create

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

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

const App: React.FC = () => {
  const [form] = Form.useForm();
  const [formValues, setFormValues] = useState<Values>();
  const [open, setOpen] = useState(false);

  const onCreate = (values: Values) => {
    console.log('Received values of form: ', values);
    setFormValues(values);
    setOpen(false);
  };

  return (
    <>
      <Button type="primary" onClick={() => setOpen(true)}>
        New Collection
      </Button>
      <pre>{JSON.stringify(formValues, null, 2)}</pre>
      <Modal
        open={open}
        title="Create a new collection"
        okText="Create"
```

```
        cancelText="Cancel"
        okButtonProps={{ autoFocus: true, htmlType: 'submit' }}
        onCancel={() => setOpen(false)}
        destroyOnClose
        modalRender={(dom) => (
          <Form
            layout="vertical"
            form={form}
            name="form_in_modal"
            initialValues={{ modifier: 'public' }}
            clearOnDestroy
            onFinish={(values) => onCreate(values)}
          >
            {dom}
          </Form>
        )}
      >
        <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>
      </Modal>
    </>
  );
};

export default App;
```

Time-related Controls

```
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 = () => (
```

```
  <Form
    name="time_related_controls"
    {...formItemLayout}
    onFinish={onFinish}
    style={{ maxWidth: 600 }}
  >
    <Form.Item name="date-picker" label="DatePicker" {...config}>
      <DatePicker />
    </Form.Item>
    <Form.Item name="date-time-picker" label="DatePicker[showTime]"
{...config}>
      <DatePicker showTime format="YYYY-MM-DD HH:mm:ss" />
    </Form.Item>
    <Form.Item name="month-picker" label="MonthPicker" {...config}>
      <DatePicker picker="month" />
    </Form.Item>
    <Form.Item name="range-picker" label="RangePicker" {...rangeConfig}>
      <RangePicker />
    </Form.Item>
    <Form.Item name="range-time-picker" label="RangePicker[showTime]"
{...rangeConfig}>
      <RangePicker showTime format="YYYY-MM-DD HH:mm:ss" />
    </Form.Item>
    <Form.Item name="time-picker" label="TimePicker" {...config}>
      <TimePicker />
    </Form.Item>
    <Form.Item label={null}>
      <Button type="primary" htmlType="submit">
        Submit
      </Button>
    </Form.Item>
  </Form>
);

export default App;
```

Handle Form Data Manually

```
import React, { useState } from 'react';
import type { InputNumberProps } from 'antd';
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 itself.';

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

  const onNumberChange: InputNumberProps['onChange'] = (value) => {
    setNumber({
      ...validatePrimeNumber(value as number),
      value: value as number,
    });
  };

