REACT 入门

- React 简介
- JSX
- 组件
- 事件
- State
- 生命周期

REACT 简介

- 高性能
- 声明式
- 组件化
- 跨平台

MVVM

- 双向绑定
- 直接操作 DOM

虚拟 DOM

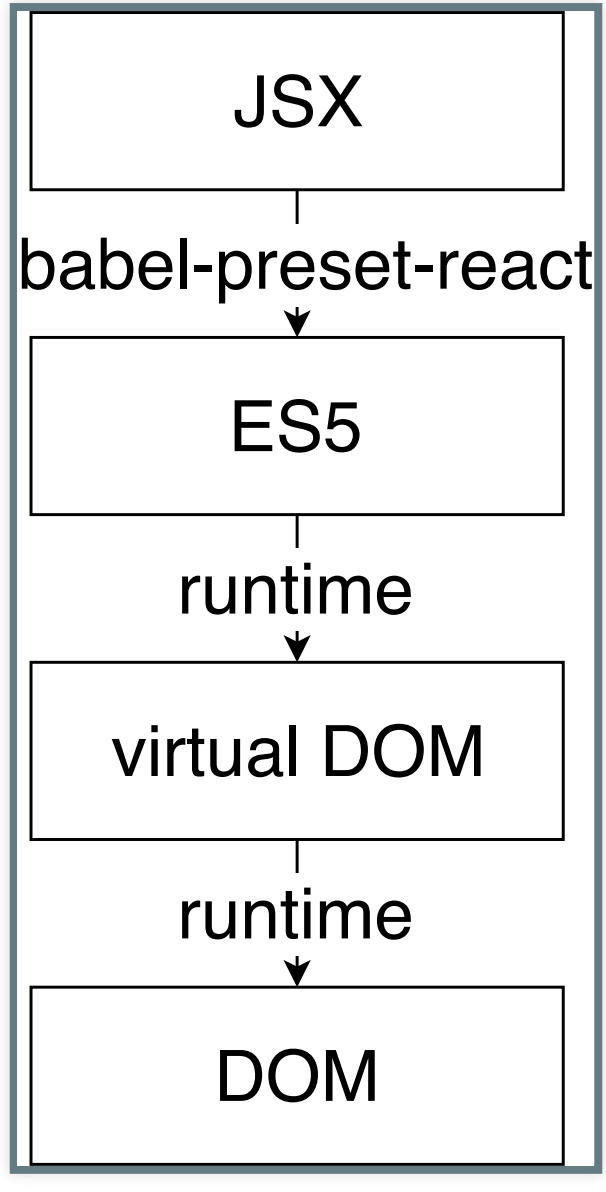
- 单向数据流
- 批量更新 DOM

```
<CommentList comments={comments} />
```

```
<!DOCTYPE HTML>
<html>
<head>React Hello World</head>
<body>
<div id="root"></div>
<script src="https://static.meituan.net/bs/react/16/umd/react.pro
<script src="https://static.meituan.net/bs/react-dom/16/umd/react
<script>
  ReactDOM.render(
    React.createElement('h1', null, 'Hello, world!'),
    document.getElementById('root'),
  );
</script>
</body>
</html>
```

JSX

```
const element = <p className="comment">Hello, world!</p>;
```


JSX
babel-preset-react
ES5
runtime
virtual DOM
runtime
DOM

ES5

```
const element = React.createElement(
  'p',
  {
    className: 'comment',
  },
  'Hello, world!'
);
```

引入 REACT

```
import React from 'react';
```

虚拟 DOM

```
const element = {
  type: 'p',
  props: {
    className: 'comment',
    children: 'Hello, world',
  },
};
```

使用变量

```
const element = <p>{`This is a comment: ${comment}`}</p>;
```

```
const element = <p>This is a comment: {comment}</p>;
```

使用属性 (PROPS)

```
const element = <img src={avatarURL} className="avatar" />;
```

```jsx
const element = <img src={avatarURL} className={'avatar'} />;
```

```jsx
const element = <img src={avatarURL} className="avatar" />;
```

```jsx
const element = <button disabled={true}>提交</button>;
```

```jsx
const element = <button disabled>提交</button>;
```

```jsx
const element = <img src={avatarURL} className="avatar" />;
```

```jsx
const element = <img src={avatarURL} style={{ display: 'block' }}
```

嵌套

```
const element = (
  <p>
    <img src={avatarURL} />
    <span>Hello World!</span>
  </p>
);
```

例子

列表和 KEY

```
const commentElements = comments.map(({ id, comment }) => {
  return <li key={id}>{comment}</li>;
});

const element = <ul>{commentElements}</ul>;
```

