React

简介

React是Facebook开发并开源的前端框架。
当时他们的团队在市面上没有找到合适的MVC框架，就自己写了一个Js框架，用来架设Instagram（图片分享社交网络）。2013年React开源。

React解决的是前端MVC框架中的View视图层的问题。

Virtual DOM

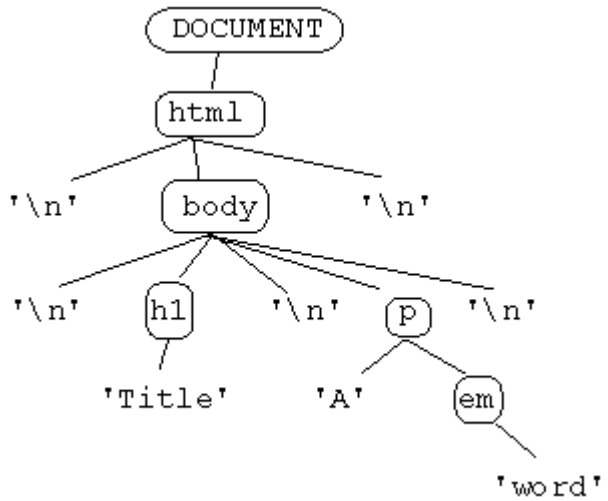
DOM（文档对象模型Document Object Model）

The Document

```
<html>
<body>
<h1>Title</h1>
<p>A <em>word</em></p>
</body>
</html>
```

The DOM Tree

```
DOCUMENT
└─ELEMENT: html
   ├─TEXT: '\n'
   ├─ELEMENT: body
   │  ├─TEXT: '\n'
   │  ├─ELEMENT: h1
   │  │  └─TEXT: 'Title'
   │  ├─TEXT: '\n'
   │  ├─ELEMENT: p
   │  │  ├─TEXT: 'A'
   │  │  └─ELEMENT: em
   │  │     └─TEXT: word
   │  └─TEXT: '\n'
   └─TEXT: '\n'
```



将网页内所有内容映射到一棵树型结构的层级对象模型上，浏览器提供对DOM的支持，用户可以是用脚本调用DOM API来动态的修改DOM结点，从而达到修改网页的目的，这种修改是浏览器中完成，浏览器会根据DOM的改变重绘改变的DOM结点部分。

修改DOM重新渲染代价太高，前端框架为了提高效率，尽量减少DOM的重绘，提出了Virtual DOM，所有的修改都是先在Virtual DOM上完成的，通过比较算法，找出浏览器DOM之间的差异，使用这个差异操作DOM，浏览器只需要渲染这部分变化就行了。
React实现了DOM Diff算法可以高效比对Virtual DOM和DOM间的差异。

支持JSX语法

JSX是一种JavaScript和XML混写的语法，是JavaScript的扩展。

```
React.render(
    <div>
        <div>
            <div>content</div>
        </div>
    </div>,
    document.getElementById('example')
);
```

测试程序

替换`/src/index.js`为下面的代码

```
import React from 'react';
import ReactDom from 'react-dom';

class Root extends React.Component {
  render() {
    return <div>Hello magedu</div>;
  }
}

ReactDom.render(<Root/>, document.getElementById('root'));
```

保存文件后，会自动编译，并重新装载刷新浏览器端页面。

```
webpack: Compiling...
Hash: ea590f222bccb8b68679
Version: webpack 2.7.0
Time: 603ms
                                Asset       Size  Chunks                    C
hunk Names
                           bundle.js    1.28 MB       0  [emitted]  [big]  a
pp
    0.31f1b8d9803a315704b5.hot-update.js    2.81 kB    0, 0  [emitted]         a
pp, app
    31f1b8d9803a315704b5.hot-update.json   43 bytes          [emitted]
                      bundle.js.map    1.57 MB       0  [emitted]         a
pp
0.31f1b8d9803a315704b5.hot-update.js.map  743 bytes    0, 0  [emitted]         a
pp, app
webpack: Compiled successfully.
```