  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

```
import React from 'react';
import { SmileOutlined } from '@ant-design/icons';
import {
  Cascader,
  DatePicker,
  Form,
  Input,
  InputNumber,
  Mentions,
  Select,
  TimePicker,
  TreeSelect,
} from 'antd';

const { Option } = Select;

const formItemLayout = {
  labelCol: {
    xs: { span: 24 },
    sm: { span: 6 },
  },
  wrapperCol: {
    xs: { span: 24 },
    sm: { span: 14 },
  },
};

const App: React.FC = () => (
  <Form {...formItemLayout} style={{ maxWidth: 600 }}>
    <Form.Item
      label="Fail"
      validateStatus="error"
      help="Should be combination of numbers & alphabets"
    >
```

```
      <Input placeholder="unavailable choice" id="error" />
    </Form.Item>

    <Form.Item label="Warning" validateStatus="warning">
      <Input placeholder="Warning" id="warning" prefix={<SmileOutlined />}
/>
    </Form.Item>

    <Form.Item
      label="Validating"
      hasFeedback
      validateStatus="validating"
      help="The information is being validated..."
    >
      <Input placeholder="I'm the content is being validated"
id="validating" />
    </Form.Item>

    <Form.Item label="Success" hasFeedback validateStatus="success">
      <Input placeholder="I'm the content" id="success" />
    </Form.Item>

    <Form.Item label="Warning" hasFeedback validateStatus="warning">
      <Input placeholder="Warning" id="warning2" />
    </Form.Item>

    <Form.Item
      label="Fail"
      hasFeedback
      validateStatus="error"
      help="Should be combination of numbers & alphabets"
    >
      <Input placeholder="unavailable choice" id="error2" />
    </Form.Item>

    <Form.Item label="Success" hasFeedback validateStatus="success">
      <DatePicker style={{ width: '100%' }} />
    </Form.Item>

    <Form.Item label="Warning" hasFeedback validateStatus="warning">
      <TimePicker style={{ width: '100%' }} />
    </Form.Item>

    <Form.Item label="Error" hasFeedback validateStatus="error">
      <DatePicker.RangePicker style={{ width: '100%' }} />
    </Form.Item>
```

```
    <Form.Item label="Error" hasFeedback validateStatus="error">
      <Select placeholder="I'm Select" allowClear>
        <Option value="1">Option 1</Option>
        <Option value="2">Option 2</Option>
        <Option value="3">Option 3</Option>
      </Select>
    </Form.Item>

    <Form.Item
      label="Validating"
      hasFeedback
      validateStatus="error"
      help="Something breaks the rule."
    >
      <Cascader placeholder="I'm Cascader" options={[{ value: 'xx', label:
'xx' }]} allowClear />
    </Form.Item>

    <Form.Item label="Warning" hasFeedback validateStatus="warning"
help="Need to be checked">
      <TreeSelect
        placeholder="I'm TreeSelect"
        treeData={[{ value: 'xx', label: 'xx' }]}
        allowClear
      />
    </Form.Item>

    <Form.Item label="inline" style={{ marginBottom: 0 }}>
      <Form.Item
        validateStatus="error"
        help="Please select right date"
        style={{ display: 'inline-block', width: 'calc(50% - 12px)' }}
      >
        <DatePicker />
      </Form.Item>
      <span
        style={{ display: 'inline-block', width: '24px', lineHeight:
'32px', textAlign: 'center' }}
      >
        -
      </span>
      <Form.Item style={{ display: 'inline-block', width: 'calc(50% -
12px)' }}>
        <DatePicker />
      </Form.Item>
```

```
    </Form.Item>

    <Form.Item label="Success" hasFeedback validateStatus="success">
      <InputNumber style={{ width: '100%' }} />
    </Form.Item>

    <Form.Item label="Success" hasFeedback validateStatus="success">
      <Input allowClear placeholder="with allowClear" />
    </Form.Item>

    <Form.Item label="Warning" hasFeedback validateStatus="warning">
      <Input.Password placeholder="with input password" />
    </Form.Item>

    <Form.Item label="Error" hasFeedback validateStatus="error">
      <Input.Password allowClear placeholder="with input password and
allowClear" />
    </Form.Item>

    <Form.Item label="Success" hasFeedback validateStatus="success">
      <Input.OTP />
    </Form.Item>
    <Form.Item label="Warning" hasFeedback validateStatus="warning">
      <Input.OTP />
    </Form.Item>

    <Form.Item label="Error" hasFeedback validateStatus="error">
      <Input.OTP />
    </Form.Item>

    <Form.Item label="Fail" validateStatus="error" hasFeedback>
      <Mentions />
    </Form.Item>

    <Form.Item label="Fail" validateStatus="error" hasFeedback help="Should
have something">
      <Input.TextArea allowClear showCount />
    </Form.Item>
  </Form>
);

export default App;
```

