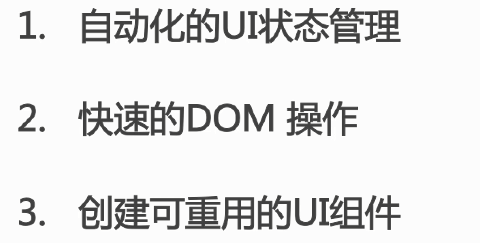
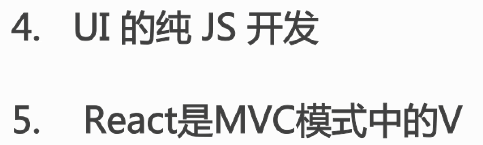
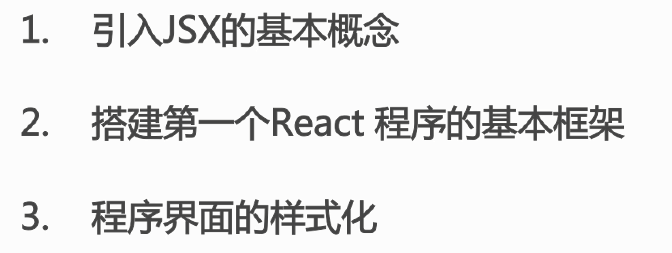
# react



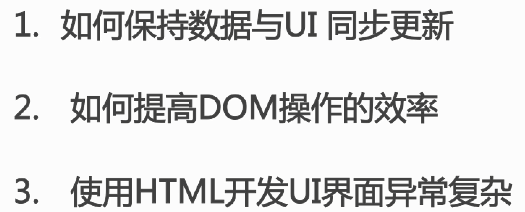




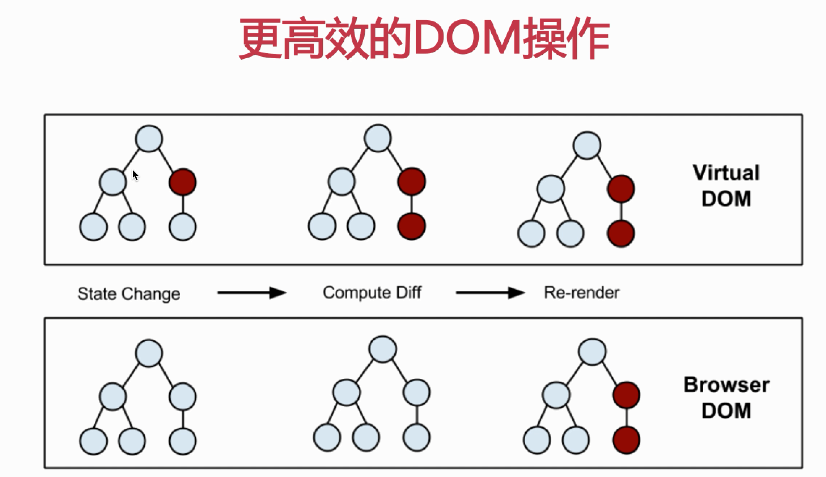




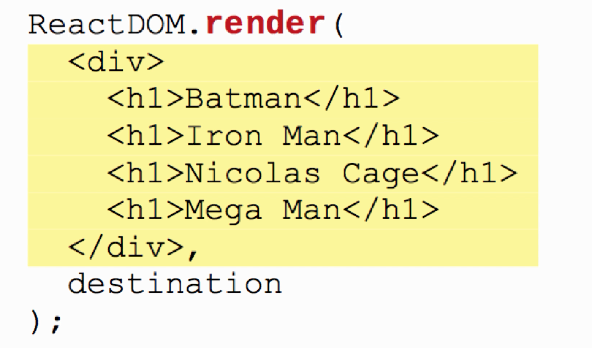
## 单页面app的挑战



## 更高效的Dom操作



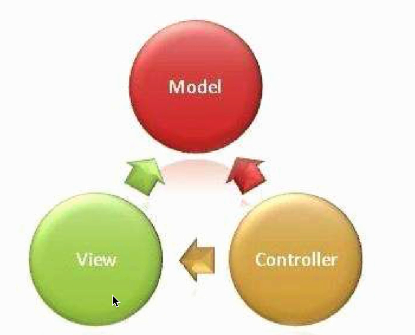
## 依赖JS开发UI界面



## JSX的实际解读



## React是MVC中的V



## 第一个react程序

<!DOCTYPE html>

<html>

<head>

<title>Hello React</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var HelloWorld= React.createClass({

render:function(){

return (

<div>

<p>Hello, {this.props.greetTarget}!</p>

</div>

);