程序解释

`import React from 'react';` 导入react模块
`import ReactDOM from 'react-dom';` 导入react的DOM模块

`class Root extends React.Component` 组件类定义，从React.Component类上继承。这个类生成JSXElement对象即React元素。
`render()` 渲染函数。返回组件中渲染的内容。注意，只能返回**唯一 一个顶级元素**回去。

`ReactDom.render(<Root/>, document.getElementById('root'));` 第一个参数是JSXElement对象，第二个是DOM的Element元素。将React元素添加到DOM的Element元素中并渲染。

还可以使用React.createElement创建react元素，第一参数是React组件或者一个HTML的标签名称（例如div、span）。

```
return React.createElement('div', null, 'Welcome to Magedu.');
ReactDom.render(React.createElement(Root), document.getElementById('root'));
```

改写后代码为

```
import React from 'react';
import ReactDom from 'react-dom';

class Root extends React.Component {
  render() {
    //return <div>Hello magedu</div>;
    return React.createElement('div', null, 'Welcome to Magedu.');
  }
}

//ReactDom.render(<Root/>, document.getElementById('root'));
ReactDom.render(React.createElement(Root), document.getElementById('root'));
```

很明显JSX更简洁易懂，推荐使用JSX语法。

增加一个子元素

```
import React from 'react';
import ReactDom from 'react-dom';


class SubEle extends React.Component {
  render() {
    return <div>Sub content</div>;
  }
}

class Root extends React.Component {
  render() {
    return (
      <div>
```

```
        <h2>Welcome magedu.com</h2>
        <br />
        <SubEle />
      </div>);
  }
}

ReactDom.render(<Root />, document.getElementById('root'));
```

注意：
1、React组件的render函数return，只能是一个顶级元素
2、JSX语法是XML，要求所有元素必须闭合，注意`<br />`不能写成`<br>`

JSX规范

- 约定标签中首字母小写就是html标记，首字母大写就是组件
- 要求严格的HTML标记，要求所有标签都必须闭合。br也应该写成`<br />`，/前留一个空格。
- 单行省略小括号，多行请使用小括号
- 元素有嵌套，建议多行，注意缩进
- **JSX表达式**：表达式使用{}括起来，如果大括号内使用了引号，会当做字符串处理，例如`<div>{'2>1?true:false'}</div>`里面的表达式成了字符串了

组件状态state **

每一个**React组件**都有一个**状态属性state**，它是一个JavaScript对象，可以为它定义属性来保存值。
如果状态变化了，会触发UI的重新渲染。使用setState()方法可以修改state值。
注意：state是每个组件自己内部使用的，是组件自己的属性。

依然修改/src/index.js

```
import React from 'react';
import ReactDom from 'react-dom';


class Root extends React.Component {
  // 定义一个对象
  state = {
    p1: 'magedu',
    p2: '.com'
  };
  render() {
    this.state.p1 = 'www.magedu'; // 可以更新
    // this.setState({p1:'www.magedu'}); // 不可以对还在更新中的state使用setState
    // Warning: setState(...): Cannot update during an existing state transition (such as within
render).
    // Render methods should be a pure function of props and state.
    return (
      <div>
```

```
        <div>Welcome to {this.state.p1}{this.state.p2}</div>
        <br />
      </div>);
  }
}

ReactDom.render(<Root />, document.getElementById('root'));
```

如果将`this.state.p1 = 'www.magedu'`改为`this.setState({p1:'www.magedu'});`就会出警告。
可以使用延时函数`setTimeout(() => this.setState({ p1: 'www.magedu' }), 5000);`即可。

复杂状态例子

先看一个网页

```
<html>
<head>
    <script type="text/javascript">
        function getEventTrigger(event) {
            x = event.target; // 从事件中获取元素
            alert("触发的元素的id是: " + x.id);
        }
    </script>
</head>
<body>
    <div id="t1" onmousedown="getEventTrigger(event)">
        点击这句话，会触发一个事件，并弹出一个警示框
    </div>
</body>
</html>
```


div的id是t1，鼠标按下事件捆绑了一个函数，只要鼠标按下就会触发调用getEventTrigger函数，浏览器会送给它一个参数event。event是事件对象，当事件触发时，event包含触发这个事件的对象。