Dynamic Rules

```tsx
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}>
          Nickname is required
        </Checkbox>
      </Form.Item>
      <Form.Item {...formTailLayout}>
        <Button type="primary" onClick={onCheck}>
          Check
        </Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Dependencies

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

const App: React.FC = () => {
  const [form] = Form.useForm();
  return (
    <Form
      form={form}
      name="dependencies"
      autoComplete="off"
      style={{ maxWidth: 600 }}
      layout="vertical"
    >
      <Alert message=" Try modify `Password2` and then modify `Password`"
type="info" showIcon />

      <Form.Item label="Password" name="password" rules={[{ required: true
}]}>
        <Input />
      </Form.Item>
```

```
      {/* Field */}
      <Form.Item
        label="Confirm Password"
        name="password2"
        dependencies={['password']}
        rules={[
          {
            required: true,
          },
          ({ getFieldValue }) => ({
            validator(_, value) {
              if (!value || getFieldValue('password') === value) {
                return Promise.resolve();
              }
              return Promise.reject(new Error('The new password that you
entered do not match!'));
            },
          }),
        ]}
      >
        <Input />
      </Form.Item>

      {/* Render Props */}
      <Form.Item noStyle dependencies={['password2']}>
        {() => (
          <Typography>
            <p>
              Only Update when <code>password2</code> updated:
            </p>
            <pre>{JSON.stringify(form.getFieldsValue(), null, 2)}</pre>
          </Typography>
        )}
      </Form.Item>
    </Form>
  );
};

export default App;
```

getValueProps + normalize

```
import React from 'react';
import type { FormProps } from 'antd';
import { Button, DatePicker, Form } from 'antd';
import dayjs from 'dayjs';
```

```
const dateTimestamp = dayjs('2024-01-01').valueOf();

type FieldType = {
  date?: string;
};

const onFinish: FormProps<FieldType>['onFinish'] = (values) => {
  console.log('Success:', values);
};

const App: React.FC = () => (
  <Form
    name="getValueProps"
    labelCol={{ span: 8 }}
    wrapperCol={{ span: 16 }}
    style={{ maxWidth: 600 }}
    initialValues={{ date: dateTimestamp }}
    onFinish={onFinish}
    autoComplete="off"
  >
    <Form.Item<FieldType>
      label="Date"
      name="date"
      rules={[{ required: true }]}
      getValueProps={(value) => ({ value: value && dayjs(Number(value)) })}
      normalize={(value) => value && `${dayjs(value).valueOf()}`}
    >
      <DatePicker />
    </Form.Item>

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

export default App;
```

Slide to error field

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

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

  return (
    <Form
      form={form}
      scrollToFirstError={{ behavior: 'instant', block: 'end', focus: true
}}
      style={{ paddingBlock: 32 }}
      labelCol={{ span: 6 }}
      wrapperCol={{ span: 14 }}
    >
      <Form.Item wrapperCol={{ offset: 6 }}>
        <Button onClick={() => form.scrollToField('bio')}>Scroll to
Bio</Button>
      </Form.Item>

      <Form.Item name="username" label="UserName" rules={[{ required: true
}]}>
        <Input />
      </Form.Item>

      <Form.Item label="Occupation" name="occupation">
        <Select
          options={[
            { label: 'Designer', value: 'designer' },
            { label: 'Developer', value: 'developer' },
            { label: 'Product Manager', value: 'product-manager' },
          ]}
        />
      </Form.Item>

      <Form.Item name="motto" label="Motto">
        <Input.TextArea rows={4} />
      </Form.Item>

      <Form.Item name="bio" label="Bio" rules={[{ required: true }]}>
        <Input.TextArea rows={6} />
      </Form.Item>

      <Form.Item wrapperCol={{ offset: 6 }}>
        <Flex gap="small">
          <Button type="primary" htmlType="submit">
            Submit
          </Button>
```

```
          <Button danger onClick={() => form.resetFields()}>
            Reset
          </Button>
        </Flex>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Other Form Controls

