지시어

React 컴파일러 지시어는 특정 함수에 대한 컴파일 적용 여부를 제어하는 특수 문자열 리터럴입니다.

```
function MyComponent() {
  "use memo"; // 이 컴포넌트를 컴파일 대상으로 설정합니다.
  return <div>{/* ... */}</div>;
}
```

개요

React 컴파일러 지시어는 컴파일러가 최적화할 함수를 세밀하게 제어할 수 있도록 합니다. 함수 본문의 시작 부분이나 모듈의 최상단에 배치되는 문자열 리터럴입니다.

사용 가능한 지시어

- **`"use memo"`** - 함수를 컴파일 대상으로 선택합니다.
- **`"use no memo"`** - 함수를 컴파일 대상에서 제외합니다.

빠른 비교

지시어	목적	사용 시점
`"use memo"`	컴파일 강제	`annotation` 모드를 사용하거나 `infer` 모드의 휴리스틱을 재정의Override할 때
`"use no memo"`	컴파일 제외	이슈를 디버깅하거나 호환되지 않는 코드를 다룰 때

사용법

함수 수준 지시어

함수의 시작 부분에 지시어를 선언하여 컴파일을 제어합니다.

```
// 컴파일 대상으로 설정
function OptimizedComponent() {
  "use memo";
  return <div>This will be optimized</div>;
}

// 컴파일 대상에서 제외
function UnoptimizedComponent() {
  "use no memo";
```

```
  return <div>This won't be optimized</div>;
}
```

모듈 수준 지시어

파일의 최상단에 선언하여 해당 모듈의 모든 함수에 적용합니다.

```
// 파일의 최상단에 선언
"use memo";

// 이 파일의 모든 함수가 컴파일됩니다
function Component1() {
  return <div>Compiled</div>;
}

function Component2() {
  return <div>Also compiled</div>;
}

// 함수 수준에서 재정의 가능
function Component3() {
  "use no memo"; // 모듈 수준에서 재정의됩니다
  return <div>Not compiled</div>;
}
```

컴파일 모드와의 상호작용

지시어는 `compilationMode` 에 따라 다르게 동작합니다.

- **`annotation` 모드**: `"use memo"` 가 선언된 함수만 컴파일됩니다.
- **`infer` 모드**: 컴파일러가 컴파일할 대상을 결정하며 지시어는 이 결정을 재정의Override합니다.
- **`all` 모드**: 모든 것이 컴파일되며 `"use no memo"` 로 특정 함수를 제외할 수 있습니다.

모범 사례

지시어는 신중하게 사용하세요

지시어는 탈출구Escape Hatch입니다. 컴파일러는 프로젝트 수준에서 설정하는 것을 권장합니다.

```
// ✅ Good - 프로젝트 전역 설정
{
  plugins: [
    ['babel-plugin-react-compiler', {
      compilationMode: 'infer'
    }]
  ]
}

// ⚠️ 필요할 때마다 지시어 사용
function SpecialCase() {
  "use no memo"; // 왜 필요한지 문서화하세요
  // ...
}
```

지시어 사용 이유를 문서화하세요

지시어를 사용하는 이유를 항상 명확히 설명하세요.

```
// ✅ Good - 명확한 설명
function DataGrid() {
  "use no memo"; // TODO: 동적 row height 이슈 해결 후 제거 (JIRA-123)
  // 복잡한 그리드 구현
}

// ❌ Bad - 설명 없음
function Mystery() {
  "use no memo";
  // ...
}
```

제거 계획을 세우세요

컴파일 제외 지시어는 임시로 사용해야 합니다.

1. TODO 주석과 함께 지시어를 추가합니다.

2. 추적용 이슈를 생성합니다.
3. 근본적인 문제를 해결합니다.
4. 지시어를 제거합니다.

```
function TemporaryWorkaround() {
  "use no memo"; // TODO: ThirdPartyLib를 v2.0으로 업그레이드한 후 제거
  return <ThirdPartyComponent />;
}
```

일반적인 패턴

점진적 도입

대규모 코드베이스에서 React 컴파일러를 도입할 때 다음과 같은 방식이 일반적입니다.

```
// annotation 모드로 시작
{
  compilationMode: 'annotation'
}

// 안정적인 컴포넌트를 컴파일 대상으로 설정
function StableComponent() {
  "use memo";
  // 충분히 테스트된 컴포넌트
}

// 나중에 infer 모드로 전환하고 문제가 있는 컴포넌트는 제외
function ProblematicComponent() {
  "use no memo"; // 제거 전에 이슈를 해결하세요
  // ...
}
```

문제 해결

지시어와 관련된 구체적인 문제는 다음 문제 해결 섹션을 참고하세요.

- `"use memo"` 문제 해결
- `"use no memo"` 문제 해결

자주 발생하는 이슈

1. **지시어가 무시됨**: 위치(파일 또는 함수의 첫 줄인지)와 철자를 확인하세요.
2. **컴파일이 계속됨**: `ignoreUseNoForget` 설정을 확인하세요.
3. **모듈 수준 지시어가 작동하지 않음**: 모든 import 문보다 앞에 있는지 확인하세요.

참고

- `compilationMode` - 컴파일러가 최적화 대상을 선택하는 방식을 설정합니다.
- `Configuration` - 전체 컴파일러 설정 옵션.
- React Compiler documentation - 시작 가이드.

이전
target

다음
"use memo"

커뮤니티

행동 강령

팀 소개

문서 기여자

감사의 말

더 보기

블로그

React Native

개인 정보 보호

약관