HTML DOM的JavaScript事件

属性	此事件发生在何时
onabort	图像的加载被中断
onblur	元素失去焦点
onchange	域的内容被改变
onclick	**当用户点击某个对象时调用的事件句柄**
ondblclick	当用户双击某个对象时调用的事件句柄
onerror	在加载文档或图像时发生错误
onfocus	元素获得焦点
onkeydown	某个键盘按键被按下
onkeypress	某个键盘按键被按下并松开
onkeyup	某个键盘按键被松开
onload	一张页面或一幅图像完成加载
onmousedown	鼠标按钮被按下
onmousemove	鼠标被移动
onmouseout	鼠标从某元素移开
onmouseover	鼠标移到某元素之上
onmouseup	鼠标按键被松开
onreset	重置按钮被点击
onresize	窗口或框架被重新调整大小
onselect	文本被选中
onsubmit	确认按钮被点击
onunload	用户退出页面

使用React实现上面的传统的HTML

```
import React from 'react';
import ReactDom from 'react-dom';

class Toggle extends React.Component {
  state = { flag: true }; // 类中定义state
  handleClick(event) {
    console.log(event.target.id);
    console.log(event.target === this);
    console.log(this);
    console.log(this.state);
```

```
    this.setState({ flag: !this.state.flag });
  }
  render() {/* 注意一定要绑定this  onClick写成小驼峰 */
    return <div id="t1" onClick={this.handleClick.bind(this)}>
      点击这句话，会触发一个事件。{this.state.flag.toString()}
    </div>;
  }
}

class Root extends React.Component {
  // 定义一个对象
  state = { p1: 'www.magedu', p2: '.com' }; // 构造函数中定义state
  render() {
    // this.state.p1 = 'python.magedu'; // 可以修改属性值
    // this.setState({p1:'python.magedu'}); // 不可以对还在更新中的state使用setState
    // Warning: setState(...): Cannot update during an existing state transition (such as within
render).
    setTimeout(() => this.setState({ p1: 'python.magedu' }), 5000);
    // setInterval(() => this.setState({ p1: 'python.magedu' }), 5000);
    return (
      <div>
        <div>Welcome to {this.state.p1}{this.state.p2}</div>
        <br />
        <Toggle />
      </div>);
  }
}

ReactDom.render(<Root />, document.getElementById('root'));
```

分析

Toggle类

它有自己的state属性。

当render完成后，网页上有一个div标签，div标签对象捆绑了一个click事件的处理函数，div标签内有文本内容。

如果通过点击左键，就触发了click方法关联的handleClick函数，在这个函数里将状态值改变。

状态值state的改变将引发render重绘。

如果组件自己的state变了，只会触发自己的render方法重绘。

注意：

{this.handleClick.bind(this)}，不能外加引号

this.handleClick.bind(this) 一定要绑定this，否则当触发捆绑的函数时，this是函数执行的上下文决定的，this已经不是触发事件的对象了。

console.log(event.target.id)，取回的产生事件的对象的id，但是这不是我们封装的组件对象。所以，console.log(event.target===this)是false。所以这里一定要用this，而这个this是通过绑定来的。

React中的事件

- 使用小驼峰命名
- 使用JSX表达式，表达式中指定事件处理函数
- 不能使用return false，如果要阻止事件默认行为，使用event.preventDefault()