}

});

var destination = document.querySelector("#container");

ReactDOM.render(

<div>

<HelloWorld greetTarget="zhangsan"/>

<HelloWorld greetTarget="lisi"/>

<HelloWorld greetTarget="wangwu"/>

</div>

,

destination

)

</script>

</body>

</html>

## React组件的样式设计

### 不用react的设计样式时，是这样的。

可以看到样式代码在上面，而页面的html结构代码在下面，分离开了。

<style type="text/css">

#container{

padding:50px;

background-color: #FFF;

}

.letter {

padding:10px;

margin:10px;

display: inline-block;

background-color: #ffde00;

color:#333;

font-family: monospace;

font-size: 32px;

text-align: center;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var Letter=React.createClass({

render:function(){

return (

<div className="letter">

{this.props.children}

</div>

);

}

});

var destination=document.querySelector("#container");

ReactDOM.render(

<div>

<Letter>T</Letter>

<Letter>Y</Letter>

<Letter>L</Letter>

<Letter>E</Letter>

<Letter>R</Letter>

</div>,

destination

)

</script>

</body>

### 用react的方式

<!DOCTYPE html>

<html lang="en">

<head>

<title>Styling in React</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<style type="text/css">

#container{

padding:50px;

background-color: #FFF;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var Letter=React.createClass({

render:function(){

var letterStyle={

padding:10,

margin:10,

backgroundColor:"#ffde00",

color:"#333",

display:"inline-block",

fontFamily:"monospace",

fontSize:32,

textAlign:"center"

};

return (

<div style={letterStyle}>

{this.props.children}

</div>

);

}

});

var destination=document.querySelector("#container");

ReactDOM.render(

<div>

<Letter>T</Letter>

<Letter>Y</Letter>

<Letter>L</Letter>

<Letter>E</Letter>

<Letter>R</Letter>

</div>,

destination

)

</script>

</body>

</html>

## 组件的组合性

<!DOCTYPE html>

<html>

<head>

<title>Compositive Card Component</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<style>

#container{

padding:50px;

background-color: #FFF;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var Square= React.createClass({

render:function(){

var squareStyle={

height:150,

backgroundColor:this.props.color

};

return (

<div style={squareStyle}>

</div>

);

}

});

var Label=React.createClass({

render:function(){

var labelStyle={

fontFamily:"sans-serif",

fontWeight:"bold",

padding:13,

margin:0

};

return (

<div style={labelStyle}>

<p>{this.props.color}</p>

</div>

);

}

});

var Card=React.createClass({

render:function(){

var cardStyle={

height:200,

width:150,

backgroundColor:"#FFF",

WebkitFilter:"drop-shadow(0px 5px 5px #666)",

filter:"drop-shadow(0px 0px 5px #666)"

};

return (

<div style={cardStyle}>

<Square color={this.props.color}/>

<Label color={this.props.color}/>

</div>

)

}

});

var destination = document.querySelector("#container");

ReactDOM.render(

<div>

<Card color="#DD6664"/>

</div>

,

destination

)

</script>

</body>

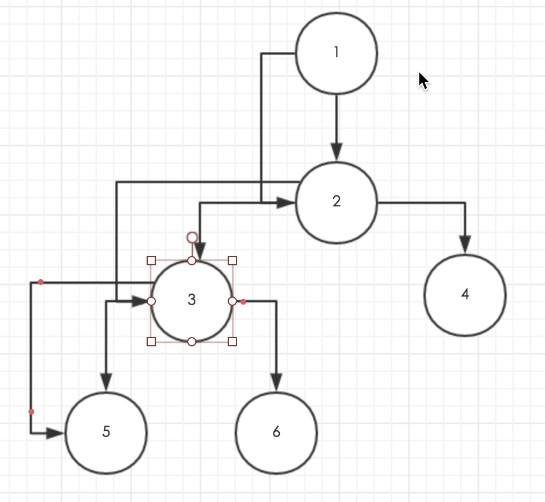
</html>

## 属性传递机制

### 不好的地方

1.只能从上往下一个一个传。所以很不容易扩展，

2.属性名称修改的话，那么从上到下都得改一遍，



<!DOCTYPE html>

<html>

<head>

<title>Compositive Card Component</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var destination = document.querySelector("#container");

var Display=React.createClass({

render:function(){

return (

<div>

<p>{this.props.color}</p>

<p>{this.props.num}</p>

<p>{this.props.size}</p>

</div>

);

