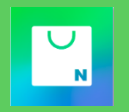
React Hooks 마법.

그리고 깔끔한 사용기



Naver 쇼핑 최효석

개발은…

재미있으신가요?

개발은…

때때로… 재미있습니다.


일어나
코딩
해야지

- 간결
- 로직 예상이 쉽고
- 읽기 쉽고
- 재활용이 쉬운

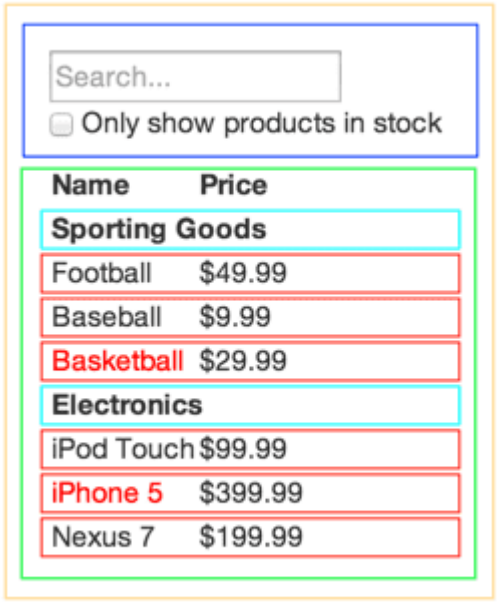
코드를 **쉽게!** 짜고 싶습니다.

React Hooks 한번 사용해 볼까?

React

- Facebook 주도의 오픈 소스
- MVC 중 view 를 처리하는 프레임워크
- 자체적인 state 를 관리하는 component 기반으로 개발

component 기반 개발

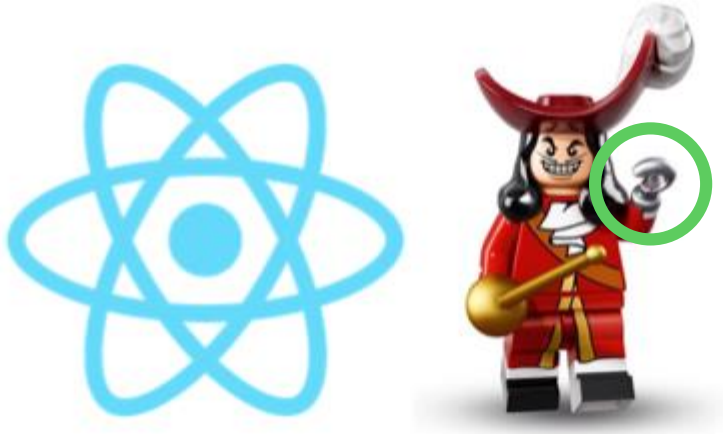


기본적으로 component 는

javascript class 를 활용

=〉 class component

React Hooks



* hook - 갈고리로 걸다

functional component 에서

class component 에서만 사용할 수 있었던

state, lifecycle feature 를 갈고리를 걸어 **사용**한다

React Hooks 마법

왜!

functional component 에서

state, lifecycle feature 사용?

- class component **복잡**
- functional component **간결**

왜 class component 가 복잡한가?

