React Native

lifecycle, props, state, navigation, mobx…

Lifecycle

"Render Phase"

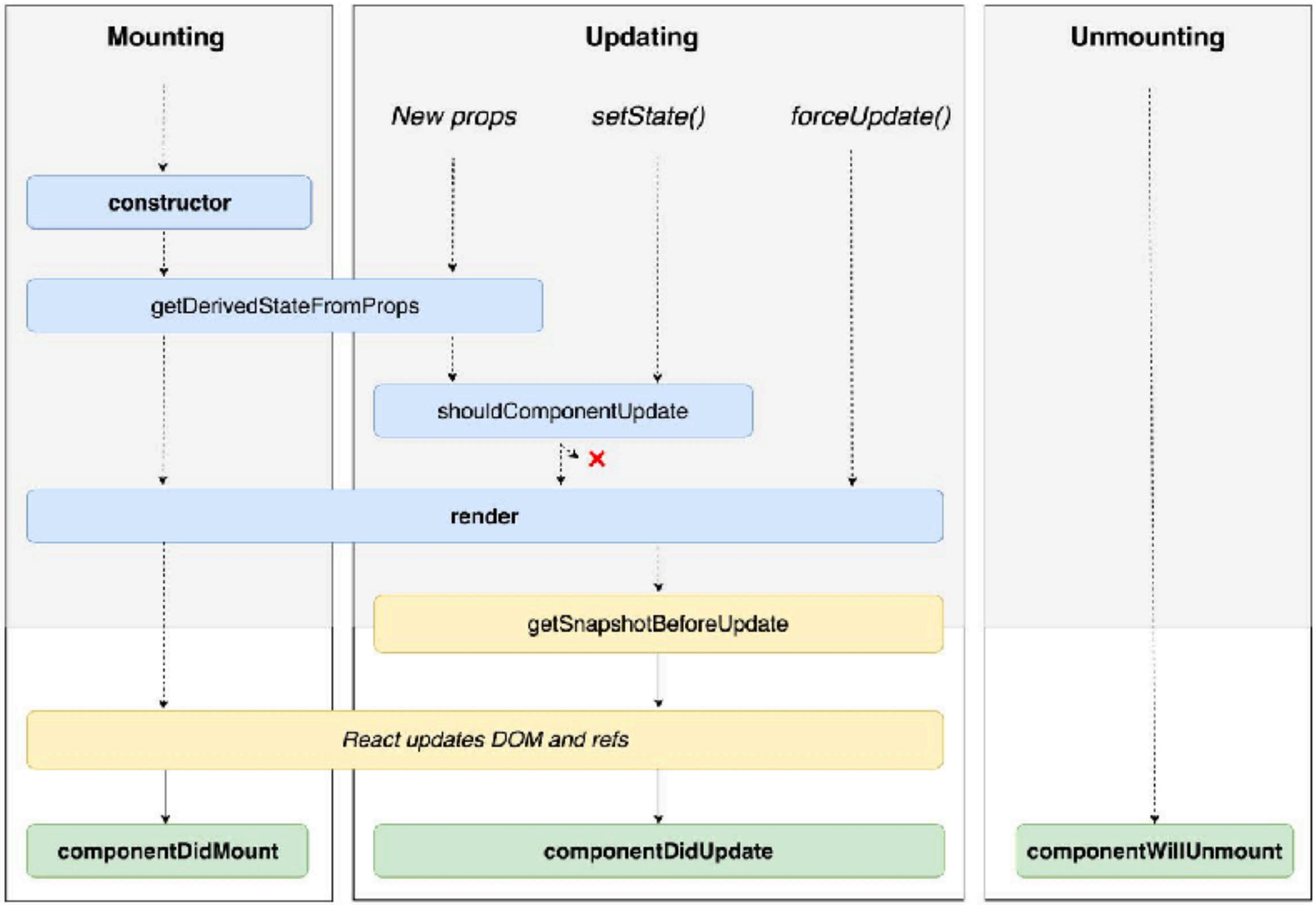
Pure and has no side effects. May be paused, aborted or restarted by React.