}

});

var Label=React.createClass({

render:function(){

return (

<Display color={this.props.color}

num={this.props.num}

size={this.props.size}/>

);

}

});

var Shirt=React.createClass({

render:function(){

return (

<Label color={this.props.color}

num={this.props.num}

size={this.props.size}/>

);

}

});

ReactDOM.render(

<div>

<Shirt color="red" num="123" size="100px"/>

</div>

,

destination

)

</script>

</body>

</html>

### 通过用es6 的属性扩展操作符

<!DOCTYPE html>

<html>

<head>

<title>Compositive Card Component</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var destination = document.querySelector("#container");

var Display=React.createClass({

render:function(){

return (

<div>

<p>{this.props.color}</p>

<p>{this.props.num}</p>

<p>{this.props.size}</p>

</div>

);

}

});

var Label=React.createClass({

render:function(){

return (

<Display {...this.props}/>

);

}

});

var Shirt=React.createClass({

render:function(){

return (

<Label {...this.props}/>

);

}

});

ReactDOM.render(

<div>

<Shirt color="red" num="123" size="100px"/>

</div>

,

destination

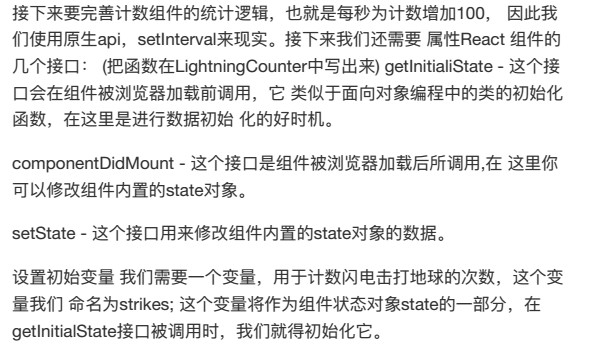
)

</script>

</body>

</html>

## 组件的状态



### GetInitialState

这个接口会在组件被浏览器加载前调用，他类似于面向对象中的类的初始化函数，在这里是进行数据初始化的好时机。

var LightingCounter=React.createClass({

getInitialState:function(){

},

render:function(){

return (

<h1>Hello</h1>

);

}

});