例子

有条件的渲染

```
// ok
const element = commentData.isBlocked ? null : <Comment />;
```

```
// better
const element = commentData.isBlocked || <Comment />;
```

- `false`
- `null`
- `undefined`
- `true`

空标签

写成 `<Tag />` 而不是 `<Tag></Tag>`

注释

```
const element = <div>{/* This is a comment */}</div>;
```

HTML 转义

```
const element = (
  <div
   dangerouslySetInnerHTML={{ __html: '<span>Text</span>' }}
  />
);
```

组件

自定义组件

- 函数式组件
- 类声明组件

函数式组件

```
const Comment = ({ comment }) => {
  return <li>{comment}</li>;
};

const element = (
  <ul>
    {comments.map(({ id, comment }) => {
        return <Comment key={id} comment={comment} />;
    })}
  </ul>
);
```

类声明组件

```
class Comment extends Component {
  state = {
    clicked: false,
  };

  render() {
    const { comment } = this.props;
    return <li>{comment}</li>;
  }
}
```

例子

V0.14 版本之前

```
const Comment = React.createClass({
  render() {
    const { comment } = this.props;
    return <li>{comment}</li>;
  }
});
```

动态的组件类型

```
const getCommentType = (commentData) => {
  if (commentData.isVIP) {
    return VIPComment;
  }
  return Comment;
};

const CommentType = getCommentType(commentData);

const element = <CommentType comment={comment} />;
```

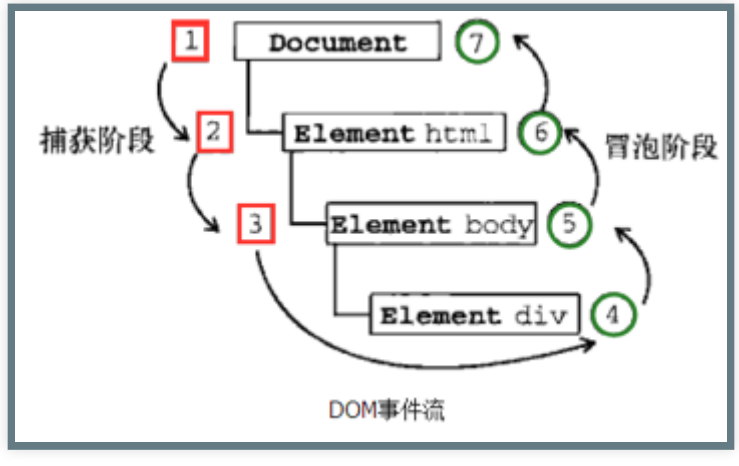
例子

事件

React 基于虚拟 DOM 实现了合成事件

事件模型

- 事件冒泡
- 事件委托



绑定方式

```
const handleClick = (e) => {
  // 浏览器标准事件
};
<button onClick={handleClick}>提交</button>;
```

THIS 的指向

```
class Submit extends Component {
  isSubmitting = false;

  handleClick(event) {
    // throws TypeError: Cannot read property 'isSubmitting' of n
    console.log(this.isSubmitting);
  }

  render() {
    return <button onClick={this.handleClick}>提交</button>;
  }
}
```

```
class Submit extends Component {
  isSubmitting = false;

  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    console.log(this.isSubmitting);
  }

  render() {
    return <button onClick={this.handleClick}>提交</button>;
  }
}
```

例子

STATE

- 定义 State

```
class Submit extends Component {
  state = {
    isSubmitting: false,
  };

  render() {
    const { isSubmitting } = this.state;
    return <button disabled={isSubmitting}>提交</button>;
  }
}
```

```
class Submit extends Component {
  constructor(props) {
    super(props);
    this.state = {
      isSubmitting: props.isSubmitting,
    };
  }

  render() {
    const { isSubmitting } = this.state;
    return <button disabled={isSubmitting}>submit</button>;
  }
}
```

例子

修改 STATE

```
// never do this
this.state = newState;
```

```
this.setState(newState);
```