"Pre-Commit Phase"

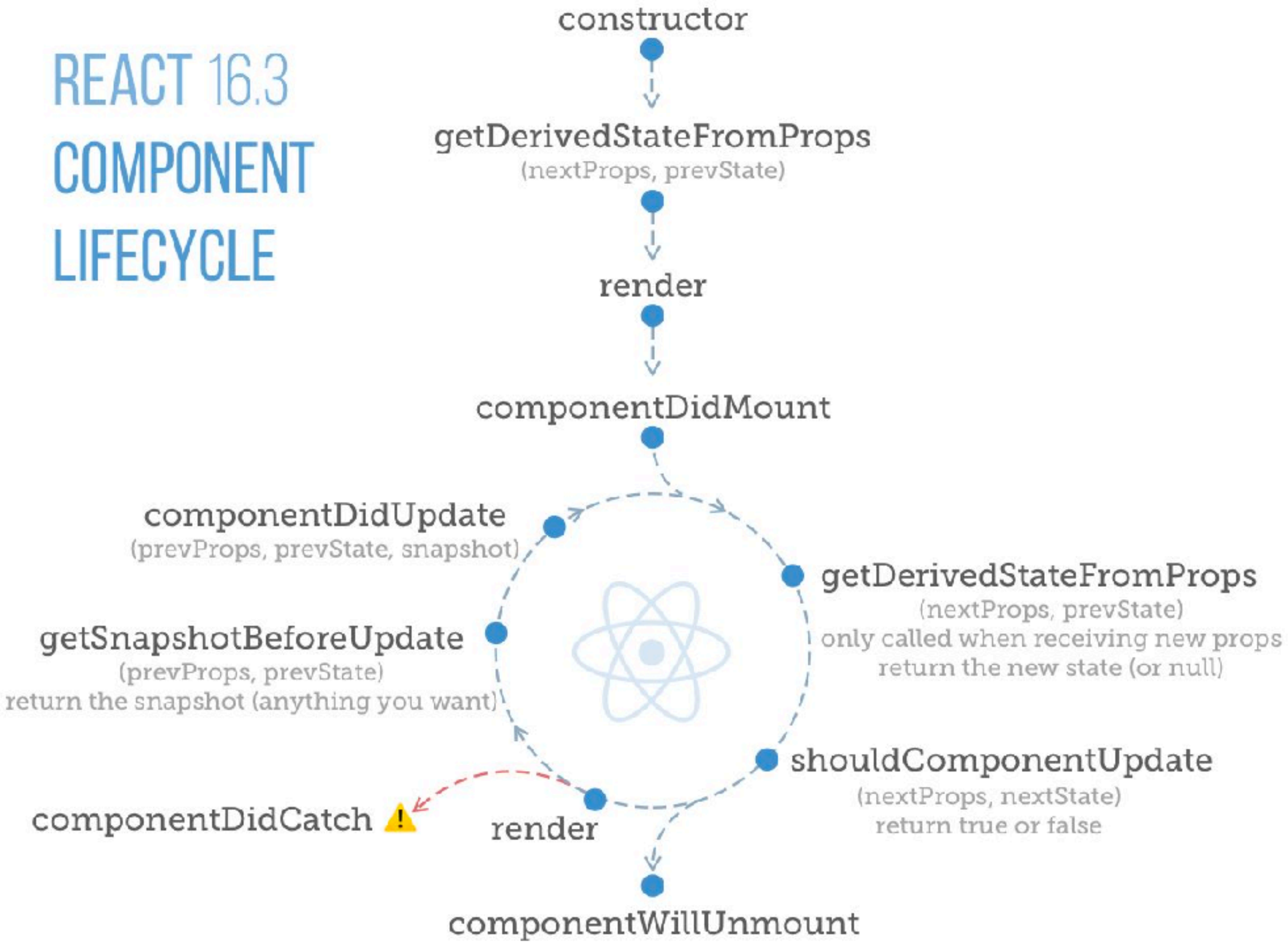
Can read the DOM.

"Commit Phase"

Can work with DOM, run side effects, schedule updates.



REACT 16.3 COMPONENT LIFECYCLE



Props

Props

- 컴포넌트의 부모 객체로부터 전달받는 속성
- 부모에서는 언제든 다른 값을 전달할 수 있음
  따라서 값을 변경하지 않고 가급적 수동적으로 처리함
- 필요시 state에 값을 복제한 후 사용

사용 예

부모 컴포넌트

```
render() {
    return <View>
        <MyButton label={ 'Submit' }/>
    </View>
}
```

MyButton.js

```
class MyButton extends React.Component {
    render() {
        return <View>
            <TouchableOpacity>
                <Text>{ this.props.label }</Text>
            </TouchableOpacity>
        </View>;
    }
}
```