属性props **

props就是组件的属性properties。

把React组件当做标签使用，可以为其增加属性，如下

`<Toggle name="school" parent={this} />`

为上面的Toggle元素增加属性：
1、`name = "school"`，这个属性会作为一个单一的对象传递给组件，加入到组件的props属性中
2、`parent = {this}`，注意这个this是在Root元素中，指的是Root组件本身
3、在Root中为使用JSX语法为Toggle增加子元素，这些子元素也会被加入Toggle组件的props.children中

```
import React from 'react';
import ReactDom from 'react-dom';

class Toggle extends React.Component {
  state = { flag: true }; // 类中定义state
  handleClick(event) {
    console.log(event.target.id);
    console.log(event.target === this);
    console.log(this);
    console.log(this.state);
    this.setState({ flag: !this.state.flag });
    //this.props.name = 1000; //readonly
  }
  render() {/* 注意一定要绑定this  onClick写成小驼峰 */
    return <div id="t1" onClick={this.handleClick.bind(this)}>
      点击这句话，会触发一个事件。{this.state.flag.toString()}<br />
      显示props<br />
      {this.props.name} : {this.props.parent.state.p1 + this.props.parent.state.p2}<br />
      {this.props.children}
    </div>;
  }
}

class Root extends React.Component {
  // 定义一个对象
  state = { p1: 'www.magedu', p2: '.com' }; // 构造函数中定义state
  render() {
    // this.state.p1 = 'python.magedu'; // 可以修改属性值
    // this.setState({p1:'python.magedu'}); // 不可以对还在更新中的state使用setState
    // Warning: setState(...): Cannot update during an existing state transition (such as within
render).
    setTimeout(() => this.setState({ p1: 'python.magedu' }), 5000);
    // setInterval(() => this.setState({ p1: 'python.magedu' }), 5000);
    return (
      <div>
        <div>Welcome to {this.state.p1}{this.state.p2}</div>
        <br />
        <Toggle name="school" parent={this}>{/*自定义2个属性通过props传给Toggle组件对象*/}
          <hr />{/*子元素通过props.children访问*/}
          <span>我是Toggle元素的子元素</span>{/*子元素通过props.children访问*/}
```

```
        </Toggle>
      </div>);
  }
}

ReactDom.render(<Root />, document.getElementById('root'));
```

尝试修改props中的属性值，会抛出`TypeError: Cannot assign to read only property 'name' of object '#<Object>'`异常。也就是说props在组件内部不能修改，只读。

> 应该说，state是私有private的属于组件自己的属性，组件外无法直接访问。可以修改state，但是建议使用setState方法。
> props是公有public属性，组件外也可以访问，但只读。

构造器constructor

使用ES6的构造器，要提供一个参数props，并把这个参数使用super传递给父类

```
import React from 'react';
import ReactDom from 'react-dom';

class Toggle extends React.Component {
  constructor (props) {
    super(props); // 一定要调用super父类构造器，否则报错
    this.state = { flag: true }; // 类中定义state
  }

  handleClick(event) {
    console.log(event.target.id);
    console.log(event.target === this);
    console.log(this);
    console.log(this.state);
    this.setState({ flag: !this.state.flag });
  }
  render() {/* 注意一定要绑定this  onClick写成小驼峰 */
    return <div id="t1" onClick={this.handleClick.bind(this)}>
      点击这句话，会触发一个事件。{this.state.flag.toString()}<br />
      显示props<br />
      {this.props.name} : {this.props.parent.state.p1 + this.props.parent.state.p2}<br />
      {this.props.children}
    </div>;
  }
}

class Root extends React.Component {
  // 定义一个对象
  constructor(props) {
    super(props); // 一定要调用super父类构造器，否则报错
    this.state = { p1: 'www.magedu', p2: '.com' }; // 构造函数中定义state
  }

  render() {
```

```
    // this.state.p1 = 'python.magedu'; // 可以修改属性值
    // this.setState({p1:'python.magedu'}); // 不可以对还在更新中的state使用setState
    // Warning: setState(...): Cannot update during an existing state transition (such as within
render).
    setTimeout(() => this.setState({ p1: 'python.magedu' }), 5000);
    // setInterval(() => this.setState({ p1: 'python.magedu' }), 5000);
    return (
      <div>
        <div>Welcome to {this.state.p1}{this.state.p2}</div>
        <br />
        <Toggle name="school" parent={this}>{/*自定义2个属性通过props传给Toggle组件对象*/}
          <hr />{/*子元素通过props.children访问*/}
          <span>我是Toggle元素的子元素</span>{/*子元素通过props.children访问*/}
        </Toggle>
      </div>);
  }
}

ReactDom.render(<Root />, document.getElementById('root'));
```

