IT前沿技术在线大学

www.zhenchuanx.com

React

Part 1. 快速入门

简单的React组件是怎样的

```jsx
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Mark" />
```

```jsx
return React.createElement('div', {className: 'shopping-list'},
  React.createElement('h1', /* ... h1 children ... */),
  React.createElement('ul', /* ... ul children ... */)
);
```

```jsx
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Mark" />
```

this.props 是从外部传进来的属性

```jsx
return React.createElement('div', {className: 'shopping-list'},
  React.createElement('h1', /* ... h1 children ... */),
  React.createElement('ul', /* ... ul children ... */)
);
```

```jsx
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Mark" />
```

className替代class

```jsx
return React.createElement('div', {className: 'shopping-list'},
  React.createElement('h1', /* ... h1 children ... */),
  React.createElement('ul', /* ... ul children ... */)
);
```

组件使用 & 传递数据

```
class Square extends React.Component {
  render() {
    return (
      <button className="square">
        {this.props.value}
      </button>
    );
  }
}
```

```
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={i} />;
  }
```

https://codepen.io/gaearon/pen/aWWQOG?editors=0010

```
class Square extends React.Component {
  render() {
    return (
      <button className="square">
        {this.props.value}
      </button>
    );
  }
}
```

```
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={i} />;
  }
```

绑事件

```
class Square extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: null,
    };
  }

  render() {
    return (
      <button className="square" onClick={() => this.setState({value: 'X'})}>
        {this.state.value}
      </button>
    );
  }
}
```

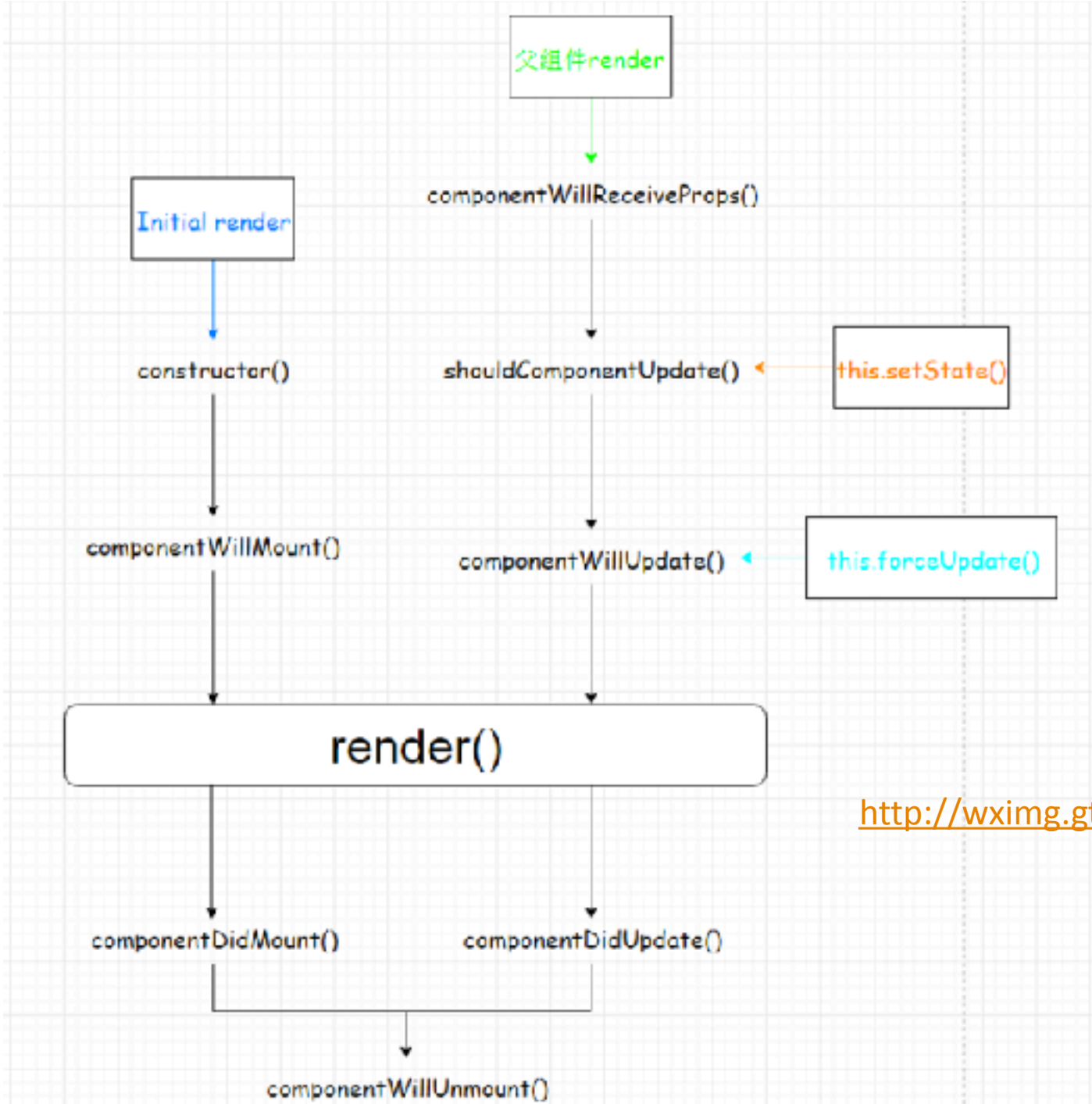
https://codepen.io/gaearon/pen/VbbVLg?editors=0010

```
class Square extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: null,
    };
  }

  render() {
    return (
      <button className="square" onClick={() => this.setState({value: 'X'})}>
        {this.state.value}
      </button>
    );
  }
}
```

