잘못된 리액트 훅의 사용을 미리 찾아보자!

리액트 훅의 엄밀한 의미구조 정의하기

이재호, 안중원, 이광근

2024 SIGPL 여름학교

ROPAS@SNU

리액트와 훅

리액트

- 사용자와 상호작용하는 동적인 웹 사이트(프로그램)를 만들고 싶다.
- 그런데 상태를 일일이 관리하는 것은 어렵다!
  - 예를 들어, 상태가 바뀌면 화면을 다시 그리기

리액트

- 사용자와 상호작용하는 동적인 웹 사이트(프로그램)를 만들고 싶다.
- 그런데 상태를 일일이 관리하는 것은 어렵다!
  - 예를 들어, 상태가 바뀌면 화면을 다시 그리기
- **리액트**에게 상태를 관리하도록 맡기자.

화면조각(element)과 컴포넌트(component)

화면의 단위를 화면조각이라고 부름

- 함수가 작업의 단위인 것처럼

화면조각(element)과 컴포넌트(component)

화면의 단위를 화면조각이라고 부름

- 함수가 작업의 단위인 것처럼

화면조각의 사양을 나타내는 JS 함수를 컴포넌트라고 부름

```
function Profile() {
  return (
    <img src="https://ropas.snu.ac.kr/~jhlee/jhlee_photo.jpg"
alt="이재호" />
  );
}
```

훅(hook)

사양을 읽는 것 = 컴포넌트를 실행하는 것

- 사양을 읽는 것만으로 부수효과(side-effect)가 발생하면 문제
- 컴포넌트는 순수한(pure) 함수여야 함

훅(hook)

사양을 읽는 것 = 컴포넌트를 실행하는 것

- 사양을 읽는 것만으로 부수효과(side-effect)가 발생하면 문제
- 컴포넌트는 순수한(pure) 함수여야 함

화면조각이 사용자와 상호작용하기 위해서는 부수효과가 필요

- 훅: 리액트가 통제하는 부수효과를 정의하는 함수

```
function Counter() {
  const [x, setX] = useState(0);
  return (
    <>
      <button onClick={() ⇒ setX(x+1)}>+</button>
      <h1>{x}</h1>
    </>
  );
}
```

두 가지 훅

useState 상태를 담고 바꾸는 훅

```
const [state, setState] = useState(initialState)
```

useEffect 상태가 바뀌거나 화면이 다시 그려질 때마다 실행되는 훅

```
useEffect(() => {
  // do something after every render
}, [])
```

문제

훅의 의미구조?

리액트가 훅을 '알아서' 잘 처리하기 때문에 편리하지만 이해하기 어려움

- 사람의 말과 주의사항만으로 설명되어 있음
  - 실제로 어떻게 작동하는지 알기 어려움
- 리액트는 사실상 또 다른 언어
  - 자바스크립트만으로 프로그래밍할 때와는 다른 규칙이 적용됨

훅의 의미구조?

Caveats

- `useEffect` is a Hook, so you can only call it **at the top level of your component** or your own Hooks. You can't call it inside loops or conditions. If you need that, extract a new component and move the state into it.
- If you're **not trying to synchronize with some external system,** you probably don't need an Effect.
- When Strict Mode is on, React will **run one extra development-only setup+cleanup cycle** before the first real setup. This is a stress-test that ensures that your cleanup logic "mirrors" your setup logic and that it stops or undoes whatever the setup is doing. If this causes a problem, implement the cleanup function.
- If some of your dependencies are objects or functions defined inside the component, there is a risk that they will **cause the Effect to re-run more often than needed.** To fix this, remove unnecessary object and function dependencies. You can also extract state updates and non-reactive logic outside of your Effect.
- If your Effect wasn't caused by an interaction (like a click), React will generally let the browser **paint the updated screen first before running your Effect.** If your Effect is doing something visual (for example, positioning a tooltip), and the delay is noticeable (for example, it flickers), replace `useEffect` with `useLayoutEffect`.
- If your Effect is caused by an interaction (like a click), **React may run your Effect before the browser paints the updated screen**. This ensures that the result of the Effect can be observed by the event system. Usually, this works as expected. However, if you must defer the work until after paint, such as an `alert()`, you can use `setTimeout`. See reactwg/react-18/128 for more information.
- Even if your Effect was caused by an interaction (like a click), **React may allow the browser to repaint the screen before processing the state updates inside your Effect.** Usually, this works as expected. However, if you must block the browser from repainting the screen, you need to replace `useEffect` with `useLayoutEffect`.
- Effects **only run on the client.** They don't run during server rendering.

