Remove Form.create

Form of v4 does not need to create context by calling `Form.create()`. Form now has it's own data scope and you don't need `getFieldDecorator` anymore. Just use Form.Item directly:

```
// antd v3
const Demo = ({ form: { getFieldDecorator } }) => (
  <Form>
    <Form.Item>
      {getFieldDecorator('username', {
        rules: [{ required: true }],
      })(<Input />)}
    </Form.Item>
  </Form>
);

const WrappedDemo = Form.create()(Demo);
```

To:

```
// antd v4
const Demo = () => (
  <Form>
    <Form.Item name="username" rules={[{ required: true }]}>
      <Input />
    </Form.Item>
  </Form>
);
```

Since `Form.create()` is removed, methods like `onFieldsChange` have moved to `Form` and form state is controlled by a `fields` prop. ref [example](#).

Form control

If you want to control form, you can use `Form.useForm()` to create Form instance for operation:

```
// antd v3
const Demo = ({ form: { setFieldsValue } }) => {
  React.useEffect(() => {
    setFieldsValue({
      username: 'Bamboo',
    });
  }, []);

  return (
    <Form>
      <Form.Item>
        {getFieldDecorator('username', {
          rules: [{ required: true }],
        })(<Input />)}
      </Form.Item>
```

```
    </Form>
  );
};

const WrappedDemo = Form.create()(Demo);
```

To:

```
// antd v4
const Demo = () => {
  const [form] = Form.useForm();

  React.useEffect(() => {
    form.setFieldsValue({
      username: 'Bamboo',
    });
  }, []);

  return (
    <Form form={form}>
      <Form.Item name="username" rules={[{ required: true }]}>
        <Input />
      </Form.Item>
    </Form>
  );
};
```

For class component, you can use `ref` to access instance:

```
// antd v4
class Demo extends React.Component {
  formRef = React.createRef();

  componentDidMount() {
    this.formRef.current.setFieldsValue({
      username: 'Bamboo',
    });
  }

  render() {
    return (
      <Form ref={this.formRef}>
        <Form.Item name="username" rules={[{ required: true }]}>
          <Input />
        </Form.Item>
      </Form>
    );
  }
}
```

If you don't want to use the Item style, you can use `noStyle` prop to remove it:

```
// antd v3
const Demo = ({ form: { getFieldDecorator } }) => {
  return <Form>{getFieldDecorator('username')(<Input />)}</Form>;
};

const WrappedDemo = Form.create()(Demo);
```

To:

```
// antd v4
const Demo = () => {
  return (
    <Form>
      <Form.Item name="username" noStyle>
        <Input />
      </Form.Item>
    </Form>
  );
};
```

Linkage with field

New Form uses incremental update which only updates related field. So if there is some linkage between fields or updates with the whole form, you can use `dependencies` or `shouldUpdate` to handle that.

replace onSubmit with onFinish

You need to listen to `onSubmit` and call `validateFields` to handle validation in old Form. New Form provides `onFinish` which will only trigger when validation has passed:

```
// antd v3
const Demo = ({ form: { getFieldDecorator, validateFields } }) => {
  const onSubmit = e => {
    e.preventDefault();
    validateFields((err, values) => {
      if (!err) {
        console.log('Received values of form: ', values);
      }
    });
  };

  return (
    <Form onSubmit={onSubmit}>
      <Form.Item>
        {getFieldDecorator('username', {
          rules: [{ required: true }],
        })(<Input />)}
      </Form.Item>
```

```
    </Form>
  );
};

const WrappedDemo = Form.create()(Demo);
```

To:

```
// antd v4
const Demo = () => {
  const onFinish = values => {
    console.log('Received values of form: ', values);
  };

  return (
    <Form onFinish={onFinish}>
      <Form.Item name="username" rules={[{ required: true }]}>
        <Input />
      </Form.Item>
    </Form>
  );
};
```

Replace validateFieldsAndScroll with scrollToField

New version recommend use `onFinish` for submit after validation. Thus `validateFieldsAndScroll` is changed to more flexible method `scrollToField`:

```
// antd v3
onSubmit = () => {
  form.validateFieldsAndScroll((error, values) => {
    // Your logic
  });
};
```

To:

```
// antd v4
onFinishFailed = ({ errorFields }) => {
  form.scrollToField(errorFields[0].name);
};
```

Initialization

Besides, we move `initialValue` into Form to avoid field with same name both using `initialValue` to cause conflict:

```
// antd v3
const Demo = ({ form: { getFieldDecorator } }) => (
```

```
  <Form>
    <Form.Item>
      {getFieldDecorator('username', {
        rules: [{ required: true }],
        initialValue: 'Bamboo',
      })(<Input />)}
    </Form.Item>
  </Form>
);

const WrappedDemo = Form.create()(Demo);
```

To:

```
// antd v4
const Demo = () => (
  <Form initialValues={{ username: 'Bamboo' }}>
    <Form.Item name="username" rules={[{ required: true }]}>
      <Input />
    </Form.Item>
  </Form>
);
```

In v3, modifying the `initialValue` of un-operated field will update the field value synchronously, which is a bug. However, since it has been used as a feature for a long time, we have not fixed it. In v4, the bug has been fixed. `initialValues` only takes effect when initializing and resetting the form.

Nested field paths using arrays

In the past versions we used `.` to represent nested paths (such as `user.name` to represent `{ user: { name: '' } }`). However, in some backend systems, `.` is also included in the variable name. This causes the user to need additional code to convert, so in the new version, nested paths are represented by arrays to avoid unexpected behavior (eg `['user', 'name']`).

Therefore, paths returned by methods such as `getFieldsError` are always in an array form for the user to handle:

```
form.getFieldsError();

/*
[
  { name: ['user', 'name'], errors: [] },
  { name: ['user', 'age'], errors: ['Some error message'] },
]
*/
```

Nested field definition has changed from:

```
// antd v3
<Form.Item label="Firstname">{getFieldDecorator('user.0.firstname', {})(<Input />)}
```

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

To

```
// antd v4
<Form.Item name={['user', 0, 'firstname']} label="Firstname">
  <Input />
</Form.Item>
```

Similarly using `setFieldsValue` has changed from:

```
// antd v3
this.formRef.current.setFieldsValue({
  'user.0.firstname': 'John',
});
```

To

```
// antd v4
this.formRef.current.setFieldsValue({
  user: [
    {
      firstname: 'John',
    },
  ],
});
```

Remove callback in validateFields

`validateFields` will return a Promise, so you can handle the error with `async/await` or `then/catch`. It is no longer necessary to determine if `errors` is empty:

```
// antd v3
validateFields((err, value) => {
  if (!err) {
    // Do something with value
  }
});
```

To

```
// antd v4
validateFields().then(values => {
  // Do something with value
});
```