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

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,
      'color-picker': null,
    }}
    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={{ marginInlineStart: 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',
      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="longggggggggggggggggggggggggggggggggg"
    >
      <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
      name="color-picker"
      label="ColorPicker"
      rules={[{ required: true, message: 'color is required!' }]}
    >
      <ColorPicker />
    </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;
```

Disabled Input Debug

Debug

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

const App: React.FC = () => (
  <Form style={{ maxWidth: 600 }}>
    <Form.Item label="Normal0">
      <Input placeholder="unavailable choice" value="Buggy!" />
    </Form.Item>
    <Form.Item label="Fail0" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" value="Buggy!" />
    </Form.Item>
    <Form.Item label="FailDisabled0" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" disabled value="Buggy!" />
    </Form.Item>
    <Form.Item label="Normal1">
      <Input placeholder="unavailable choice" value="Buggy!" />
    </Form.Item>
    <Form.Item label="Fail1" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" value="Buggy!" />
    </Form.Item>
    <Form.Item label="FailDisabled1" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" disabled value="Buggy!" />
    </Form.Item>
    <Form.Item label="Normal2">
      <Input placeholder="unavailable choice" addonBefore="Buggy!" />
    </Form.Item>
    <Form.Item label="Fail2" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" addonBefore="Buggy!" />
    </Form.Item>
    <Form.Item label="FailDisabled2" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" disabled addonBefore="Buggy!"
/>
    </Form.Item>
    <Form.Item label="Normal3">
      <Input placeholder="unavailable choice" prefix="人民币" value="50" />
    </Form.Item>
    <Form.Item label="Fail3" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" prefix="人民币" value="50" />
    </Form.Item>
    <Form.Item label="FailDisabled3" validateStatus="error" help="Buggy!">
      <Input placeholder="unavailable choice" disabled prefix="人民币"
value="50" />
    </Form.Item>
  </Form>
);
```

```
export default App;
```

label ellipsis

Debug

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

const App: React.FC = () => (
  <Form
    name="label-ellipsis"
    labelCol={{ span: 8 }}
    wrapperCol={{ span: 16 }}
    style={{ maxWidth: 600 }}
  >
    <Form.Item
      label={
        <Typography.Text ellipsis>
          longtextlongtextlongtextlongtextlongtextlongtextlongtext
        </Typography.Text>
      }
      name="username"
    >
      <Input />
    </Form.Item>

    <Form.Item
      label={
        <Typography.Text ellipsis>
          longtext longtext longtext longtext longtext longtext longtext
        </Typography.Text>
      }
      name="password"
    >
      <Input.Password />
    </Form.Item>
  </Form>
);

export default App;
```

Test col 24 usage

Debug

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

const sharedItem = (
  <Form.Item
    label={
      <a
        href="https://github.com/ant-design/ant-design/issues/36459"
        target="_blank"
        rel="noreferrer"
      >
        #36459
      </a>
    }
    initialValue={['bamboo']}
    name="select"
    style={{ boxShadow: '0 0 3px red' }}
  >
    <Select
      style={{ width: '70%' }}
      mode="multiple"
      options={[
        { label: 'Bamboo', value: 'bamboo' },
        { label: 'Little', value: 'little' },
        { label: 'Light', value: 'light' },
      ]}
    />
  </Form.Item>
);

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

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

  return (
    <>
      <Form
        name="col-24-debug"
        labelCol={{ span: 24 }}
        wrapperCol={{ span: 24 }}
        initialValues={{ remember: true }}
```

```
        onFinish={onFinish}
        onFinishFailed={onFinishFailed}
        style={{ maxWidth: 600 }}
        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>

        {sharedItem}

        <Form.Item>
          <Button type="primary" htmlType="submit">
            Submit
          </Button>
        </Form.Item>
      </Form>
      <Form
        name="responsive"
        labelCol={{ sm: 24, xl: 24 }}
        wrapperCol={{ sm: 24, xl: 24 }}
        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>
          <Button type="primary" htmlType="submit">
            Submit
          </Button>
        </Form.Item>
      </Form>

      <Divider />

      <Form layout="vertical">
        {sharedItem}

        <Form.Item label="col12" name="col12" labelCol={{ span: 12 }}
wrapperCol={{ span: 12 }}>
          <Input />
        </Form.Item>
      </Form>
    </>
  );
};

export default App;
```