组件的生命周期 *

组件的生命周期可分成三个状态：

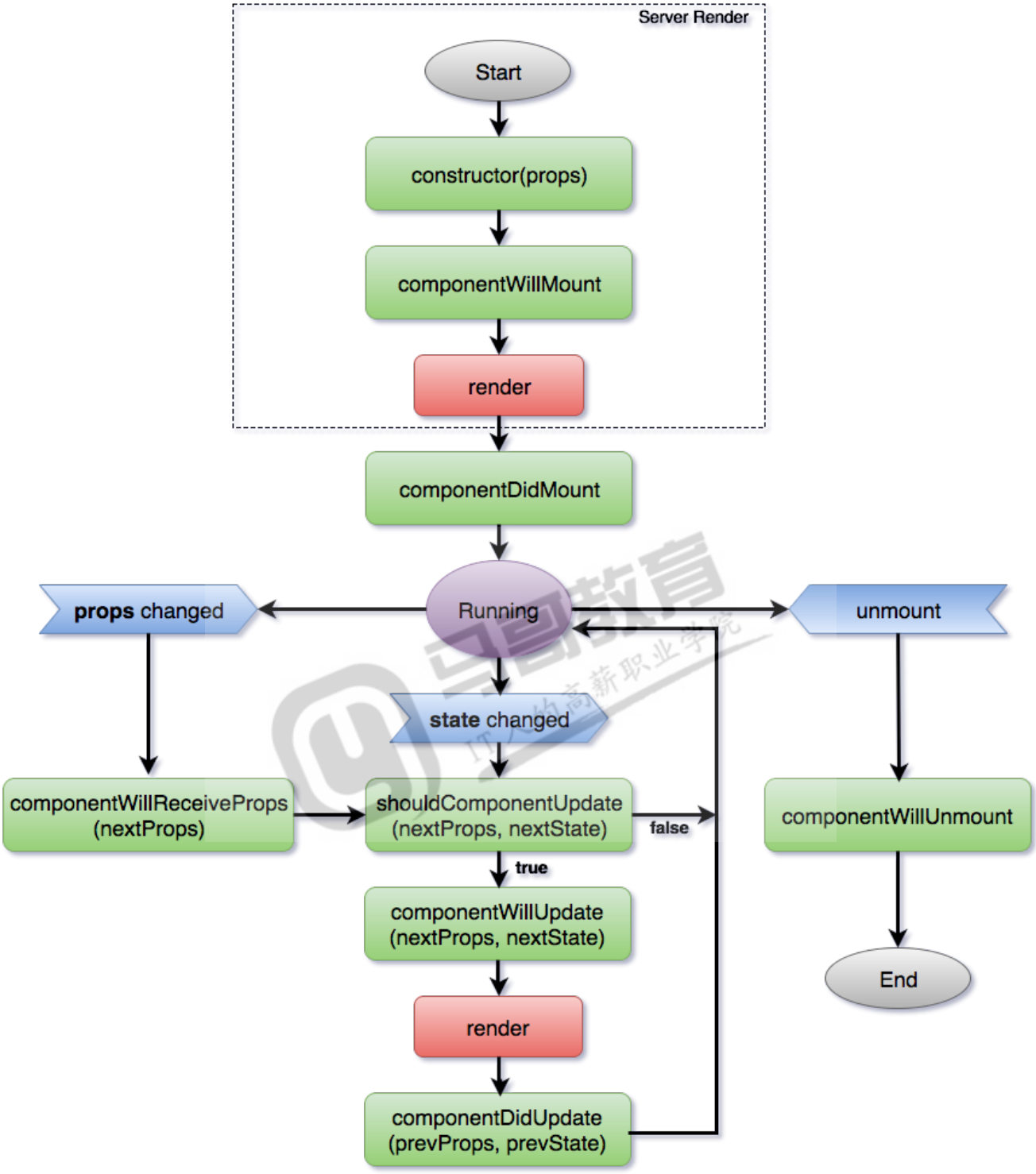
- Mounting：已插入真实 DOM
- Updating：正在被重新渲染
- Unmounting：已移出真实 DOM

