Context API와 JavaScript 기술 면접

Closure?

Closure?

- Closure: 폐쇄

Closure?

- Closure: **폐쇄**
- MDN 공식문서:

> 클로저는 주변 상태(어휘적 환경)에 대한 참조와 함께 묶인(포함된) 함수의 조합입니다. 즉, 클로저는 내부 함수에서 **외부 함수의 범위에 대한 접근**을 제공합니다.

```
function init() {
  var name = "Mozilla";
  function displayName() {
    console.log(name); // 부모 함수에서 선언된 변수를 사용한다.
  }
  displayName();
}
```

Closure?

- Closure: **폐쇄**
- MDN 공식문서:

> 클로저는 주변 상태(어휘적 환경)에 대한 참조와 함께 묶인(포함된) 함수의 조합입니다. 즉, 클로저는 내부 함수에서 **외부 함수의 범위에 대한 접근**을 제공합니다.

```
function init() {
  var name = "Mozilla";
  function displayName() {
    console.log(name); // 부모 함수에서 선언된 변수를 사용한다.
  }
  displayName();
}
```

- "외부 함수의 범위에 대한 **접근**": 그럼 열려있는 것 아닌가?

결론

- Closure: **런타임에 대해 닫혀있음.** (Lexical scoping)

결론

- Closure: **런타임에 대해 닫혀있음.** (Lexical scoping)
- 런타임에 열려있는 건 Dynamic scoping
    - Lexical(Static) scoping ↔️ Dynamic scoping

결론

- Closure: **런타임에 대해 닫혀있음.** (Lexical scoping)
- 런타임에 열려있는 건 Dynamic scoping
    - Lexical(Static) scoping ↔️ Dynamic scoping
- 리액트의 Context API는 Dynamic scoping 처럼 작동함

Lexical scoping

```
function init() {
  var name = "Mozilla";
  function displayName() {
    console.log(name);
  }
  displayName();
}
```

- `displayName()`의 `name`이 `init()` Execution context의 `name`

Lexical scoping

```
function init() {
  var name = "Mozilla";
  function displayName() {
    console.log(name);
  }
  displayName();
}
```

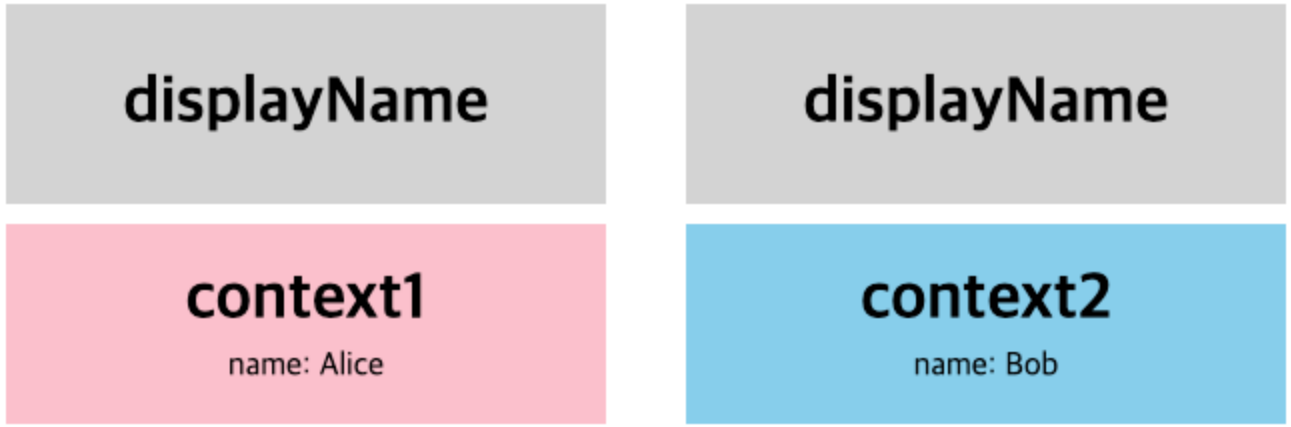
- `displayName()`의 `name`이 `init()` Execution context의 `name`
- 변수가 가리키는 대상이 **런타임**에 변하지 않으므로 static(정적)
- Static scoping

Dynamic scoping

- 변수가 가리키는 대상이 **런타임**에 변함.