Ref item

Debug

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

const App: React.FC = () => {
  const [form] = Form.useForm();
  const ref = React.useRef<InputRef>(null);
```

```
  return (
    <Form form={form} initialValues={{ list: ['light'] }} style={{
maxWidth: 600 }}>
      <Form.Item name="test" label="test">
        <Input ref={ref} />
      </Form.Item>

      <Form.List name="list">
        {(fields) =>
          fields.map((field) => (
            <Form.Item {...field} key={field.key}>
              <Input ref={ref} />
            </Form.Item>
          ))
        }
      </Form.List>

      <Button
        htmlType="button"
        onClick={() => {
          form.getFieldInstance('test').focus();
        }}
      >
        Focus Form.Item
      </Button>
      <Button
        onClick={() => {
          form.getFieldInstance(['list', 0]).focus();
        }}
      >
        Focus Form.List
      </Button>
    </Form>
  );
};

export default App;
```

Custom feedback icons

Debug

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

```tsx
import { createStyles, css } from 'antd-style';
import uniqueId from 'lodash/uniqueId';

const useStyle = createStyles(() => ({
  'custom-feedback-icons': css`
    .ant-form-item-feedback-icon {
      pointer-events: all;
    }
  `,
}));

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

  return (
    <Form
      name="custom-feedback-icons"
      form={form}
      style={{ maxWidth: 600 }}
      feedbackIcons={({ errors }) => ({
        error: (
          <Tooltip
            key="tooltipKey"
            title={errors?.map((error) => <div key={uniqueId()}>{error}
</div>)}
            color="red"
          >
            <CloseSquareFilled />
          </Tooltip>
        ),
      })}
    >
      <Form.Item
        name="custom-feedback-test-item"
        label="Test"
        className={styles['custom-feedback-icons']}
        rules={[{ required: true, type: 'email' }, { min: 10 }]}
        help=""
        hasFeedback
      >
        <Input />
      </Form.Item>
      <Form.Item
        name="custom-feedback-test-item2"
        label="Test"
```

```
        className={styles['custom-feedback-icons']}
        rules={[{ required: true, type: 'email' }, { min: 10 }]}
        help=""
        hasFeedback={{
          icons: ({ errors }) => ({
            error: (
              <Tooltip
                key="tooltipKey"
                title={errors?.map((error) => <div key={uniqueId()}>{error}
</div>)}
                color="pink"
              >
                <AlertFilled />
              </Tooltip>
            ),
            success: false,
          }),
        }}
      >
        <Input />
      </Form.Item>
      <Form.Item>
        <Button htmlType="submit">Submit</Button>
      </Form.Item>
    </Form>
  );
};

export default App;
```

Component Token

Debug

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

const App: React.FC = () => (
  <ConfigProvider
    theme={{
      components: {
        Form: {
          labelRequiredMarkColor: 'pink',
          labelColor: 'green',
          labelFontSize: 16,
          labelHeight: 34,
```

```
          labelColonMarginInlineStart: 4,
          labelColonMarginInlineEnd: 12,
          itemMarginBottom: 18,
          inlineItemMarginBottom: 18,
        },
      },
    }}
  >
    <Form
      name="component-token"
      labelCol={{ span: 8 }}
      wrapperCol={{ span: 16 }}
      style={{ maxWidth: 600 }}
      initialValues={{ remember: true }}
      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>
  </ConfigProvider>
);

export default App;
```

API

Common props ref：Common props

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	4
component	Set the Form rendering element. Do not create a DOM node for `false`	ComponentType | false	form	
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	-	
feedbackIcons	Can be passed custom icons while `Form.Item` element has `hasFeedback`	FeedbackIcons	-	5
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	4
labelCol	Label layout, like `<Col>` component. Set `span` `offset` value like `{span: 3, offset:`	object	-	