组件的生命周期状态，说明在不同时机访问组件，组件正处在生命周期的不同状态上。
在不同的生命周期状态访问，就产生不同的方法。

生命周期的方法如下：

- 装载组件触发
  - componentWillMount 在渲染前调用，在客户端也在服务端。只会在装载之前调用一次。
  - componentDidMount : 在第一次渲染后调用，只在客户端。之后组件已经生成了对应的DOM结构，可以通过this.getDOMNode()来进行访问。 如果你想和其他JavaScript框架一起使用，可以在这个方法中调用setTimeout, setInterval或者发送AJAX请求等操作(防止异部操作阻塞UI)。只在装载完成后调用一次，在render之后。
- 更新组件触发。这些方法不会在首次render组件的周期调用
  - componentWillReceiveProps(nextProps) 在组件接收到一个新的prop时被调用。这个方法在初始化render时不会被调用。
  - shouldComponentUpdate(nextProps, nextState) 返回一个布尔值。在组件接收到新的props或者state时被调用。在初始化时或者使用forceUpdate时不被调用。
    - 可以在你确认不需要更新组件时使用。
    - 如果设置为false，就是不允许更新组件，那么componentWillUpdate、componentDidUpdate不会执行。
  - componentWillUpdate(nextProps, nextState) 在组件接收到新的props或者state但还没有render时被调用。在初始化时不会被调用。
  - componentDidUpdate(prevProps, prevState) 在组件完成更新后立即调用。在初始化时不会被调用。

- 卸载组件触发
  - componentWillUnmount在组件从 DOM 中移除的时候立刻被调用。



由图可知
constructor构造器是最早执行的函数。
组件构建好之后，如果更新组件的state或props（注意在组件内props是只读的），就会在render渲染前触发一系列的`更新生命周期函数`。

因此，重新编写/src/index.js。
构造两个组件，在子组件Sub中，加入所有生命周期函数。

下面的例子添加是装载、卸载组件的生命周期函数

```
import React from 'react';
import ReactDom from 'react-dom';

class Sub extends React.Component {
  constructor(props) {
    console.log('Sub constructor')
    super(props); // 调用父类构造器
    this.state = { count: 0 };
  }

  handleClick(event) {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    console.log('Sub render');
    return (<div id="sub" onClick={this.handleClick.bind(this)}>
      Sub's count = {this.state.count}
    </div>);
  }

  componentWillMount() {
    // constructor之后，第一次render之前
    console.log('Sub componentWillMount');
  }

  componentDidMount() {
    // 第一次render之后
    console.log('Sub componentDidMount');
  }

  componentWillUnmount() {
    // 清理工作
    console.log('Sub componentWillUnmount');
  }
}

class Root extends React.Component {
  constructor(props) {
    console.log('Root Constructor')
    super(props); // 调用父类构造器
    // 定义一个对象
    this.state = {};
  }

  render() {
    return (
      <div>
        <Sub />
      </div>);
  }
}
```

```
ReactDom.render(<Root />, document.getElementById('root'));
```