Props로 함수 전달

부모 컴포넌트

```
render() {
    return <View>
        <MyButton
            label={ 'Submit' }
            onPress={ () => Alert.alert( 'Click!', '' ) }
        />
    </View>
}
```

MyButton.js

```
onPress = () => {
    if( this.props.onPress )
        this.props.onPress();
};
render() {
    return <View>
        <TouchableOpacity onPress={ this.onPress }>
            <Text>{ this.props.label }</Text>
        </TouchableOpacity>
    </View>;
}
```

props.children

부모 컴포넌트

```
return <View>
    <ImageButton onPress={ () => {} }>
        <Image src={{ uri: '...' } }/>
    </ImageButton>
</View>
```

ImageButton.js

```
return <View>
    <TouchableOpacity onPress={ this.onPress }>
        { this.props.children }
        {/*부모 객체에서 전달한 자식 노드가 그대로 표시된다*/}
    </TouchableOpacity>
</View>;
```

State

State

- 컴포넌트 내부의 상태를 관리
- 상태의 변화는 화면의 렌더에 즉시 반영됨
- 반드시 setState() 를 통해서만 업데이트

초기화, setState()

```
state = {
    seconds: 0,
};
componentDidMount() {
    setTimeout( () => {
        this.setState( { seconds: this.state.seconds + 1 } );
    }, 1000 );
}
render() {
    return (
        <View style={styles.container}>
            <Text>
                {/*state 가 변경될 때 마다 렌더에 자동으로 반영된다*/}
                앱이 열린지 { this.state.seconds } 초 지났습니다.
            </Text>
        </View>
    );
}
```

state에 따른 렌더 분기 #1

```
// state 초기값 설정
state = {
    activeMenu: 'inch',
};

render() {
    // state.activeMnu의 값에 따라 달라지는 뷰 렌더링
    if( this.state.activeMenu === 'inch' ) {
        return <View>
            <Text>인치 변환</Text>
        </View>;
    }
    else {
        return <View>
            <Text>센티 변환</Text>
        </View>;
    }
}
```

state에 따른 렌더 분기 #2

```
// state 초기값 설정
state = {
    activeMenu: 'inch',
};

render() {
    return <View>
        {/*state.activeMnu의 값에 따라 달라지는 뷰 렌더링*/}
        {this.state.activeMenu === 'inch' &&
            <Text>인치 변환</Text>
        }
        {this.state.activeMenu === 'cm' &&
            <Text>센티 변환</Text>
        }
    </View>
}
```

state에 따른 렌더 분기 #3

```
// state 초기값 설정
state = {
    activeMenu: 'inch',
};

renderText = () => {
    if( this.state.activeMenu === 'inch' )
        return <Text>인치 변환</Text>;
    else
        return <Text>센티 변환</Text>;
};

render() {
    return <View>
        { this.renderText() }
    </View>
}
```

InputText 에서의 state 문제

```
// state 초기값 설정
state = {
    value: 0,
};

render() {
    return <View>
        <TextInput
            style={{height: 40}}
            value={this.state.value}
            // state -> view 방향으로만 데이터 바인딩이 일어나기 때문에
            // 사용자가 view 를 조작해도 state 의 값이 다시 덮어써짐
            // 즉 readonly 같은 상태의 TextInput 이 됨
        />
    </View>
}
```

InputText 에서 setState() #1

```
// state 초기값 설정
state = {
    value: 0,
};

render() {
    return <View>
        <TextInput
            style={{height: 40}}
            value={this.state.value}
            onChangeText={ text => this.setState( { value: text } ) }
            // onChangeText 이벤트에서 state 를 업데이트 해줘야 함
        />
    </View>
}
```

InputText 에서 setState() #2

```
// state 초기값 설정
state = {
    value: 0,
};

updateText = text => {
    // setState 전 처리할 작업이 많다면
    text = text.trim();
    this.setState( { value: text } );
};

render() {
    return <View>
        <TextInput
            style={{height: 40}}
            value={this.state.value}
            onChangeText={ this.updateText }
        />
    </View>
}
```

setState 의 한계

```
// state 초기값 설정
state = {
    activeMenu: 'inch',
    value: 0,
};

update = () => {
    this.setState( { value: 1 } );
    // 일부 값만 업데이트 가능
};
```

setState 의 한계

```
// state 초기값 설정
state = {
    activeMenu: 'inch',
    userData: {
        name: 'John',
        email: 'john@example.com'
    }
};

update = () => {
    this.setState( {
        userData: {
            email: 'john@example.org'
        }
    } );
    // userData.name 값은 사라짐
};
```