1. Stateful logic 재활용이 어렵다

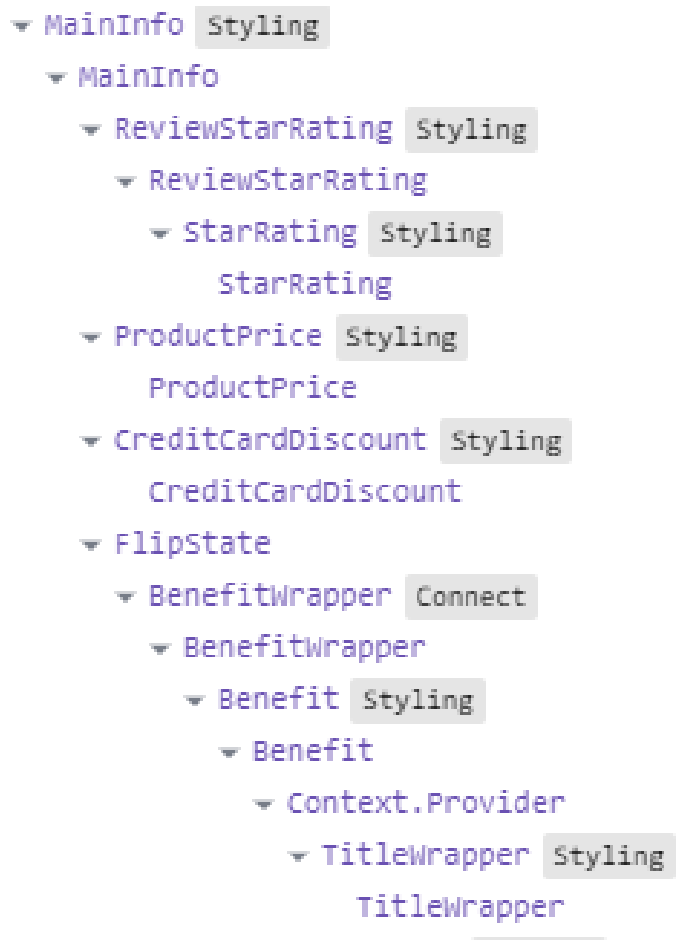
- higher order component (HOC)
- render props

Component hierarchy 변경

```
function styling(...styles) {
  return function wrapStyling(ComposedComponent) {
    class Styling extends React.Component {
      render() {
        return <ComposedComponent {...this.props} />
      }
    }
    return hoistStatics(Styling, ComposedComponent)
  }
}
```

```
export default styling(s)(Home)
```

DevTools 의| wrapper hell



2. lifecycle method 작성이 어렵다

1) lifecycle feature 간 로직 중복

```
class PhotoVideoList extends React.Component {

  async componentDidMount() {
    await this.fetchPhotoVideoItems()
  }

  async componentDidUpdate(prevProps) {
    if (this.props.photoVideoReviewIds !== prevProps.photoVideoReviewIds)
{
      await this.fetchPhotoVideoItems()
    }
  }

  fetchPhotoVideoItems = async () => {}
}
```

2. lifecycle method 작성이 어렵다

2) 1개 lifecycle feature에 lifecycle 에 수행되는 로직 몽땅

```
class EventDetail extends React.Component {

        componentDidMount() {
          setInitApplyProducts({})
          moveScrollByHistory()
          setApplyProducts({})
        }

        componentDidMount(prevProps) {
          if (prevProps.applyProductsState !== this.props.prevProps.applyProductsState) {
            moveScrollByHistory()
            setApplyProducts({})
          }
        }
}
```

2. lifecycle method 작성이 어렵다

3) 관련 있는 로직이 다른 lifecycle feature 에 위치

```
class FloatingBanner extends React.Component<OwnProps, OwnState> {

  componentDidMount() {
    window.addEventListener('orientationchange', this.handleOrientationChange)
  }

  componentDidUpdate() {

  }

  componentWillUnmount() {
    window.removeEventListener('orientationchange', this.handleOrientationChange)
  }
}
```