### componentDidMount

这个接口是组件被浏览器加载后，在render函数执行前，所调用，在这里可以修改组件内置的state对象。

## 组件的生命周期

getDefaultProps

ComponentWillMount

ComponentDidMount

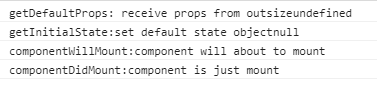
componentWillUnmount

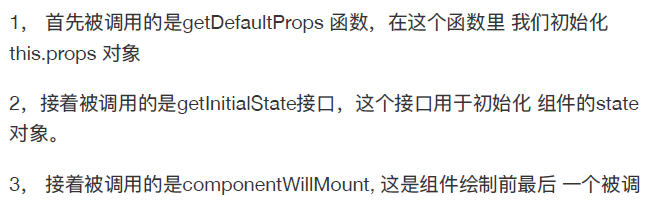
componentWillUpdate

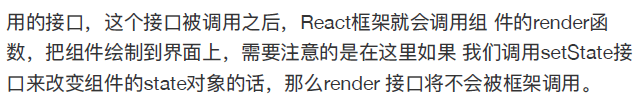
componentDidUpdate

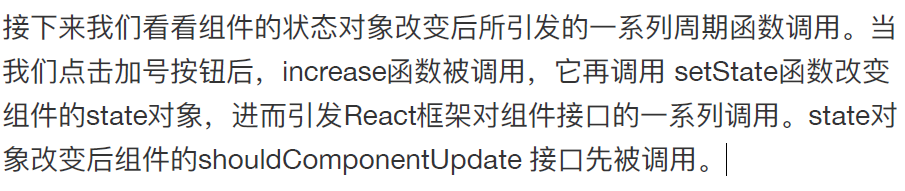
shouldComponentUpdate

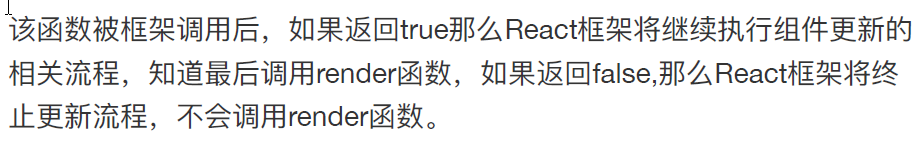
componentWillReceiveProps

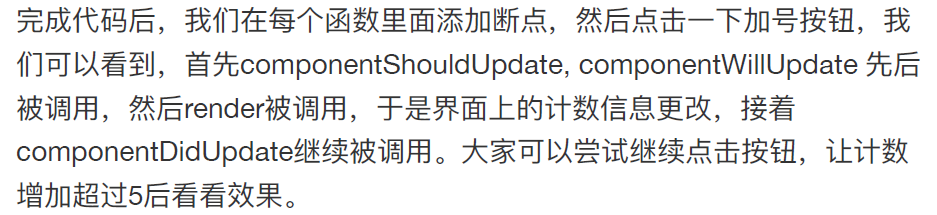




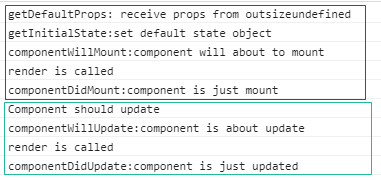








### 页面加载



红框中的代码是页面加载的时候调用的顺序，绿框的代码是点击按钮时调用的顺序。

### 组件的消亡流程

ReactDOM.unmountComponentAtNode(destination);

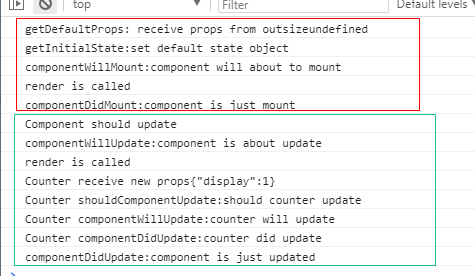
componentWillUnmount:function() {

console.log("componentWillUnmount:component is removed from dom");

return ;

},

### 组件的属性更改



## 响应用户输入

### 数据逻辑与UI的结合

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Compositive Card Component</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<style>

#container{

padding:50px;

background-color: #FFF;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var destination = document.querySelector("#container");

var Circle=React.createClass({

render:function(){

var circleStyle={

padding:10,

margin:20,

display:"inline-block",

backgroundColor:this.props.bgColor,

borderRadius:"50%",

width:100,

height:100

};

return (

<div style={circleStyle}></div>

);

}

});

function showCircle(){

var colors=["#393E41","#E94F37","#1C89BF","#A1D363","#85FFC7","#297373","#FF8552",

"#A40E4C"];

var renderData=[];

for(var i=0;i<colors.length;i++){

var ran=Math.floor(Math.random()\*colors.length);

renderData.push(<Circle key={i+colors[ran]} bgColor={colors[ran]}/>);

}

return renderData;

}

var theCircle=<Circle bgColor="#F9D240"/>;

ReactDOM.render(

<div>

{showCircle()}

</div>

,

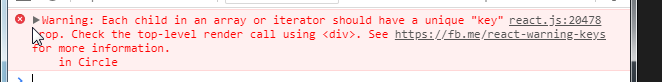
destination

)

</script>

</body>

</html>



### 动态生成组件必须要为每个组件设置唯一key

var ran=Math.floor(Math.random()\*colors.length);

renderData.push(<Circle key={i+colors[ran]} bgColor={colors[ran]}/>);