剩下的大家自己看着办～

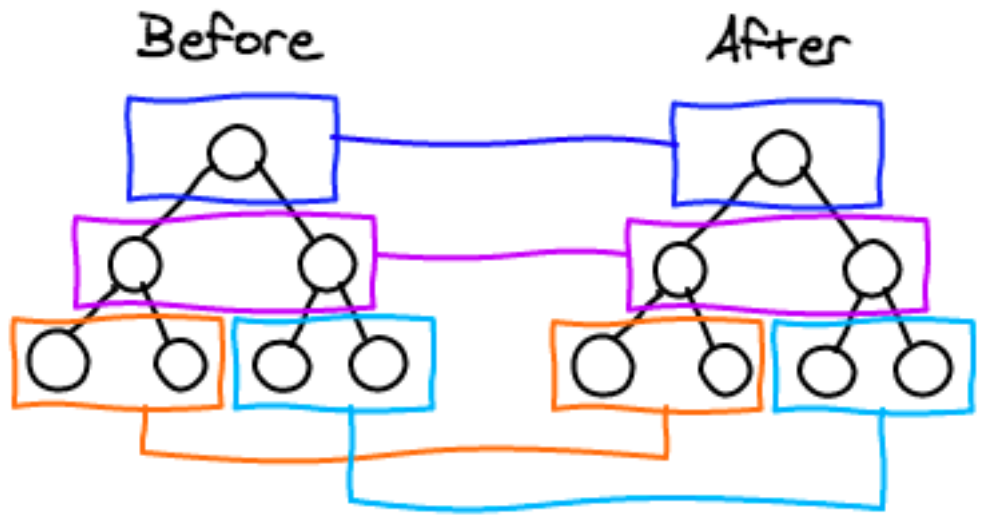
Part 2. 生命周期



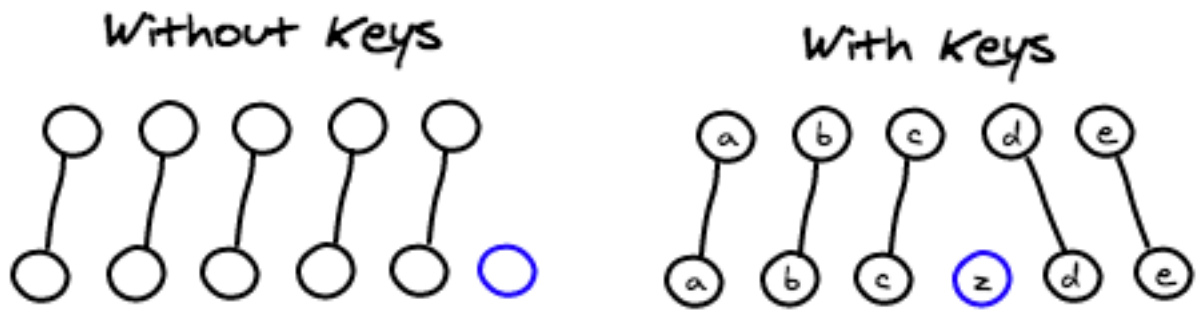
http://wximg.gtimg.com/shake_tv/test/lifeCycle2113.html