3. Javascript class 가 어렵다

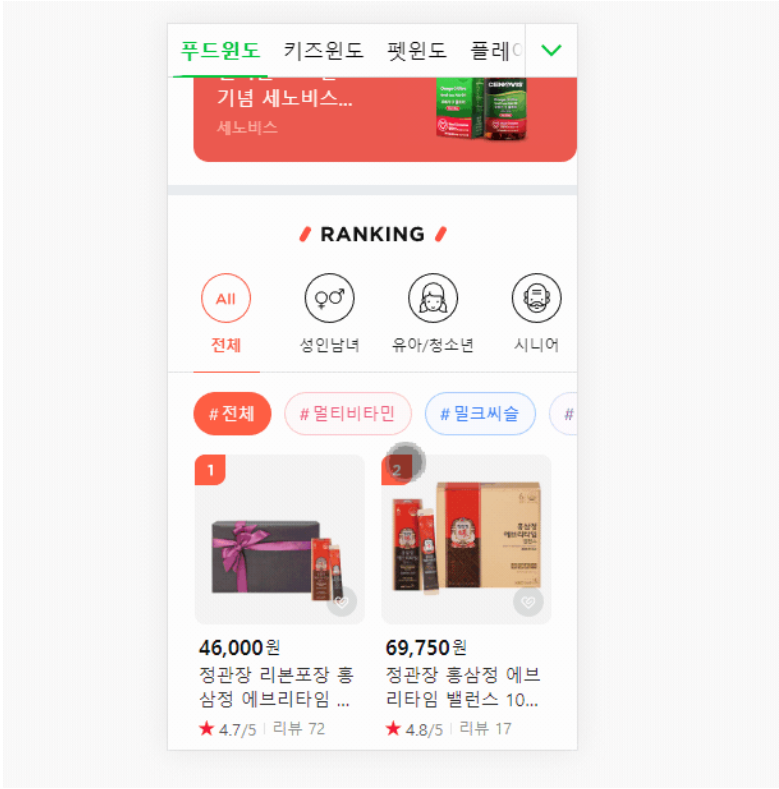
- 개념이 어렵다
  - this, bind event handler, extends, prototype 등

- 쉽게 사용하기 위해 여러 plugin 필요

왜 functional component 가 간결한가?

1. Stateful logic 재활용이 쉽다

- custom hooks




```
const ProductBenefitMenus = () => {
  useUISmoothScroll()
  return (
    <div />
  )
}
```

```
const HomeFilterOption = () => {
  useUISmoothScroll()
  return (
    <div />
  )
}
```

2. lifecycle method 작성이 쉽다

1) lifecycle feature 간 로직 중복 없음

class component

```
class PhotoVideoList extends React.Component {
fetchPhotoVideoItems = async () => {}

async componentDidMount() {
  await this.fetchPhotoVideoItems()
}

async componentDidUpdate(prevProps) {
  if (this.props.photoVideoReviewIds !==
prevProps.photoVideoReviewIds) {
    await this.fetchPhotoVideoItems()
  }
}
}
```

functional component

```
const PhotoVideoList : React.FC => {


useEffect(() => {
  const fetchPhotoVideoItems = async () => {}
  fetchPhotoVideoItems()
}, [photoVideoReviewIds])
}
```

2. lifecycle method 작성이 쉽다

2) 다른 logic 은 다른 lifecycle feature 에 설정

class component

```
class EventDetail extends React.Component {
componentDidMount() {
   setInitApplyProducts({})
   moveScrollByHistory()
   setApplyProducts({})
}

componentDidUpdate(prevProps) {
   if (prevProps.applyProductsState !==
this.props.applyProductsState) {
    moveScrollByHistory()
    setApplyProducts({})
   }
}
```

functional component

```
const EventDetail : React.FC => {
useEffect(() => {
  setInitApplyProducts({})
}, [])

useEffect(() => {
   setApplyProducts({})
  }, [applyProductsState])

useEffect(() => {
    moveScrollByHistory()
  }, [applyProductsState])
```

2. lifecycle method 작성이 쉽다

3) 관련 있는 로직은 같은 lifecycle feature 에 설정

class component

```
class FloatingBanner extends React.Component {
 handleChange = () => {}

 componentDidMount() {
   window.addEventListener('orientationchange', this. handleChange)
 }

  componentDidUpdate() {}

 componentWillUnmount() {
   window.removeEventListener('orientationchange', this.handleChange)
 }
}
```

functional component

```
const FloatingBanner: React.FC = () => {

  useEffect(() => {
    const handleChange = () => {}
    window.addEventListener('orientationchange', handleChange)
    return () => {
      window.removeEventListener('orientationchange', handleChange)}
  }, [])
}
```