## react事件处理机制

### 各种事件介绍

在increase函数的实现中，它传⼊入了了⼀一个变量量e,这个变量量就是事 件对象，

该对象包含有很多属性变量量，不不同事件，⾥里里⾯面包含的内 容还不不⼀一样，例例

如，如果事件是⿏鼠标点击后发⽣生的，那么e的对象类型就是MouseEvent, 这

个对象⾥里里就能够包含很多与⿏鼠标相关 的信息，例例如⿏鼠标按下时，被点击的

是左键还是右键，如果事件是由键盘发出的，那么事件的类型就是

KeyboardEvent, 该对象包含的属性能显示很多键盘相关的信息，例例如我们

能得知，键盘上那⼀一按钮被按下了了。







### React把浏览器的各种事件都封装在了SyntheticEvent对象中 比较重要

React把各个版本的浏览器的事件都封装在这个对象中，集中处理，就可以不用关心浏览器的兼容性的问题。

### React事件响应机制

React的事件响应机制如此设计是出⾃自于⼀一些考量量的： ⾸首先考虑的是浏览器器的兼容性，其次要考虑的是事件处理理的效率。

React 把所有事件对象封装到SyntheticEvent这个对象⾥里里，是为了了兼容旧版本的浏览器器。封装了了后，我们在React开发过程中，就不不必要考虑浏览器器的不不同版本，然后编写不不同逻辑的代码，因此React⼤大⼤大简化了了事件处理理的开发逻辑。

其次，当我们设计的⽤用户界⾯面越复杂，需要响应的事件越多，那么⼿手动实现各种事件响应的代码逻辑就会越来越复杂，为了了应对这种情况，React并不不是机械的去处理理各个不不同的事件，在低下它事件上使⽤用的是 ⼀一个总的事件响应函数来捕捉DOM产⽣生的所有事件，先做相应处理理，然后再分发给对应的响应函数，通过这种⽅方式，React能够帮我们优化事件的分发流程，进⽽而提⾼高事件的响应效率。

结论： 本节，我们深⼊入了了解了了React的事件处理理机制，我们先掌握了了基本的事件监听和响应⽅方法，然后，我们⼜又研究了了在⼀一些特殊情况下，如何实现事件响应，通过本节，我想⼤大家对React的事件处理理机制应该会有较为深刻的认识

### 代码

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<style type="text/css">

#container{

padding:50px;

background-color: #FFF;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var destination=document.querySelector("#container");

var Counter=React.createClass({

render:function(){

var textStyle={

fontSize:72,

fontFamily:"sans-serif",

color:"#333",

fontWeight:"bold"

};

return (

<div style={textStyle}>{this.props.count}</div>

);

}

});

var CounterParent=React.createClass({

getInitialState:function(){

return {count:0};

},

increase:function(e){

var currentCount=this.state.count;

if(e.shiftKey==true){

currentCount+=10;

}else{

currentCount+=1;

}

this.setState({

count:currentCount

});

},

render:function(){

var counterStyle={

padding:20,

textAlign:"center"

};

return (

<div>

<Counter count={this.state.count}/>

<button onClick={this.increase}>+</button>

</div>

);

}

});

var divStyle={

padding:40,

width:250,

backgroundColor:"#999",

borderRadius:10,

textAlign:"center"

};

ReactDOM.render(

<div style={divStyle}>

<CounterParent/>

</div>,

destination

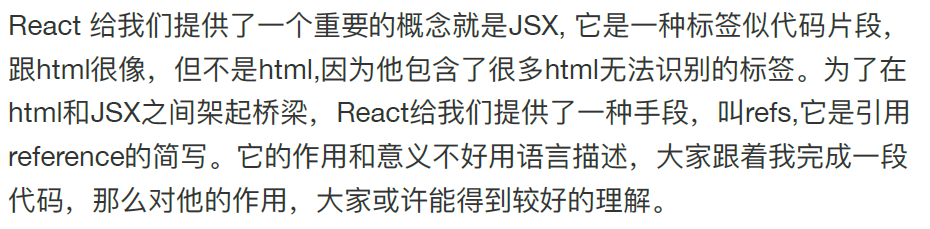
);

</script>

</body>

</html>

## React对DOM模型的操作



### Ref

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<style type="text/css">

#container{

padding:50px;

background-color: #FFF;

}

.colorSquare{

box-shadow: 0 0 25px 0 #333;

width: 242px;

height: 242px;

margin-bottom: 15px;

}

.colorArea input{

padding:10px;

font-size: 16px;

border:2px solid #ccc;

}

.colorArea button{

padding:10px;

font-size: 16px;

margin:10px;

background-color: black;

color: #FFF;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var destination=document.querySelector("#container");

var OperationSection=React.createClass({

getInitialState:function(){

return {

color:'',

bgColor:''

};

},

colorValue:function(e){

this.setState({

color:e.target.value

});