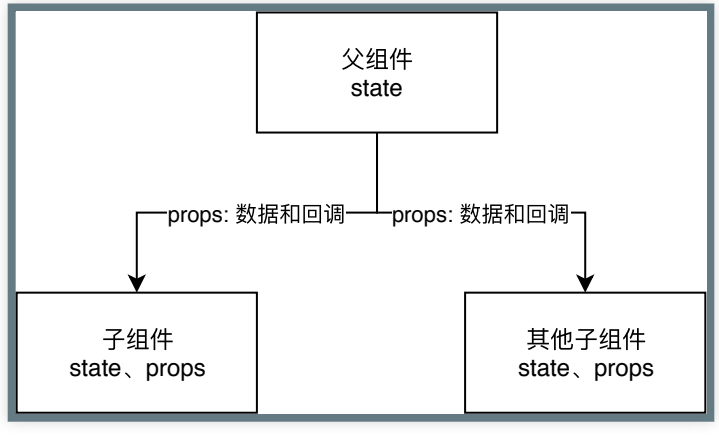
例子

STATE 的更新是异步的

```
this.setState({ isSubmitting: true });
if (comment.isBlocked) {
  this.setState({ buttonDisabled: false });
}
```

例子

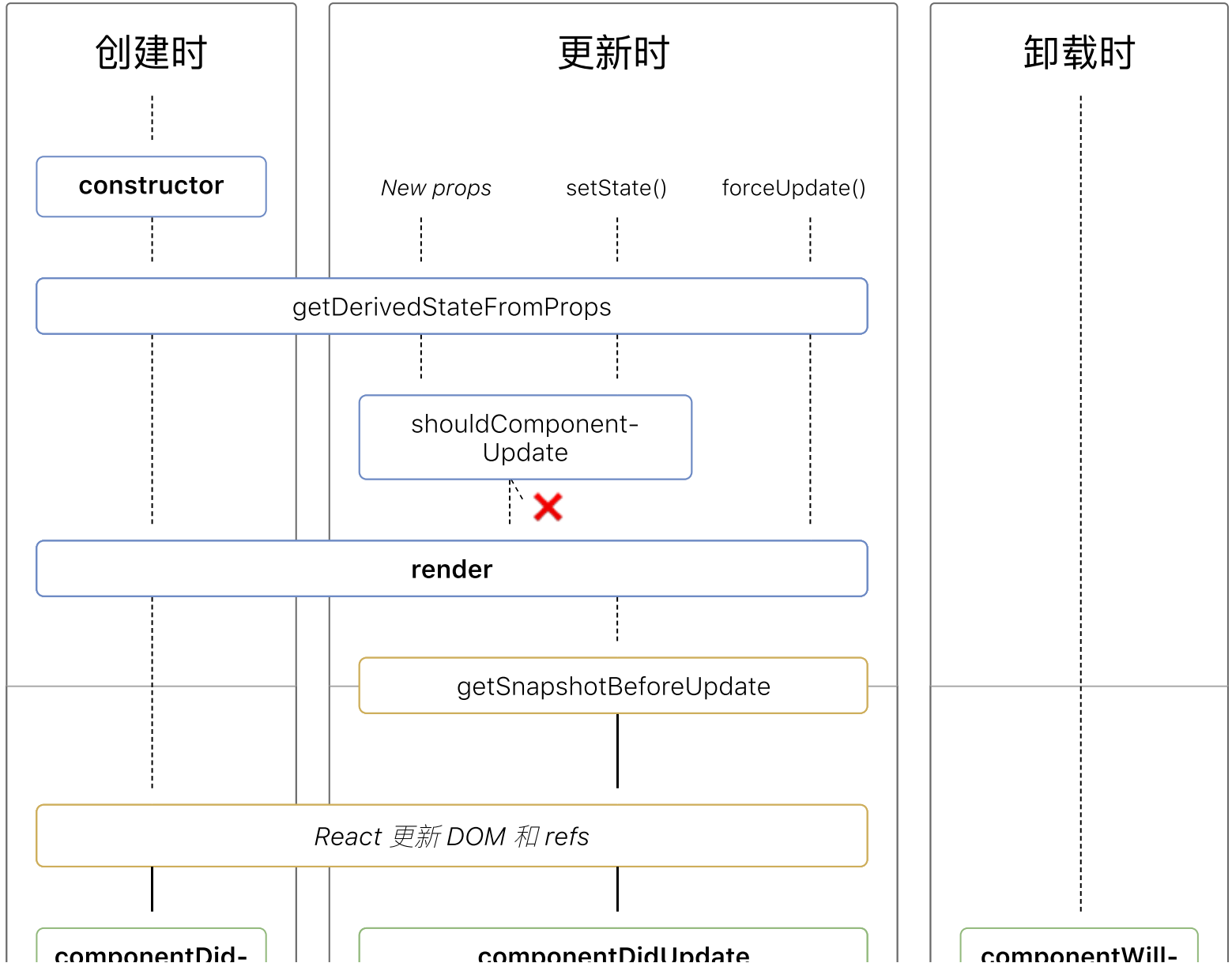
数据流


父组件
state
props: 数据和回调
props: 数据和回调
子组件
state、props
其他子组件
state、props

生命周期

显示更少的生命周期

GitHub



"Render阶段"