3. Javascript class 사용 안함

One more thing…

useContext

class component

```
class Statistics extends React.Component {

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

```
class Statistics extends React.Component {
  render() {
    return (
      <VerticalContext.Consumer>
        {({ subVertical }) => (
          <div className={subVertical}>
          </div>
        )}
      </VerticalContext.Consumer>
    )
  }
}
```

functional component

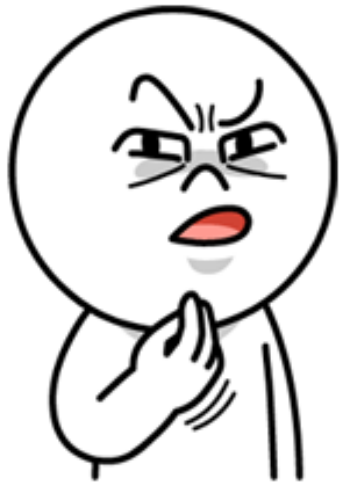
```
const Statistics: React.FC = () => {

  return (
    <div>
    </div>
  )
}
```

```
const Statistics: React.FC = () => {
  const { pathname} = useContext(VerticalContext)
  return (
    <div className={subVertical}>
    </div>
  )
}
```

그리고 깔끔한 사용기

functional component 는 다 좋은가?



lifecycle feature 수행 방식 비교 (class component vs. functional component)

class component

```
class CategoryList extends React.Component {
  constructor() {
      this.state = {name: 'Mary'}
  }
  componentDidMount() {
  }

  shouldComponentUpdate () {

  }
  componentDidUpdate() {
  }

  componentWillUnMount() {
  }

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

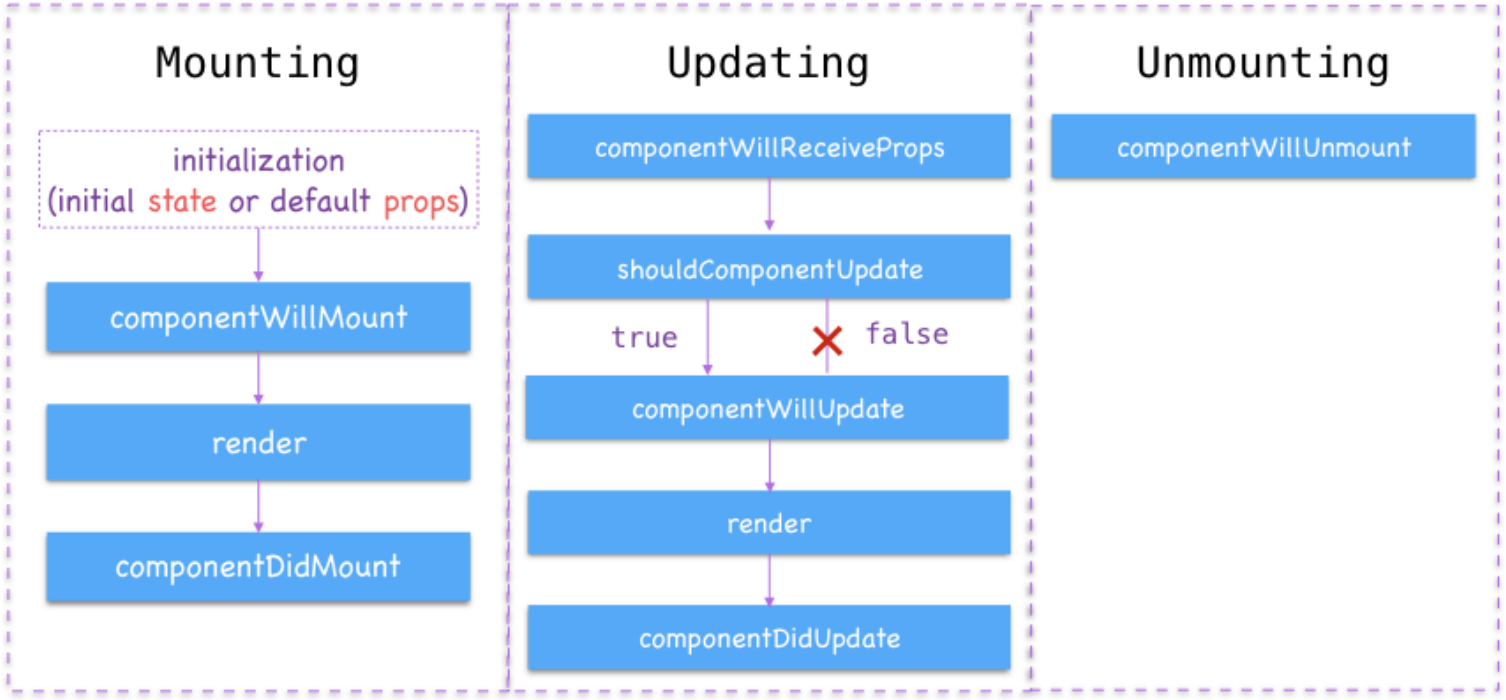
functional component

```
const CategoryList: React.FC = () => {
 const [name, setName] = useState('Mary')
  useEffect(() => {
  })

return (
      <div>
      </div>
    )

}
```