	12} or sm: {span: 3, offset: 12}			
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. You can get the preserve field value by getFieldsValue(true)	boolean	true	4
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 | ((label: ReactNode, info: { required: boolean }) => ReactNode)	true	r 5
scrollToFirstError	Auto scroll to first failed field when submit	boolean | Options | { focus: boolean }	false	f
size	Set field component size (antd components only)	small | middle | large	-	
validateMessages	Validation prompt template, description see below	ValidateMessages	-	
validateTrigger	Config field validate trigger	string | string[]	onChange	4
variant	Variant of components inside form	outlined | borderless | filled | underlined	outlined	5 u 5
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)	-	
clearOnDestroy	Clear form values when the form is uninstalled	boolean	false	

> *It accepts all props which native forms support but* `onSubmit` .

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 (It's not recommended to generate dynamic function prop by `getValueProps`. Please pass it to child component directly)	(value: any) => Record<string, any>	-	4.2.0
hasFeedback	Used with `validateStatus`, this option specifies the validation status icon. Recommended to be used only with `Input`. Also, It can get feedback icons via icons prop.	boolean | { icons: FeedbackIcons }	false	icons: 5.9.0
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	string	-	4.2.0

	`initialValues` get higher priority when conflict			
label	Label text. When there is no need for a label but it needs to be aligned with a colon, it can be set to null	ReactNode	-	null: 5.22.0
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: 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	object	-	
messageVariables	The default validate field info, description see below	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. Will inherit parent Form.Item `validateStatus` if self `validateStatus` not configured	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	
validateDebounce	Delay milliseconds to start validation	number	-	5.9.0
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 provided, it will be generated by validation rule. options: `success` `warning` `error` `validating`	string	-	
validateTrigger	When to validate the value of children node	string | string[]	onChange	
valuePropName	Props of children node, for example, the prop of Switch or Checkbox is `checked`. This prop is an encapsulation of	string	`value`	

	`getValueProps`, which will be invalid after customizing `getValueProps`			
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	-	
layout	Form item layout	`horizontal` | `vertical`	-	5.18.0

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.

FeedbackIcons

`({ status: ValidateStatus, errors: ReactNode, warnings: ReactNode }) => Record<ValidateStatus, ReactNode>`

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. It should be noted that the child component should be returned in a function, otherwise `shouldUpdate` won't behave correctly:

related issue: #34500

```
<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.additional}>
  {() => {
    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>
```

Since `5.20.2`, when you don't want to convert `${}`, you can use `\\${}` to skip:

```
{ required: true, message: '${label} is convert, \\${label} is not convert'
}

// good is convert, ${label} is not convert
```

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. List is also a field, so it will return all the values by `getFieldsValue`. You can change this logic by config	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	GetFieldsValue	
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 | { focus: boolean }) => void	focus: 5.24.0
setFields	Set fields status	(fields: FieldData[]) => void	
setFieldValue	Set fields value(Will directly pass to form store and **reset validation message**. 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 and **reset validation message**. 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. Use `recursive` to validate all the field in the path	(nameList?: NamePath[], config?: ValidateConfig) => Promise	

validateFields

```
export interface ValidateConfig {
  // New in 5.5.0. Only validate content and not show error message on UI.
  validateOnly?: boolean;
  // New in 5.9.0. Recursively validate the provided `nameList` and its
sub-paths.
  recursive?: boolean;
  // New in 5.11.0. Validate dirty fields (touched + validated).
  // It's useful to validate fields only when they are touched or
validated.
  dirty?: boolean;
}
```

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 | (selector: (values: Store) => any), formInstance?: FormInstance | WatchOptions): Value`

`5.12.0` add `selector`

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

```
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`:

```
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:

```
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)[]`

GetFieldsValue

`getFieldsValue` provides overloaded methods:

getFieldsValue(nameList?: true | NamePath[], filterFunc?: FilterFunc)

When `nameList` is empty, return all registered fields, including values of List (even if List has no Item children).

When `nameList` is `true`, return all values in store, including unregistered fields. For example, if you set the value of an unregistered Item through `setFieldsValue`, you can also get all values through `true`.

When `nameList` is an array, return the value of the specified path. Note that `nameList` is a nested array. For example, you need the value of a certain path as follows:

```
// Single path
form.getFieldsValue([['user', 'age']]);

// multiple path
form.getFieldsValue([
  ['user', 'age'],
  ['preset', 'account'],
]);
```

getFieldsValue({ strict?: boolean, filter?: FilterFunc })

New in `5.8.0`. Accept configuration parameters. When `strict` is `true`, only the value of Item will be matched. For example, in `{ list: [{ bamboo: 1, little: 2 }] }`, if List is only bound to the `bamboo` field, then `getFieldsValue({ strict: true })` will only get `{ list: [{ bamboo: 1 }] }`.

FilterFunc

To filter certain field values, `meta` will provide information related to the fields. For example, it can be used to retrieve values that have only been modified by the user, and so on.

```
type FilterFunc = (meta: { touched: boolean; validating: boolean }) =>
boolean;
```

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	
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 | ReactElement	
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

FAQ

Why can't Switch、Checkbox bind data?

Form.Item default bind value to `value` prop, but Switch or Checkbox value prop is `checked`. You can use `valuePropName` to change bind value prop.

```
<Form.Item name="fieldA" valuePropName="checked">
  <Switch />
</Form.Item>
```

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 can't `getFieldsValue` get value at first render?

`getFieldsValue` returns collected field data by default, but the Form.Item node is not ready at the first render. You can get all field data by `getFieldsValue(true)`.

Why some component not response with `setFieldsValue` to `undefined`?

`value` change from certain one to `undefined` in React means from controlled mode to uncontrolled mode. Thus it will not change display value but modified FormStore in fact. You can HOC to handle this:

```
const MyInput = ({
  // Force use controlled mode
  value = '',
  ...rest
}) => <input value={value} {...rest} />;

<Form.Item name="my">
  <MyInput />
</Form.Item>;
```

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`:

```
<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?

1. use custom form control

See similar issues: #28370 #27994

Starting from version `5.17.0`, the sliding operation will prioritize using the ref element forwarded by the form control elements. Therefore, when considering custom components to support verification scrolling, please consider forwarding it to the form control elements first.

`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.

2. multiple forms on same page

If there are multiple forms on the page, and there are duplicate same `name` form item, the form scroll probably may find the form item with the same name in another form. You need to set a different `name` for the `Form` component to distinguish it.

Continue, why not use `ref` to bind element?

Form can not get real DOM node when customize component not support `ref`. It will get warning in React Strict Mode if wrap with Class Component and call `findDOMNode`. So we use `id` to locate element.

`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.

Why Form.Item not update value when children is nest?

Form.Item will inject `value` and `onChange` to children when render. Once your field component is wrapped, props will not pass to the correct node. Follow code will not work as expect:

```
<Form.Item name="input">
  <div>
    <h3>I am a wrapped Input</h3>
    <Input />
  </div>
</Form.Item>
```

You can use HOC to solve this problem, don't forget passing props to form control component:

```
const MyInput = (props) => (
  <div>
    <h3>I am a wrapped Input</h3>
    <Input {...props} />
  </div>
);

<Form.Item name="input">
  <MyInput />
</Form.Item>;
```

Why does clicking the label in the form change the component state?

> *Related issue: #47031, #43175, #52152*

Form label use HTML label elements to wrap form controls, which focuses the corresponding control when clicked. This is the native behavior of label elements, designed to improve accessibility and user experience. This standard interaction pattern makes it easier for users to interact with form controls. If you need to disable this behavior, you can use `htmlFor={null}`, though it's generally not recommended.

```
- <Form.Item name="switch" label="Switch">
+ <Form.Item name="switch" label="Switch" htmlFor={null}>
```

```
    <Switch />
  </Form.Item>
```