라이브러리 컴파일

이 가이드는 라이브러리 작성자가 React 컴파일러를 사용하여 최적화된 라이브러리 코드를 사용자에게 제공하는 방법을 설명합니다.

컴파일된 코드를 배포해야 하는 이유

라이브러리 작성자는 npm에 배포하기 전에 라이브러리 코드를 컴파일할 수 있습니다. 이는 여러 가지 이점을 제공합니다.

- **모든 사용자를 위한 성능 향상** - 라이브러리 사용자가 아직 React 컴파일러를 사용하지 않더라도 최적화된 코드를 얻습니다.
- **사용자에게 설정이 필요 없음** - 최적화가 바로 작동합니다.
- **일관된 동작** - 모든 사용자가 빌드 설정에 관계없이 동일한 최적화된 버전을 얻습니다.

컴파일 설정하기

라이브러리의 빌드 프로세스에 React 컴파일러를 추가하세요.

터미널 복사

```
npm install -D babel-plugin-react-compiler@latest
```

라이브러리를 컴파일하도록 빌드 도구를 설정하세요. Babel을 사용하는 예시입니다.

```
// babel.config.js
module.exports = {
  plugins: [
    'babel-plugin-react-compiler',
  ],
  // ... other config
};
```

하위 호환성

라이브러리가 React 19 미만 버전을 지원하는 경우 추가 설정이 필요합니다.

1. 런타임 패키지 설치하기

`react-compiler-runtime` 을 직접 의존성 dependencies 으로 설치하는 것을 권장합니다.

터미널 복사

```
npm install react-compiler-runtime@latest
```

```
{
  "dependencies": {
    "react-compiler-runtime": "^1.0.0"
  },
  "peerDependencies": {
    "react": "^17.0.0 || ^18.0.0 || ^19.0.0"
  }
}
```

2. `target` 버전 설정하기

라이브러리가 지원하는 최소 React 버전을 설정하세요.

```
{
  target: '17', // 지원하는 최소 React 버전
}
```

테스트 전략

호환성을 보장하기 위해 컴파일 유무에 관계없이 라이브러리를 테스트하세요. 컴파일된 코드에 대해 기존 테스트를 실행하고 컴파일러를 우회하는 별도의 테스트 설정도 만드세요. 이렇게 하면 컴파일 과정에서 발생할 수 있는 문제를 발견하고 모든 시나리오에서 라이브러리가 올바르게 작동하는지 확인할 수 있습니다.

문제 해결

라이브러리가 이전 React 버전에서 작동하지 않는 경우

컴파일된 라이브러리가 React 17 또는 18에서 오류를 발생시키는 경우입니다.

1. `react-compiler-runtime` 이 의존성으로 설치되어 있는지 확인하세요.
2. `target` 설정이 지원하는 최소 React 버전과 일치하는지 확인하세요.
3. 런타임 패키지가 배포된 번들에 포함되어 있는지 확인하세요.

다른 Babel 플러그인과의 컴파일 충돌

일부 Babel 플러그인은 React 컴파일러와 충돌할 수 있습니다.

1. `babel-plugin-react-compiler` 를 플러그인 목록의 앞쪽에 배치하세요.
2. 다른 플러그인에서 충돌하는 최적화를 비활성화하세요.
3. 빌드 출력을 철저히 테스트하세요.

런타임 모듈을 찾을 수 없는 경우

사용자가 “Cannot find module ‘react-compiler-runtime’” 오류를 보는 경우입니다.

1. 런타임이 `devDependencies`가 아닌 `dependencies`에 나열되어 있는지 확인하세요.
2. 번들러가 출력에 런타임을 포함하는지 확인하세요.
3. 패키지가 라이브러리와 함께 npm에 배포되었는지 확인하세요.

다음 단계

- 컴파일된 코드를 위한 디버깅 기법에 대해 알아보세요.
- 모든 컴파일러 옵션을 위한 설정 옵션을 확인하세요.
- 선택적 최적화를 위한 컴파일 모드를 살펴보세요.

이전
"use no memo"

감사의 말

약관