Antd UI 组件：

1. Form表单

1.1：栅栏是24格。

1.2：两种排列方式：

1.2.1：水平排列，可以实现 label 标签和表单控件的水平排列;

<Form horizontal onSubmit={this.handleSubmit}>

1.2.2：行内排列，使其表现为 inline-block 级别的控件。

<Form inline onSubmit={this.handleSubmit}>

1.3：使用Form.create 来获取和更新表单提交的数值。

let Demo = React.createClass({});

Demo = Form.create()(Demo);

1.4：表单控件定义了三种校验状态，为 <FormItem> 定义 validateStatus 属性即可。

validateStatus: ['success', 'warning', 'error', 'validating']

1.5：另外为输入框添加反馈图标，设置 <FormItem> 的 hasFeedback 属性值 为 true 即可。注意: 反馈图标只对 <Input /> 有效。

1.6：如果想要input和你想要的图标组合的话，需要用到Input.Group

1.7：<formItem>组件常用的属性label、labelCol、wrapperCol。

Label：标签名；

labelCol：标签名的样式布局；

wrapperCol：输入控件的样式布局。



创建reactApp的几种方式：

1. create-react-app filename 适用于npm6及以下。
2. npm init react-app filename 适用于npm6以上。
3. npx create-react-app filename 适用于npm5.2以上。

利用脚手架创建vue项目的几种方式：

1. vue create filename 适用于vue-cli3.0以上。
2. vue init webpack filename 适用于 vue-cli2。

React之reflux：

1. 创建action和store：

1.1：reflux.createAction()： 创建一个action；返回值是一个函数，调用这个函数就

会触发相应的事件，在store中监听这个函数，并作相 应的处理。

reflux.createActions([])：创建多个action；返回值是一个对象，包含多个函数。

1.2：reflux.createStore({}); 创建store实例，返回一个对象。

1.3：reflux.ActionMethods.pck = function(){}; 创建action方法的公用方法。

单个action使用公用方法：action.pck();

多个action使用公用方法：action.add.pck();

1.4：reflux.StoreMethods.pck = function(){}; 创建Store的公用方法。

1.5：拓展store的公用方法：

1.5.1：reflux.StoreMethods.pck = function(){}; //pck是定义的方法名。

1.5.2：let mixin = { pck: function(){} };

let store = reflux.createStore({ mixins:[mixin], ...});

1. Store监听action：

let actions = reflux.createActions([ ‘add’, ‘delete’, ‘check’]);

let actions1 = reflux.createAction();

let store = reflux.createStore({ init: function(){}, ...});

2.1：this.listenTo( actions.add, “add”); 第一个参数是actions对象的方法。第二个参

数是store对象监听action的方法调用的方法名。如果是actions1的话，那么 写法就是this.listenTo( actions1, “actions1”); first和second参数名不需要一致。

2.2：this.listenToMany(actions); 参数actions必须是json对象。

处理方法的写法只需让action的标识首字母大写并加上on就可以了。

例如：add→onAdd。

2.3：listenables: [actions]; 属性值actions必须是json对象。

处理方法的写法只需让action的标识首字母大写并加上on就可以了。

例如：add→onAdd。

注意：2.1和2.2都是写在init函数内部，2.3方法是属性写法，listenables是配置 属性。并且2.2和2.3的参数或者属性值必须是createActions创建的action。

1. 异步action ：let asyncAction = Reflux.createAction({asyncResult: true});



1. Action hooks：Reflux为每个action都提供了两个hook方法。

4.1：preEmit(params) ，action emit之前调用，参数是action传递过来的，返回值 会传递给shouldEmit()；

4.2：shouldEmit(params) ，action emit之前调用，参数默认是action传递，如果 preEmit有返回值，则是preEmit返回值；shouldEmit()的返回值决定是否 触发action事件。

4.3：注意，如果preEmit有返回值，则该返回值会作为action事件的参数；否则就 是action的初始参数；不论shouldEmit的返回值是什么，它只是决定是否触 发action事件。

1. 同组件结合：

5.1：store.listen(func); func：被监听的函数。返回值是解除store监听的一个函数。

5.1.1：当组件的生命周期结束时需要解除对Store的监听。

5.1.2： 当Store调用trigger时，才会执行func函数，所以每次Store 更新时，需要手动调用trigger函数。

5.1.3：并且store.trigger(params); 的参数将是func的参数（如果有）。

5.2：mixins: [Reflux.ListenerMixin] + store.listen(func);

5.2.1：适用于作为React.createClass({})的配置属性。

例如：let test = React.createClass({ mixins: [Reflux.ListenerMixin] });

5.2.2：接下来的写法和5.1方法一样，不过没有了5.1.1的限制。

5.3：mixins: [Reflux.listenTo(store, “func”)]

5.3.1：适用于作为React.createClass({})的配置属性。

例如：let test = React.createClass({mixins: [Reflux.listenTo(store, “func”)] });

5.3.2：它比5.2更优化的是，没有了store.listen(func); 但是也必须要手动触发 store.trigger();

5.4：minxins: [Reflux.connect(store, ‘list’)]

5.4.1：它相对应5.3来说，优化了一点，也就是不需要函数来更新state里面 的list，但是还是必须手动触发store.trigger()。

5.4.2：注意，这里的‘list’参数是state里面的属性名（被监听的）。

5.5：mixins: [Reflux.connectFilter(store, ‘list’, function(list){})];

5.5.1：相对于5.4来说，只是增加了一个数据过滤器。

5.5.2：第三个参数就是过滤函数。参数名不必和第二个参数名相同。

5.5.3：还是必须要手动触发 store.trigger(); 。

总结：这5种结合的方法，都必须手动触发store.trigger(); ；

5.4方法与5.5方法的第二个参数名必须和state里面的属性（被监听）名一 致（字符串）。

React-mixin:



createClass中常规属性的许多东西现在都是类的静态属性。要使getDefaultProps，

propTypes和getInitialState正常工作，您需要将react-mixin应用于比原型更高的级别：

类本身。

Localstorage:

1. 写入：