setState 의 한계 - 해결책 #1

```
// state 초기값 설정
state = {
    activeMenu: 'inch',
    userData: {
        name: 'John',
        email: 'john@example.com'
    }
};

update = () => {
    this.setState( {
        userData: {
            ...this.state.userData,
            // 전개 연산자 spread operation 을 이용해 기존 값을 유지
            email: 'john@example.org'
            // 변경할 값만 덮어씀
        }
    } );
};
```

setState 의 한계 - 해결책 #2

```
// state 초기값 설정
state = {
    activeMenu: 'inch',
    userData: {
        name: 'John',
        email: 'john@example.com'
    }
};

update = () => {
    const newState = { ...this.state }; // state 복제
    newState.userData.email = 'john@example.org';
    this.setState( newState );
};
```

setState 의 한계 - 해결책 #3

...은 좀 더 뒷쪽 파트에서 다루어봅니다!

Navigation

Navigation

- 여러 페이지를 구성하는 앱을 만든다면 네비게이션이 필요
- 단순히 라우팅을 나누는 것 이상으로 사용자의 하드웨어 제어에 대한 대응이 필요

  iOS: Swipe Back 제스쳐로 페이지 이동
  Android: 하드웨어 Back 버튼으로 페이지 이동

주요 네비게이션 라이브러리

- React Navigation
  https://github.com/react-navigation/react-navigation

- React Native Navigation
  https://github.com/wix/react-native-navigation

- Native Navigation
  https://github.com/airbnb/native-navigation

- React Router
  https://github.com/ReactTraining/react-router

Navigator의 종류

- StackNavigator
- SwitchNavigator
- DrawerNavigator
- TabNavigator
- BottomTabNavigator
- MaterialBottomTabNavigator
- MaterialTopTabNavigator

설치

yarn add react-navigation
or
npm i --save react-navigation

Navigator 생성

```
const Navigator = createStackNavigator( {
    List: {
        screen: CocktailList,
        navigationOptions: {
            title: '칵테일 목록'
        }
    },
    Detail: {
        screen: CocktailDetail,
        navigationOptions: {
            title: '상세'
        }
    },
} );
```

Navigator 출력

```
<View style={styles.container}>
    <Navigator/>
</View>
```

페이지 이동

```
const params = {
    key: 'GinTonic'
};
this.props.navigation.navigate( 'Detail', params );
// 네비게이터 하위의 페이지는 navigation 이라는 prop 을 받게 되고
// navigate() 명령을 통해 페이지를 이동할 수 있다
```

Navigation Parameter

```
const key = this.props.navigation.state.params.key;
// Parameter 를 전달받을 경우 navigation.state.params 에서 확인할 수 있다.
// console.log( key ); // GinTonic
```

Mobx

Mobx 소개

- 반응형 프로그래밍 Reactivity Programming 의 일종
- 가장 간단하고 단순한 형태로 데이터의 상태 state 를 observable 과 observer 형태로 관리할 수 있게 한다
- mobx 단독으로도 사용할 수 있고 리액트를 위한 mobx-react 를 사용하면 더욱 편리

setState 의 불편함

```
this.setState( { a: 'b' } );
// to something
this.setState( { c: 'd' } );
// setState 는 비동기로 동작하기 때문에
// 둘 중 늦게 동작하는 setState가
// 다른 동작을 덮어쓸 가능성이 있다
```

setState 의 불편함

```
this.setState( { a: 'b' }, () => {
    // do something
    this.setState( { c: 'd' } );
} );
// setState 는 비동기로 동작하기 때문에
// 콜백을 제공한다.
// 하지만 콜백 지옥이 만들어질 우려가 있다.
```

observable + observer

```
class Store {
    @observable value = 'GinTonic';
}

@observer class App extends React.Component {
    store = new Store();

    render() {
        return <Text>{ this.store.value }</Text>
            ...
```

observable 업데이트

```
<Picker
    style={{ width: 300, height: 40 }}
    selectedValue={this.store.value}
    onValueChange={value => this.store.value = value}
>
    <Picker.Item label="진토닉" value="GinTonic"/>
    <Picker.Item label="AMF" value="AMF"/>
    <Picker.Item label="모히토" value="Mojito"/>
</Picker>
```

설치

yarn add mobx mobx-react
or
npm i --save mobx mobx-react