Part 3. diff算法取巧

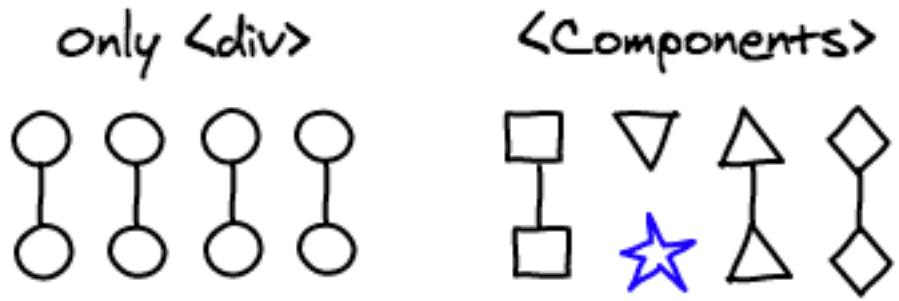
分层对比



基于key匹配



基于自定义组件优化



Part 4. 渲染逻辑

- 首次渲染　　对一个虚拟DOM遍历并创建相应的DOM，并append到容器中
- 再次渲染　　对比两个虚拟DOM，生成patch，然后一次性执行完所有patch

Part 4. key

```
// 旧v-dom
<ul>
  <li key="1">first</li>
  <li key="2">second</li>
</ul>
// 新v-dom
<ul>
  <li key="0">zero</li>
  <li key="1">first</li>
  <li key="2">second</li>
</ul>
```

React 会怎么操作呢?

```
<ul>{list.map((v,idx)=><li key={idx}>{v}</li>)}</ul>
// ['a','b','c']=>
<ul>
    <li key="0">a</li>
    <li key="1">b</li>
    <li key="2">c</li>
</ul>
// 数组重排 -> ['c','a','b'] =>
<ul>
    <li key="0">c</li>
    <li key="1">a</li>
    <li key="2">b</li>
</ul>
```

React 会怎么操作呢?

其实不仅仅是列表才能用key！

```
// 旧
<div>
    <div>hello</div>
    <p>User: Daniel</p>
    <div>
        <input>
    </div>
</div>

// 新
<div>
    <div>hello</div>
    <div>
        <input>
    </div>
</div>
```

React 会怎么操作呢?

Part 5. Pure Component

```
225     if (ctor.prototype && ctor.prototype.isPureReactComponent) {
226       return (
227         !shallowEqual(oldProps, newProps) || !shallowEqual(oldState, newState)
228       );
229     }
```

https://github.com/facebook/react/blob/0f2f90bd9a9daf241d691bf4af3ea2e3a263c0e3/packages/react-reconciler/src/ReactFiberClassComponent.js#L225-L229

```
function shallowEqual(objA: mixed, objB: mixed): boolean {
  if (is(objA, objB)) {
    return true;
  }

  if (typeof objA !== 'object' || objA === null ||
      typeof objB !== 'object' || objB === null) {
    return false;
  }

  const keysA = Object.keys(objA);
  const keysB = Object.keys(objB);

  if (keysA.length !== keysB.length) {
    return false;
  }

  // Test for A's keys different from B.
  for (let i = 0; i < keysA.length; i++) {
    if (
      !hasOwnProperty.call(objB, keysA[i]) ||
      !is(objA[keysA[i]], objB[keysA[i]])
    ) {
      return false;
    }
  }

  return true;
}
```

由于Pure Component的实现原理，
其跟不可变数据搭配味道更佳～

如果不跟不可变数据搭配有啥问题?

```
{this.props.items.map(i =>
    <Cell data={i} options={this.props.options || []} />
)}
```

在Pure Component中，这会有啥问题?