},

setNewColor:function(e){

this.setState({

bgColor:this.state.color

});

this.\_input.value='';

this.\_input.focus();

e.preventDefault();

},

render:function(){

var squareStyle={

backgroundColor:this.state.bgColor

};

var self=this;

return (

<div className="colorArea">

<div className="colorSquare" style={squareStyle}></div>

<form onSubmit={this.setNewColor}>

<input onChange={this.colorValue} placeholder="Please enter a color"

ref={

function(el){

self.\_input=el;

}

} />

<button type="submit">show</button>

</form>

</div>

);

}

});

ReactDOM.render(

<div>

<OperationSection/>

</div>,

destination

);

</script>

</body>

</html>

## 单页应用中的路由

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<script src="ReactRouter.min.js"></script>

<style type="text/css">

body{

background-color: #FFCC00;

padding: 0;

margin: 0;

}

#container{

padding:10px;

}

ul{

background-color: black;

padding: 0;

}

ul li {

display: inline;

}

ul li a{

padding:20px;

display: inline-block;

color: #fff;

font-weight: bold;

text-decoration: none;

}

.active{

background-color: #0099FF;

color: #fff;

font-weight: bold;

}

.content{

margin-top: 20px;

padding: 20px;

background-color: #fff;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var {

Router,

Route,

IndexRoute,

IndexLink,

Link,

hashHistory

}=ReactRouter;

var destination=document.querySelector("#container");

var Home=React.createClass({

render:function(){

return (

<div>

<h2>望天门山</h2>

<p>天门中断楚江开，碧水东流至此回</p>

<p>两岸青山相对出，孤帆一片日边来</p>

</div>

);

}

});

var Dufu=React.createClass({

render:function(){

return (

<div>

<h2>望天门山 杜甫</h2>

<p>杜甫 天门中断楚江开，碧水东流至此回</p>

<p>杜甫 两岸青山相对出，孤帆一片日边来</p>

</div>

);

}

});

var LiShangYin=React.createClass({

render:function(){

return (

<div>

<h2>望天门山 李商隐</h2>

<p>李商隐 天门中断楚江开，碧水东流至此回</p>

<p>李商隐 两岸青山相对出，孤帆一片日边来</p>

</div>

);

}

});

var SinglePage=React.createClass({

getInitialState:function(){

return {

menu:[{

title:'李白',

contentTitle:'望天门山 李白',

content:'李白 天门中断楚江开，碧水东流至此回 \r\n 两岸青山相对出，孤帆一片日边来'

},{

title:'杜甫',

contentTitle:'望天门山 杜甫',

content:'杜甫 天门中断楚江开，碧水东流至此回 \r\n 两岸青山相对出，孤帆一片日边来'

},{

title:'李商隐',

contentTitle:'望天门山 李商隐',

content:'李商隐 天门中断楚江开，碧水东流至此回 \r\n 两岸青山相对出，孤帆一片日边来'

}],

activeMenu:{

contentTitle:'望天门山 李白',

content:'李白 天门中断楚江开，碧水东流至此回 \r\n 两岸青山相对出，孤帆一片日边来'

}

};

},

selectMenu:function(){

this.setState({

activeMenu:this.state.menu[this.refs.\_li.index]

});

},

render:function(){

return (

<div className="mainDiv">

<h1>Simple SPA</h1>

<ul>

<li>

<IndexLink activeClassName="active" to="/">李白</IndexLink>

</li>

<li>

<Link activeClassName="active" to="/df">杜甫</Link>

</li>

<li>

<Link activeClassName="active" to="/lsy">李商隐</Link>

</li>

</ul>

<div className="content">

{this.props.children}

</div>

</div>

);

}

});

ReactDOM.render(

<Router history={hashHistory}>

<Route path="/" component={SinglePage}>

<IndexRoute component={Home}/>

<Route path="df" component={Dufu} />

<Route path="lsy" component={LiShangYin} />

</Route>

</Router>

,

destination

);

</script>

</body>

</html>

## to do list demo



<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<script src="ReactRouter.min.js"></script>

<style type="text/css">

body{

background-color: #26B6B6;

}

.mainDiv{

margin:50px;

padding:50px;

}

.inputDiv input{

height: 30px;

border:2px solid #8820E9;

}

.inputDiv button{

height: 34px;

background-color:blue;

color:#fff;

margin-left:10px;