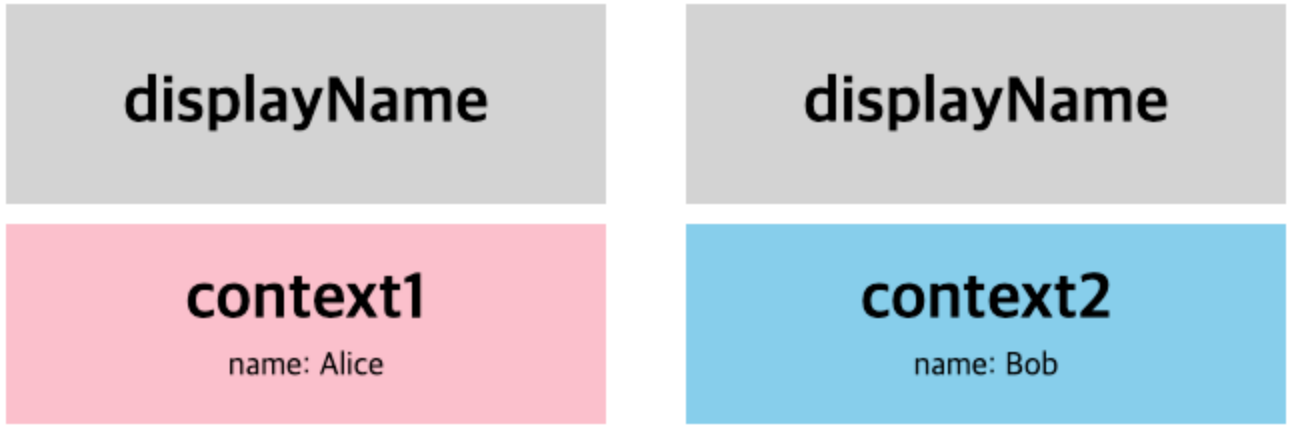
```
function displayName() {
  console.log(name);
}

function context1() {
  var name = "Alice";
  displayName(); // Alice
}

function context2() {
  var name = "Bob";
  displayName(); // Bob
}
```

Dynamic Scoping



- 콜 스택을 타고 내려가면서 context를 확인함.
- 콜 스택을 타야 알 수 있다. → 런타임에 알 수 있다. → 동적

Dynamic Scoping



- 콜 스택을 타고 내려가면서 context를 확인함.
- 콜 스택을 타야 알 수 있다. → 런타임에 알 수 있다. → 동적
- 정상적인 프로그래밍 언어라면 채택하지 않음
    - 타입 안정성, 디버깅 난이도 ...

Context API

```
function UserCard() {
  const { user } = useUserContext()

  return <div>
    <p>{user.name}</p>
    <p>{user.age}</p>
  </div>
}

<UserProvider user={{ name: 'Alice', age: 20 }}>
  <UserCard>
</UserProvider>

<UserProvider user={{ name: 'Bob', age: 10 }}>
  <UserCard>
</UserProvider>
```

Context API

```
function UserCard() {
  const { user } = useUserContext()
  return <div>
    <p>{user.name}</p>
    <p>{user.age}</p>
  </div>
}
<UserProvider user={{ name: 'Alice', age: 20 }}>
  <UserCard>
</UserProvider>
<UserProvider user={{ name: 'Bob', age: 10 }}>
  <UserCard>
</UserProvider>
```

- 의존성 주입: `UserCard` 컴포넌트는 `user` 가 어떤 인스턴스인지 모름.

Context API 트릭

```
function UserCard() {
  const { user } = useUserContext()  // Alice
  return <div>
    <p>{user.name}</p>
    <p>{user.age}</p>
  </div>
}

function App() {
  return (
    <UserProvider user={{ name: 'Alice', age: 20 }}>
      <UserCard>
    </UserProvider>
  )
}
```

Context API 트릭

```
function UserCard() {
  const { user } = useUserContext()
  return <div>
    <p>{user.name}</p>
    <p>{user.age}</p>
  </div>
}

function App() {
  return (
    <UserProvider user={{ name: 'Alice', age: 20 }}>
      <UserProvider user={{ name: 'Bob', age: 10 }}>
        <UserCard>
      </UserProvider>
    </UserProvider>
  )
}
```

Context API 트릭

```
function UserCard() {
  const { user } = useUserContext() // Bob
  return <div>
    <p>{user.name}</p>
    <p>{user.age}</p>
  </div>
}

function App() {
  return (
    <UserProvider user={{ name: 'Alice', age: 20 }}>
      <UserProvider user={{ name: 'Bob', age: 10 }}>
        <UserCard>
      </UserProvider>
    </UserProvider>
  )
}
```

Context API 트릭

Context API 트릭

```
function UserCard() {
  const { user } = useUserContext()
  return <div>
    <p>{user.name}</p>
    <p>{user.age}</p>
  </div>
}

function App() {
  return (
    <UserProvider user={{ name: 'Alice', age: 20 }}> // 범용적인 Context
      <UserProvider user={{ name: 'Anonymous', age: 99 }}> // 특수한 Context
        <UserCard>
      </UserProvider>
    </UserProvider>
  )
}
```

Recap

- Closure: 런타임에 닫혀있음
- Dynamic Scoping과 리액트 Context API 유사성

Recap

- Closure: 런타임에 닫혀있음
- Dynamic Scoping과 리액트 Context API 유사성
- 그럼 `this`는?

참고

- 왜 나는 React를 사랑하는가
- 洪民憙 홍민희 블로그
- 최강혁님 트위터:
  - 스레드 1
  - 스레드 2