리액트 훅의 잘못된 사용 미리 찾기

"훅"의 잘못된 사용의 대표적인 예시:

1. 상태가 바뀌어도 화면이 다시 그려지지 않는 경우
2. **사용자의 상호작용 없이 자기 혼자 여러 번 그려지는 경우**

리액트 훅의 잘못된 사용 미리 찾기

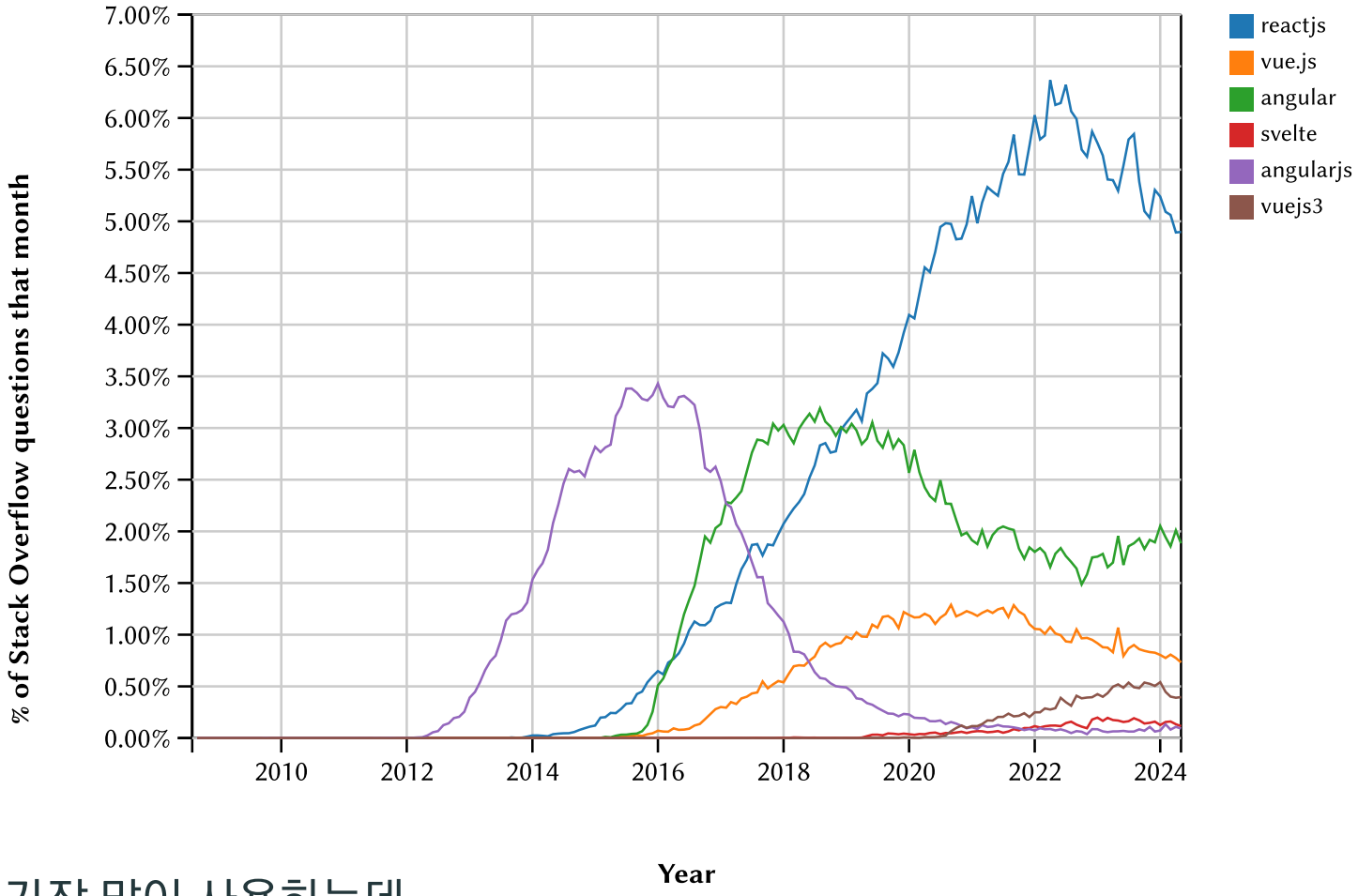
“훅”의 잘못된 사용의 대표적인 예시:

1. 상태가 바뀌어도 화면이 다시 그려지지 않는 경우
2. **사용자의 상호작용 없이 자기 혼자 여러 번 그려지는 경우**

지금은 모양만 보고 흔한 패턴을 경고하는 수준

- 의미구조를 정의해야 명확히 문제를 정의할 수 있다!

리액트 훅 분석의 배경

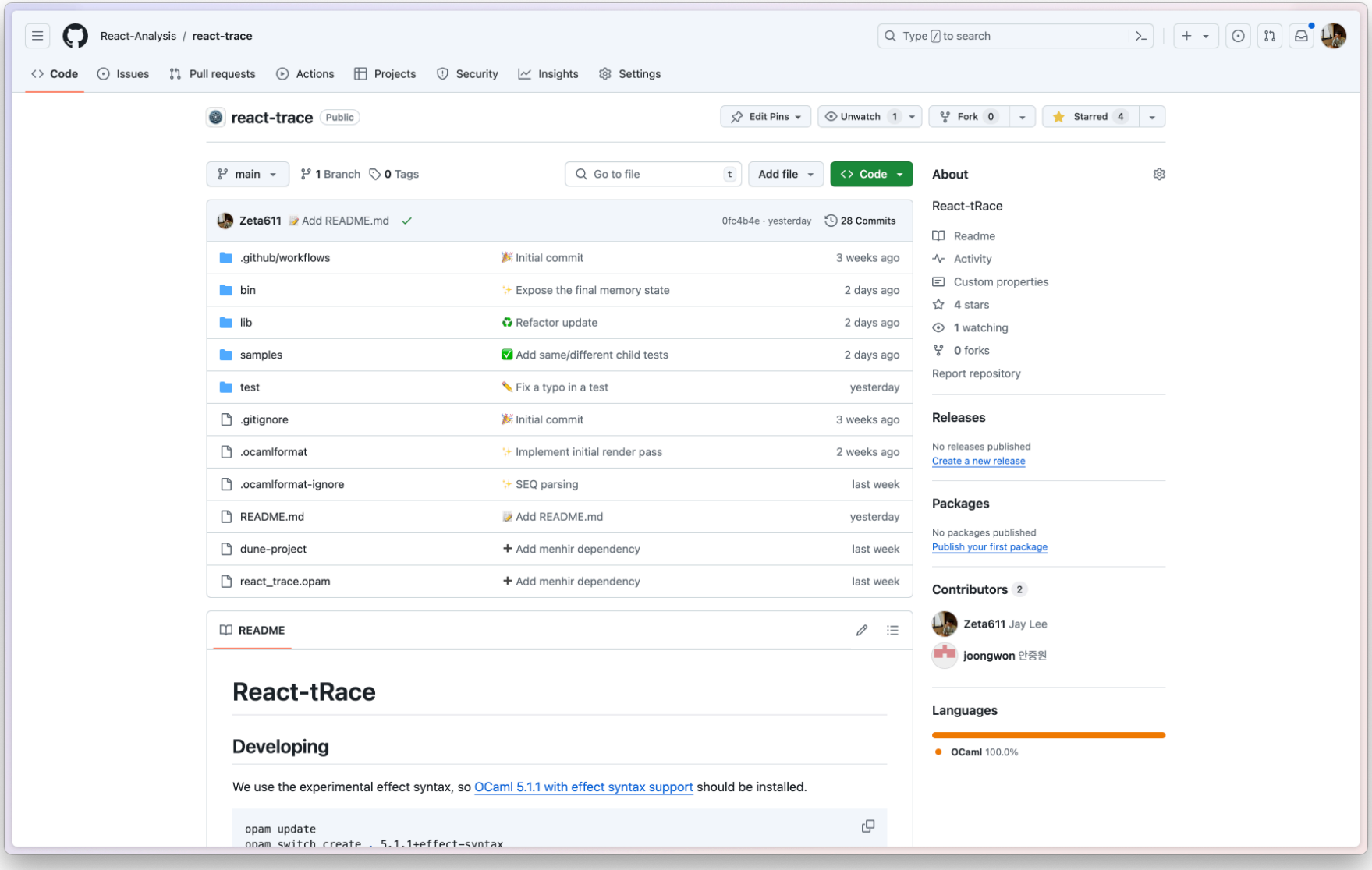


- 사람들이 가장 많이 사용하는데
- 잘 이해하고 있지 못하고
- 실수하기도 쉽기 때문에

→ 리액트 훅의 **정리된 의미구조**와 **실수를 찾아주는 분석**이 필요하다!

실행기 구현

실행기 (reference interpreter) 데모



`github.com/React-Analysis/react-trace`

실제 리액트와 비교하는 테스트

```
ReacttRace on  main [$] via 🐫 v5.1.1+effect-syntax (*ReacttRace)
λ dune runtest
Testing `Interpreter'.
This run has ID `SZFP0LL0'.

  [OK]          parse          0   unit.
  [OK]          parse          1   true.
  [OK]          parse          2   false.
  [OK]          parse          3   int.
  [OK]          parse          4   var.
  [OK]          parse          5   view.
  [OK]          parse          6   open cond.
  [OK]          parse          7   closed cond.
  [OK]          parse          8   fn.
  [OK]          parse          9   app.
  [OK]          parse         10   let.
  [OK]          parse         11   stt.
  [OK]          parse         12   eff.
  [OK]          parse         13   seq.
  [OK]          parse         14   op.
  [OK]          steps          0   No side effect should step one time.
  [OK]          steps          1   Set in body should not terminate.
  [OK]          steps          2   Guarded set in body should step one time.
  [OK]          steps          3   Set in effect should step one time.
  [OK]          steps          4   Set in effect should step two times.
  [OK]          steps          5   Set in effect should step indefintely.
  [OK]          steps          6   Guarded set in effect should step two times.
  [OK]          steps          7   Guarded set in effect should step five times.
  [OK]          steps          8   Set in effect with arg should step one time.
  [OK]          steps          9   Set in effect with arg should step two times.
  [OK]          steps         10   Set passed to child should step two times.
  [OK]          steps         11   Set passed to child should step indefintely.
  [OK]          steps         12   Set in effect twice should step one time.
  [OK]          steps         13   Set in removed child should step two times.
  [OK]          steps         14   Same child gets persisted.
  [OK]          steps         15   New child steps again.

Full test results in `~/Developer/ReacttRace/_build/default/test/_build/_tests/Interpreter'.
Test Successful in 0.010s. 31 tests run.
```

포스터 발표에서 뵙겠습니다!