Reactjs is not a framework which contains all parts of mvc like angularjs. It’s more like a component maker, like angularjs’s “directive”. although it can build a frontend structure with other libraries like “Backbone.Router”. but the main purpose for reactjs is not to build a framework. It’s just to improve the “view” part.

<https://github.com/reactjs/react-rails>

this link shows how to set up react with ruby on rails. And for the record the JSX content only works in \*\*\*\*\*.js.jsx file.

Plus It works perfect with some css component design solution like bootstrap.

1. JSX

JSX is not necessary for reactjs, but it can accelerate the speed of making react component, and it looks like html, so it’s more readable for some people familiar with html.

1. Life cycle for a reactjs component

A component is like

var Node = React.createClass({

propTypes: {

id:React.PropTypes.string,

title: React.PropTypes.string,

type: React.PropTypes.string

},

render: function() {

return (

<form className="form-inline">

<div className="form-group">

<icon id={this.props.id} className="caret"></icon>

</div>

<div className="form-group">

<div className="control-label col-md-4 col-xs-4 col-sm-4">{this.props.title}</div>

<div className="control-label col-md-8 col-xs-8 col-sm-8">{this.props.type}</div>

</div>

<div className="form-group"><input id={this.props.id} type="checkbox"/></div>

</form>

);

}

});

and you use It like this:

ReactDOM.render(

React.createFactory(Node)({title:'title',type:'type',id:'asfdasfasdfasdf3'}),

document.getElementById('example')

);

The component has it’s own life cycle. Which correspond to a function defined by reactjs.

When initialize:

The following function will be called as the following sequence:

* getDefaultProps
* getInitialState
* componentWillMount
* render
* componentDidMount

When the states changes:

* componentWillReceiveProps
* shouldComponentUpdate
* componnetWillUpdate
* render
* componnetDidUpdate

When destroy:

* componnetWillUnmount

When the component were created the getInitialState() was called and initialize the component.

Render() function:

You will create a virtual dom here,It has some rules:

* Can only access data through this.props and this.state
* Return null or fals or any react component
* Only one top element contains all children element
* Can’t change component state or dom output

componnetDidMout function:

when the dom has been rendered, you can get it in componnetDidMout() function with function this.getDOMNode(). This is where you access the original dom life cycle of the browser.

Like this:

….

componnetDidMount:function(){

this.getDOMNode().className = “”;

}

…

1. Data flow

In react the data just passed from parent component to children component.

They just get props from parent component and render them. If some props changed react will check all It’s children and change them all.

Props is for the parameter, they can be accessed from outside of the component.

States can only be accessed within the component.

1. Event handler

The event handler for reactjs is just the same with original javascript event handler.

…

render:function(){

return(

<div onDragOver={this.handleDragOver}></div>

….)

…

handleDragOver:function(){

…

}

1. React composability

Var componentA = React.createClass({

Render:function(){

Return (

<div>a</div>);

}

});

var componentB = React.createClass({

render:function(){

return(

<div>b<componentA></componnetA></div>

);

}

});

now we can only show children component. We can’t reflect the change from children component.

Now we can use it on html page like this

<componentB></componentB>

It should display something like “b /n a”.

Now we modify the component definition and connect the parent and child component:

Var componentA = React.createClass({

propTypes:{

onChanged:React.PropTypes.func.isRequired

},

handleChanged:function(){

this.props.onChanged();

},

Render:function(){

return (

<div onClick={this.handleChanged}>a</div>);

}

});

var ComponentB = React.createClass({

handleChanged:function(){

alert("changed");

},

render: function() {

return <div>

b {React.createElement(ComponentA, {onChanged:this.handleChanged})}

</div>;

}

});

We use callback function connect parent and children component.

1. mixin

<http://facebook.github.io/react/docs/reusable-components.html>

mixin is a way to define a public method that we share in multiple components.

var SetIntervalMixin = {

componentWillMount: function() {

this.intervals = [];

},

setInterval: function() {

this.intervals.push(setInterval.apply(null, arguments));

},

componentWillUnmount: function() {

this.intervals.forEach(clearInterval);

}

};

var Timer = React.createClass({

mixins: [SetIntervalMixin], // Use the mixin

getInitialState: function() {

return {seconds: 0};

},

componentDidMount: function() {

this.setInterval(this.tick, 1000); // Call a method on the mixin

},

tick: function() {

this.setState({seconds: this.state.seconds + 1});

},

render: function() {

return (

<p>

React has been running for {this.state.seconds} seconds.

</p>

);

}

});

Above is a example of mixins as a timer.

1. DOM operation

Usually the virtual dom operation method provided by react is enough. But there are some instance than we need to operate the real dom.

Before you can reach the real dom you need to define the unique ref value in render method

…

render:function(){

return <canvas ref=”mainCanvas”/>;

}

…

and after this you can reach to the dom in your componnetDidMount mothod or event handler Methods.

If you want to reach the dom in other method it may fail. Because After the real dom component is created the componentDidMount will be triggered.

You can reach the dom component like this:

…

componentDidMount:function(){

var canvasNode = this.refs.mainCanvas.getDOMNode()

}

….

1. library that works with react

* Backbone.Router
* Aviator
* React-router
* Om
* Flux