纯净且没有副作用。可能会被React暂停，中止或重新启动。

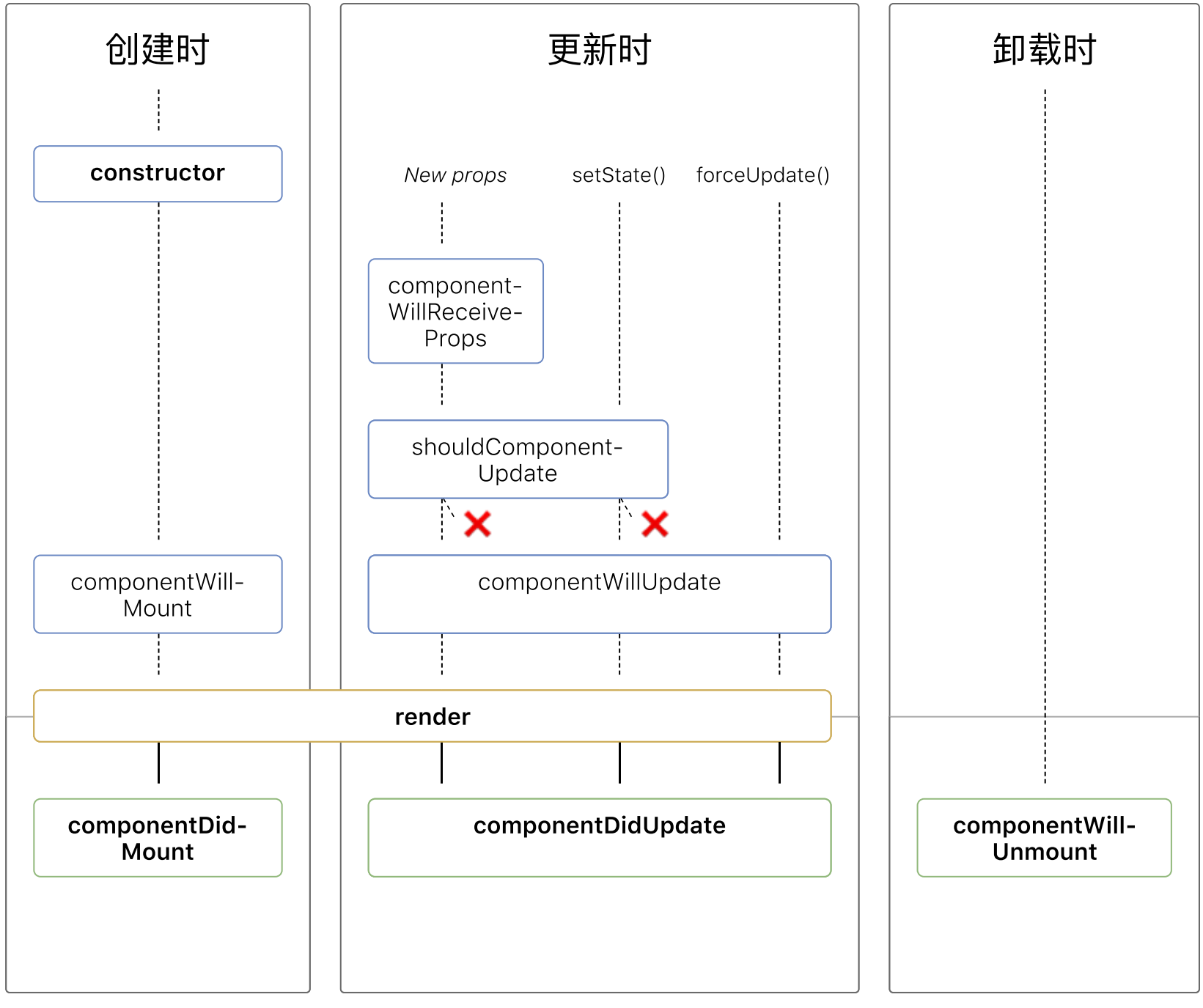
"Pre-commit阶段"

可以读取DOM。

"Commit阶段"

可以使用DOM，运行副作用，安排更新

V16.3 版本之前


创建时
更新时
卸载时
constructor
New props
setState()
forceUpdate()
component-WillReceive-Props
shouldComponent-Update
componentWill-Mount
componentWillUpdate
render
componentDid-Mount
componentDidUpdate
componentWill-Unmount