Part 6. Stateless Component

```
import React from 'react';
const Button = ({
  day,
  increment
}) => {
  return (
    <div>
      <button onClick={increment}>Today is {day}</button>
    </div>
  )
}

Button.propTypes = {
  day: PropTypes.string.isRequired,
  increment: PropTypes.func.isRequired,
}
```

大家觉得Stateless Component用途是啥?

Component vs Stateless Functional component

1. `Component` 包含内部state，而 `Stateless Functional Component` 所有数据都来自props，没有内部state;
2. `Component` 包含的一些生命周期函数，`Stateless Functional Component` 都没有，因为 `Stateless Functional component` 没有 `shouldComponentUpdate` ,所以也无法控制组件的渲染，也即是说只要是收到新的props，`Stateless Functional Component` 就会重新渲染。
3. `Stateless Functional Component` 不支持Refs

对了，理论上他可以做性能优化，虽然现在（至少V15）根本没有，反而更慢。。。

Part 7. Container & Presentational components

大家来对比一下呗～

Part 8. HOC

```
function ppHOC(WrappedComponent) {
  return class PP extends React.Component {
    constructor(props) {
      super(props)
      this.state = {
        name: ''
      }

      this.onNameChange = this.onNameChange.bind(this)
    }
    onNameChange(event) {
      this.setState({
        name: event.target.value
      })
    }
    render() {
      const newProps = {
        name: {
          value: this.state.name,
          onChange: this.onNameChange
        }
      }
      return <WrappedComponent {...this.props} {...newProps}/>
    }
  }
}
```

```
@ppHOC
class Example extends React.Component {
  render() {
    return <input name="name" {...this.props.name}/>
  }
}
```

input 这样就可以快速变成受控组件了

```
function ppHOC(WrappedComponent) {
  return class PP extends React.Component {
    render() {
      return (
        <div style={{display: 'block'}}>
          <WrappedComponent {...this.props}/>
        </div>
      )
    }
  }
}
```

```
function iiHOC(WrappedComponent) {
  return class Enhancer extends WrappedComponent {
    render() {
      if (this.props.loggedIn) {
        return super.render()
      } else {
        return null
      }
    }
  }
}
```

渲染劫持

```
export function IIHOCDEBUGGER(WrappedComponent) {
  return class II extends WrappedComponent {
    render() {
      return (
        <div>
          <h2>HOC Debugger Component</h2>
          <p>Props</p> <pre>{JSON.stringify(this.props, null, 2)}</pre>
          <p>State</p><pre>{JSON.stringify(this.state, null, 2)}</pre>
          {super.render()}
        </div>
      )
    }
  }
}
```

开启debug的模式

但HOC有些坑～

```
render() {
    // 显然EnhancedComponent1 !== EnhancedComponent2
    const EnhancedComponent = enhance(MyComponent);
    // 导致组件每次都unmount/remount
    return <EnhancedComponent />;
}
```

不要在render里面写HOC

```
1  WrappedComponent.staticMethod = function() {/*...*/}
2  // apply to HOC
3  const EnhancedComponent = enhance(WrappedComponent);
4
5  typeof EnhancedComponent.staticMethod === 'undefined' // true
```

```
1  function enhance(WrappedComponent) {
2      class Enhance extends React.Component {/*...*/}
3      Enhance.staticMethod = WrappedComponent.staticMethod;
4      return Enhance;
5  }
```

静态方法不会被继承

```
function enhance(WrappedComponent) {
    return class Enhance extends React.Component {
        /*...*/
        getWrappedInstance() {
            return this.wrappedInstance;
        }

        render() {
            return <WrappedComponent ref={(el) => this.wrappedInstance = el} />
        }
    }
}

/*...*/
const EnhancedComponent = enhance(WrappedComponent);

<EnhancedComponent ref={(el) => {
    this.enhancedComponent = el;
}}/>

console.log(this.enhancedComponent.getWrappedInstance())
```

无法获得ref

IT前沿技术在线大学