class component



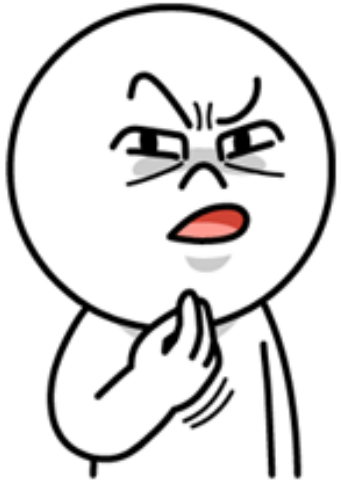
- lifecycle feature 는 class method
- React 가 필요할 때 class method 를 호출
- 특정 class method 호출이 다른 method 호출 영향 X

functional component

```
const CategoryList: React.FC = () => {
 const [name, setName] = useState('Mary')
  useEffect(() => {
  })

return (
      <div>
      </div>
    )

}
```



functional component 가 실행될 때마다

functional component 안의
모든 React Hooks 관련 함수들이 실행됨

1. Hooks 실행순서는

모든 render 에서 동일해야 함

- 동일하지 않으면 runtime 잘못 작동

정상 동작

```
// ------------
// 첫번째 render
// ------------
const [name, setName] = useState('Mary')
useEffect(persistForm)
const [surname, setSurname] = useState('Poppins')
useEffect(updateTitle)
```

// 1. 'Mary' 로 name state 초기화
// 2. persistForm 로 render 후에 실행할 effect 추가
// 3. 'Poppins' 로 surName state 초기화
// 4. updateTitle 로 render 후에 실행할 effect 추가

```
// ------------
// 두번째 render
// ------------
const [name, setName] = useState('Mary')
useEffect(persistForm)
const [surname, setSurname] = useState( ' Poppins ' )
useEffect(updateTitle)
```

// 1. name state 읽기 (argument 로 ‘Mary’ 는 무시)
// 2. persistForm 로 render 후에 실행할 effect 교체
// 3. surName state 읽기 (argument 로 ‘Poppins’ 는 무시)
// 4. updateTitle 로 render 후에 실행할 effect 교체

runtime 시 잘못 작동

```
// ------------
// 첫번째 render
// ------------
const [name, setName] = useState('Mary')
name && useEffect(persistForm)
const [surname, setSurname] = useState('Poppins')
useEffect(updateTitle)
setName(null)

// ------------
// 두번째 render
// ------------
const [name, setName] = useState('Mary')
name && useEffect(persistForm)
const [surname, setSurname] = useState( ' Poppins ' )
useEffect(updateTitle)
```

```
// 1. 'Mary' 로 name state 초기화
// 2. persistForm 로 render 후에 실행할 effect 추가
// 3. 'Poppins' 로 surName state 초기화
// 4. updateTitle 로 render 후에 실행할 effect 추가

// 1. name state 읽기 (argument 로 'Mary' 는 무시)

// 2. persistForm 로 render 후에 실행할 effect 교체 (실패)
// 3. 'Poppins' 로 surName state 초기화 (실패)
```

Help!