上面可以看到顺序是

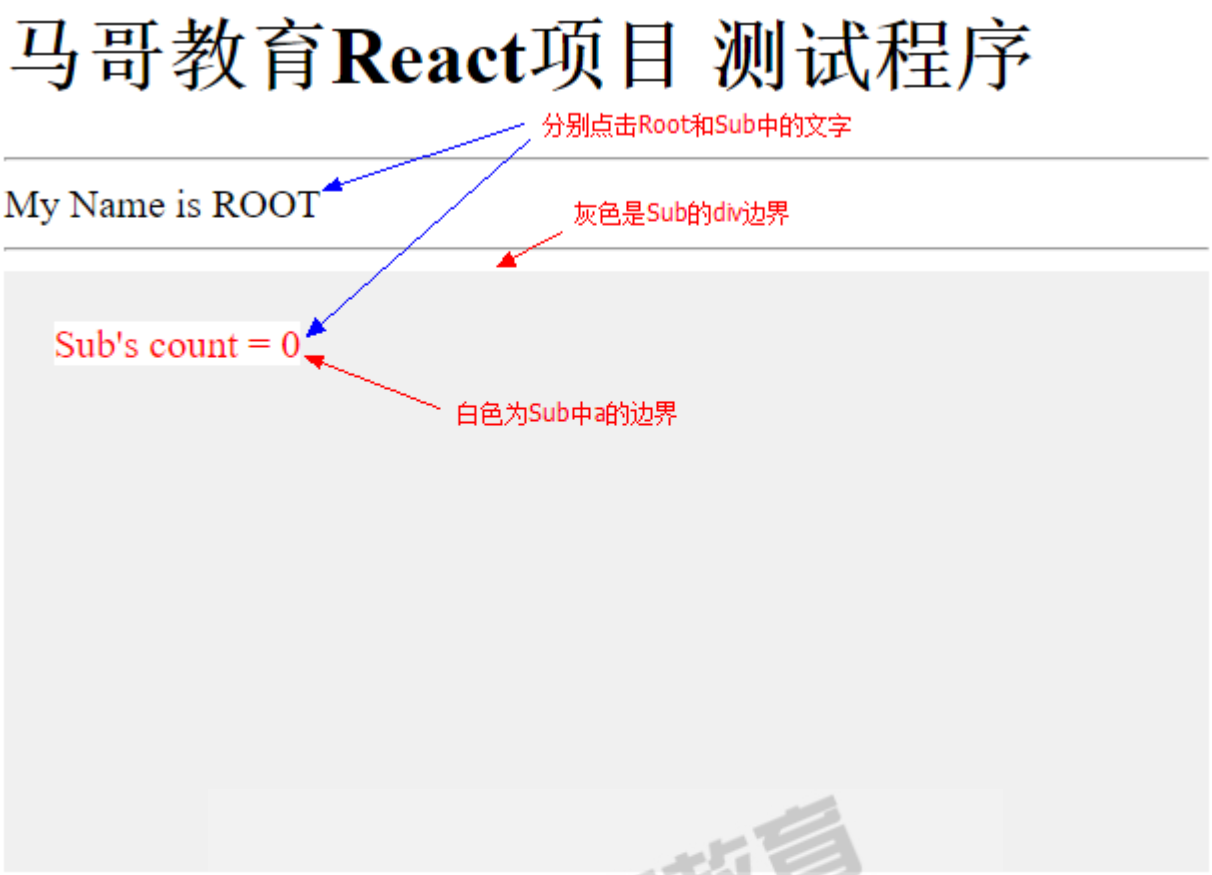
`constructor -> componentWillMount -> render -> componentDidMount ----state或props改变----> render`