}

ul{

margin: 0;

padding:0;

}

ul li{

list-style: none;

background-color: rgba(255, 255, 255, 0.5);

padding:10px;

margin:10px 0;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var destination=document.querySelector("#container");

var Li=React.createClass({

render:function(){

var items=this.props.items;

function createTasks(item){

return (

<li key={item.key}>{item.text}</li>

);

}

var listItems=items.map(createTasks);

return (

<ul>

{listItems}

</ul>

);

}

});

var Input=React.createClass({

getInitialState:function(){

return {

items:[]

}

},

add:function(e){

var itemArray=this.state.items;

itemArray.push({

text:this.\_input.value,

key:Date.now()

});

this.setState({

items:itemArray

});

this.\_input.value='';

this.\_input.focus();

e.preventDefault();

},

render:function(){

return (

<div className="mainDiv">

<div className="inputDiv">

<form onSubmit={this.add}>

<input type="text" placeholder="请输入一个任务" ref={

(el)=>this.\_input=el

}/>

<button>添加</button>

</form>

</div>

<Li items={this.state.items}/>

</div>

);

}

});

ReactDOM.render(

<div>

<Input/>

</div>

,

destination

);

</script>

</body>

</html>

## JsX 语法

### 注释的写法

ReactDOM.render(

<div>

{/\* hello this is a comment \*/}

<Input/>

</div>

### 大小写必须注意

如果不是组件，而是html的标准控件，那么不需要注意大小写，但是如果是react组件，必须采用驼峰式写法。

JSX 不是HTML!. JSX是一段可执行代码，经过babel转译成

## 语法

### 创建json样式之后怎么用

var Card=React.createClass({

render:function(){

var cardStyle={

height:200,

width:150,

backgroundColor:"#FFF",

WebkitFilter:"drop-shadow(0px 5px 5px #666)",

filter:"drop-shadow(0px 0px 5px #666)"

};

return (

<div style={cardStyle}>

<Square/>

<Label/>

</div>

)

}

});

## Css样式兼容问题

### 首字母必须大写

WebkitFilter:"drop-shadow(0px 5px 5px #666)",

### 外部数据传到子控件中

必须在父组件中通过属性绑定传递一下，然后才能再传递给子组件。如果

<!DOCTYPE html>

<html>

<head>

<title>Compositive Card Component</title>

<script src="react.js"></script>

<script src="react-dom.js"></script>

<script src="browser.min.js"></script>

<style>

#container{

padding:50px;

background-color: #FFF;

}

</style>

</head>

<body>

<div id="container"></div>

<script type="text/babel">

var Square= React.createClass({

render:function(){

var squareStyle={

height:150,

backgroundColor:this.props.color

};

return (

<div style={squareStyle}>

</div>

);

}

});

var Label=React.createClass({

render:function(){

var labelStyle={

fontFamily:"sans-serif",

fontWeight:"bold",

padding:13,

margin:0

};

return (

<div style={labelStyle}>

<p>{this.props.color}</p>

</div>

);

}

});

var Card=React.createClass({

render:function(){

var cardStyle={

height:200,

width:150,

backgroundColor:"#FFF",

WebkitFilter:"drop-shadow(0px 5px 5px #666)",

filter:"drop-shadow(0px 0px 5px #666)"

};

return (

<div style={cardStyle}>

<Square color={this.props.color}/>

<Label color={this.props.color}/>

</div>

)

}

});

var destination = document.querySelector("#container");

ReactDOM.render(

<div>

<Card color="#DD6664"/>

</div>

,

destination

)

</script>

</body>

</html>

# 常识

## Jsx是什么

React 使用 JSX 来替代常规的 JavaScript。

JSX 是一个看起来很像 XML 的 JavaScript 语法扩展。

我们不需要一定使用 JSX，但它有以下优点：

* JSX 执行更快，因为它在编译为 JavaScript 代码后进行了优化。
* 它是类型安全的，在编译过程中就能发现错误。
* 使用 JSX 编写模板更加简单快速。

## 路由配置简写

<script type="text/babel">

var {

Router,

Route,

IndexRoute,

IndexLink,

Link,

hashHistory

}=ReactRouter;

ReactDOM.render(

<Router history={hashHistory}>

<Route path="/" component={SinglePage} />

</Router>

,

destination

);

</script>

# 开发工具

## Babel webpack