- eslint 설정
  - react-hooks/rules-of-hooks

```
C:\dev\workspace\shopping-web\packages\shopping\src\mobile\components\Home\Play\Fishing\TabList.tsx
  26:5  warning  React Hook "useEffect" is called conditionally. React Hooks must be called in the exact s
ooks/rules-of-hooks

x 1 problem (0 errors, 1 warning)

 @ ./src/mobile/components/Home/Play/Fishing/Home.tsx 213:0-32 560:28-35
 @ ./src/mobile/routes/home/play/index.tsx
 @ ./src/mobile/routes/index.tsx
 @ ./src/common/universalRouter.ts
 @ ./src/common/clientEntry.tsx
 @ multi C:/dev/workspace/shopping-web/config/polyfills.ts webpack-hot-middleware/client?reload=true&path=
mmon/clientEntry.tsx
```

2. useEffect 잘 사용하기

useEffect(effect, list of dependencies)

```
const FloatingBanner: React.FC = () => {

  useEffect(() => {
    const handleChange = () => {}
    window.addEventListener('orientationchange', handleChange)
    return () => {
      window.removeEventListener('orientationchange', handleChange)}
  }, [])
}
```

- render 이후에 실행할 effect 를 생성

React 는

- 데이터 조회
- 이벤트 핸들러 등록
- DOM 조작
- timer
- logging

작업을 **언제** 수행해야 할까?

Element 를 브라우저에 그린 이후 (render 이후)

React 는

- 데이터 조회
- 이벤트 핸들러 등록
- DOM 조작
- timer
- Logging

작업을 **render 이후** 에 수행해야 한다.

main task 인 render 에 포함되지 않음

=〉(side) **effect** 라고 부름

useEffect(effect, list of dependencies)

```
const FloatingBanner: React.FC = () => {

  useEffect(() => {
    const handleChange = () => {}
    window.addEventListener('orientationchange', handleChange)
    return () => {
      window.removeEventListener('orientationchange', handleChange)}
  }, [])
}
```

- render 이후에 실행할 effect 를 생성

문제!!

functional component 실행해서 **render 할 때 마다**

useEffect 가 실행

⇒ Effect 가 생성

⇒ render 이후에 effect 가 실행

```
const FloatingBanner: React.FC = () => {
    useEffect(() => {
        const handleOrientationChange = () => {}
       window.addEventListener('orientationchange', handleOrientationChange)
       return () => {
         window.removeEventListener('orientationchange', handleOrientationChange)}
    })
  return (<div></div>)
}
```

```
const PhotoVideoList : React.FC => {
       useEffect(() => {
          const fetchPhotoVideoItems = async () => {}
          fetchPhotoVideoItems()
       })
   return (<div></div>)
}
```

render 이후

내가 원할 때만 effect 를 실행할 수 있을까?

원할 때만 effect 를 생성

useEffect(effect, list of depedencies)

** list of dependencies

이전 render 의 list of dependencies
현재 render 의 list of dependencies
item 이 다를 때
effect 를 생성

첫번째 render 이후에만 effect 를 실행하고 싶어요!

useEffect(effect, list of depedencies)

useEffect(() => {}, [])

```
const FloatingBanner: React.FC = () => {
  const handleOrientationChange = () => {}

  useEffect(() => {
    window.addEventListener('orientationchange', handleOrientationChange)
    return () => {
      window.removeEventListener('orientationchange', handleOrientationChange)    }
  }, [])
}
```

특정 props, state 가 변경됐을때만 effect 실행하고 싶어요!

useEffect(effect, list of depedencies)

useEffect(() => {}, [props1, props2, state1, state2])

```
const PhotoVideoList : React.FC = ({photoVideoReviewIds}) => {

  useEffect(() => {

    const fetchPhotoVideoItems = async () => { }

}, [photoVideoReviewIds]);
}
```

list of dependencies 사용 시 주의할 점

list of dependencies 에는 effect 에서 사용하는 모든 props, state 가 포함되어야 함

```
const Example = ({ someProp, anotherProp }) => {
  const fetch = () => {
    fetchItem(someProp, anotherProp)
  }

  useEffect(() => {
    fetch()
  }, [someProp])
}
```

Help!