增加 更新组件函数

为了演示props的改变，为Root元素增加一个click事件处理函数

```
import React from 'react';
import ReactDom from 'react-dom';

class Sub extends React.Component {
  constructor(props) {
    console.log('Sub constructor')
    super(props); // 调用父类构造器
    this.state = { count: 0 };
  }

  handleClick(event) {
    this.setState({ count: this.state.count + 1 });// 不要用++
  }

  render() {
    console.log('Sub render');
    return (<div style={{ height: 200 + 'px', color: 'red', backgroundColor: '#f0f0f0',
padding:'20px' }}>
      <a id="sub" onClick={this.handleClick.bind(this)} style={{backgroundColor: 'white'}}>
        Sub's count = {this.state.count}
      </a>
    </div>);
  }

  componentWillMount() {
    // constructor之后，第一次render之前
    console.log('Sub componentWillMount');
  }

  componentDidMount() {
    // 第一次render之后
    console.log('Sub componentDidMount');
  }

  componentWillUnmount() {
    // 清理工作
    console.log('Sub componentWillUnmount');
  }

  componentWillReceiveProps(nextProps) {
    // props变更时，接到新props了，交给shouldComponentUpdate。
    // props组件内只读，只能从外部改变
    console.log(this.props);
    console.log(nextProps);
```

```
    console.log('Sub componentWillReceiveProps', this.state.count);
  }

  shouldComponentUpdate(nextProps, nextState) {
    // 是否组件更新，props或state方式改变时，返回布尔值，true才会更新
    console.log('Sub shouldComponentUpdate', this.state.count, nextState);
    return true; // return false将拦截更新
  }

  componentWillUpdate(nextProps, nextState) {
    // 同意更新后，真正更新前，之后调用render
    console.log('Sub componentWillUpdate', this.state.count, nextState);
  }

  componentDidUpdate(prevProps, prevState) {
    // 同意更新后，真正更新后，在render之后调用
    console.log('Sub componentDidUpdate', this.state.count, prevState);
  }
}

class Root extends React.Component {
  constructor(props) {
    console.log('Root Constructor')
    super(props); // 调用父类构造器
    // 定义一个对象
    this.state = { flag: true, name: 'root' };
  }

  handleClick(event) {
    this.setState({
      flag: !this.state.flag,
      name: this.state.flag ? this.state.name.toLowerCase() : this.state.name.toUpperCase()
    });
  }

  render() {
    return (
      <div id="root" onClick={this.handleClick.bind(this)}>
        My Name is {this.state.name}
        <hr />
        <Sub /> {/*父组件的render，会引起下一级组件的更新流程，导致props重新发送，即使子组件props没有
改变过*/}
      </div>);
  }
}

ReactDom.render(<Root />, document.getElementById('root'));
```



componentWillMount 第一次装载，在首次render之前。例如控制state、props
componentDidMount 第一次装载结束，在首次render之后。例如控制state、props

componentWillReceiveProps 在组件内部，props是只读不可变的，但是这个函数可以接收到新的props，可以对props做一些处理，this.props = {name:'roooooot'};这就是偷梁换柱。componentWillReceiveProps触发，也会走shouldComponentUpdate。
shouldComponentUpdate 判断是否需要组件更新，就是是否render，精确的控制渲染，提高性能。
componentWillUpdate 在除了首次render外，每次render前执行，componentDidUpdate在render之后调用。

不过，大多数时候，用不上这些函数，这些钩子函数是为了精确的控制。

修改Root组件render中的这一句为 `<div id="root" onDoubleClick={this.handleClick.bind(this)}>` ，可以看到点击Sub中红色的文字，Root不会重绘。

如果子组件和父组件使用了相同的事件，可以认为点击子组件也是点击了父组件，父组件重绘，就会把子组件props更新，引起子组件组件更新流程，就会从componentWillReceiveProps开始执行。如果子组件自己修改自己的state，不会执行componentWillReceiveProps。

无状态组件

React从15.0开始支持无状态组件，定义如下

```
import React from 'react';
import ReactDom from 'react-dom';

function Root(props) {
  return <div>{props.schoolName}</div>;
}

ReactDom.render(<Root schoolName="magedu" />, document.getElementById('root'));
```

无状态组件，也叫**函数式组件**。
开发中，很多情况下，组件其实很简单，不需要state状态，也不需要使用生命周期函数。无状态组件很好的满足了需要。
无状态组件函数应该提供一个参数props，返回一个React元素。
无状态组件函数本身就是render函数。

改写上面的代码

```
import React from 'react';
import ReactDom from 'react-dom';

let Root = props => <div>{props.schoolName}</div>;

ReactDom.render(<Root schoolName="magedu" />, document.getElementById('root'));
```