- eslint 설정
  - react-hooks/exhaustive-deps

```
C:\dev\workspace\shopping-web\packages\shopping\src\mobile\components\Home\Play\Fishing\TabList.tsx
  36:6  warning  React Hook React.useEffect has a missing dependency: 'handleScroll'. Either include it or
 remove the dependency array  react-hooks/exhaustive-deps

x 1 problem (0 errors, 1 warning)
  0 errors and 1 warning potentially fixable with the `--fix` option.

 @ ./src/mobile/components/Booking/Booking.tsx 34:0-66 150:27-34
 @ ./src/mobile/components/Booking/index.tsx
 @ ./src/mobile/routes/booking/index.tsx
 @ ./src/mobile/routes/index.tsx
 @ ./src/common/universalRouter.ts
 @ ./src/common/clientEntry.tsx
 @ multi C:/dev/workspace/shopping-web/config/polyfills.ts webpack-hot-middleware/client?reload=true&path=
http://localhost:4001/__webpack_hmr ./src/common/clientEntry.tsx
```

React Hooks 마법

- 간결
- 로직 예상이 쉽고,
- 읽기 쉽고,
- 재활용이 쉬운

그리고 깔끔한 사용기

- Hooks 순서
- useEffect

- Eslint 의 도움을 받자!

React Hooks 한번 사용해 볼까?

Q & A

못다한
Q & A

- React Context/state 와 Redux 의 차이는 무엇인가?
- React Context/state 로 Redux 를 완벽하게 대체가능한가?

`React Context/state` vs. `Redux` - 1

`React context/state` 로 대체 가능/불가한 Redux 기능

대체 가능 기능

Redux	React Context/State
action creator / dispatch / reducer	useReducer
`react-redux` 의 connect	useContext

대체 불가 기능

Redux	React Context/State
store	N/A

https://reactjs.org/docs/context.html
https://reactjs.org/docs/hooks-reference.html#usereducer
https://reactjs.org/docs/hooks-reference.html#usecontext

`React context/state` 와 `Redux` 의 차이

- 데이터 저장소

Redux

- 단일 저장소
  - `combineReducers` 에 argument 로 전달된 `reducers` 기준으로 단일 저장소 하위 state tree 생성
- React component 바깥에서 데이터 저장소 접근/설정 가능

React context/state

- 복수 저장소
  - `Context.Provider` 별로 데이터 저장소 생성
- React component 내부에서만 데이터 저장소 접근/설정 가능

`React Context/state` vs. `Redux` - 2

Redux 와 React Context/state 의 장단점

Redux

- 장점
  - server 의 state 를 serialize 해서 client 의 state 에 hydrate 하기가 쉬움
    - https://redux.js.org/introduction/three-principles#single-source-of-truth
  - 모든 dispatch 호출/reducer 호출에 일괄 작업이 필요 시, middleware 적용이 쉬움
    - https://redux.js.org/advanced/middleware
- 단점
  - 사용과 개념 이해가 React Context 에 비해 어려움
  - global state 관리를 위해 server/client 에서 redux 관련 library 가 추가로 실행
    - `redux`, `react-redux`

React Context/state

- 장점
  - `useContext`, `useReducer` React Hooks API 활용시, 사용이 Redux 에 비해 간단
  - global state 관리를 위해 server/client 에서 추가 실행할 library 가 없음
- 단점
  - server 의 state 를 serialize 해서 client 의 state 에 hydrate 하기가 까다로움
  - 모든 dispatch 호출/reducer 호출에 일괄 작업이 필요시, middleware 적용이 까다로움

Redux 와 React Context/state 는 각각 언제 사용할까?

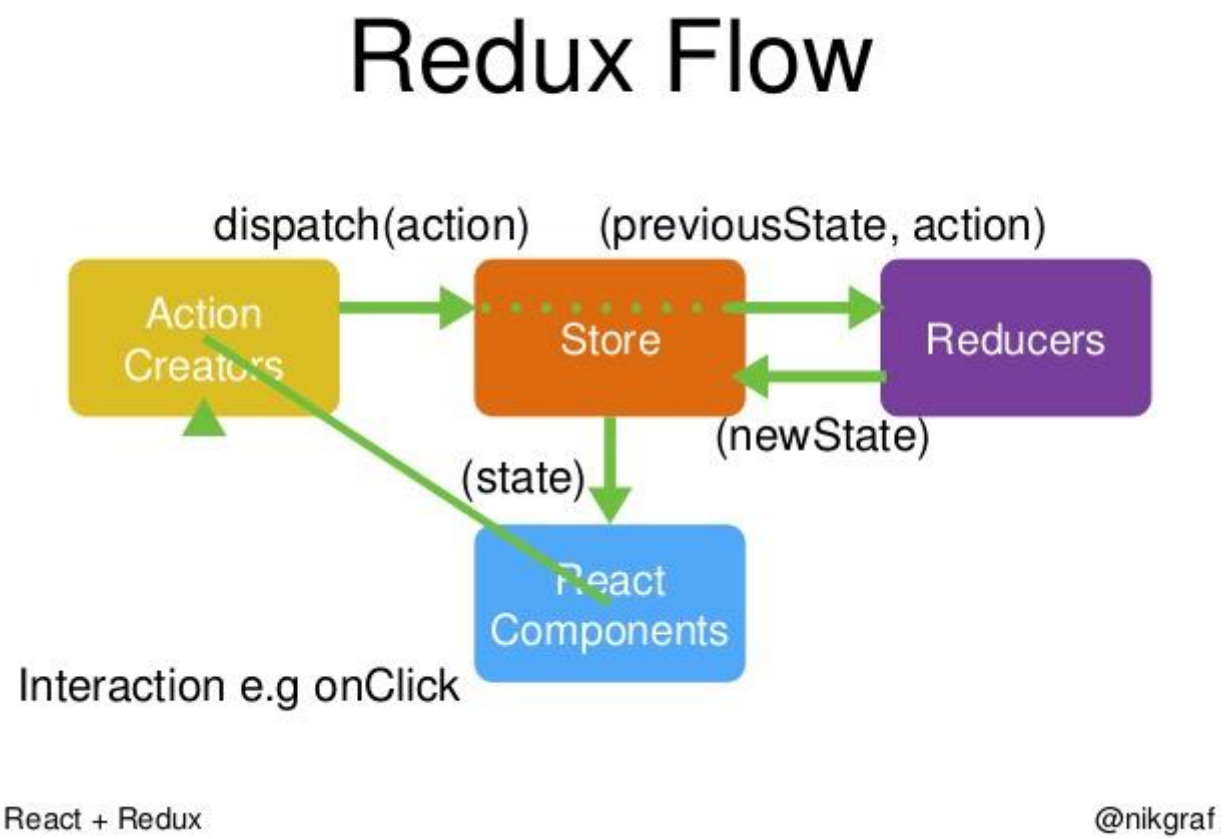
Redux

- server-side rendering 와 client-side rendering 시 함께 설정하는 global state

React Context/state

- client-side rendering 시에만 설정하는 global state
- 한개 페이지의 복수개 component 에서 동일 interface 의 global state 를 설정해야 할 때
  - 동일 페이지에서 복수개의 InfiniteGrid component 를 설정해야 할 때

참고) React 에서 Redux 사용 구조



발표 들어주셔서 감사합니다!