



# compilationMode

compilationMode 옵션은 React 컴파일러가 어떤 함수를 컴파일할지 선택하는 방식을 제어합니다.

```
{
  compilationMode: 'infer' // or 'annotation', 'syntax', 'all'
}
```

- [레퍼런스](#)

- [compilationMode](#)

- [사용법](#)

- [기본 추론 모드](#)
- [어노테이션 모드를 사용한 점진적 도입](#)
- [Flow 문법 모드 사용하기](#)
- [특정 함수 제외하기](#)

- [문제 해결](#)

- ['infer' 모드에서 컴포넌트가 컴파일되지 않는 경우](#)

---

## 레퍼런스

### compilationMode

React 컴파일러가 최적화할 함수를 결정하는 전략을 제어합니다.

### 타입

## 기본값

'infer'

## 옵션

- **'infer'** (기본값): 컴파일러가 지능형 휴리스틱을 사용하여 React 컴포넌트와 Hook을 식별합니다.
  - "use memo" 지시어로 명시적으로 표시된 함수.
  - 컴포넌트(PascalCase) 또는 Hook(use 접두사)처럼 이름이 지어진 함수이면서 JSX를 생성하거나 다른 Hook을 호출하는 함수.
- **'annotation'**: "use memo" 지시어로 명시적으로 표시된 함수만 컴파일합니다. 점진적 도입에 이상적입니다.
- **'syntax'**: Flow의 **Component** 및 **Hook** 문법을 사용하는 컴포넌트와 Hook만 컴파일합니다.
- **'all'**: 모든 최상위 함수를 컴파일합니다. React가 아닌 함수도 컴파일할 수 있으므로 권장하지 않습니다.

## 주의 사항

- 'infer' 모드는 함수가 React 명명 규칙을 따라야 감지할 수 있습니다.
- 'all' 모드를 사용하면 유틸리티 함수까지 컴파일하여 성능에 부정적인 영향을 미칠 수 있습니다.
- 'syntax' 모드는 Flow가 필요하며 TypeScript와는 작동하지 않습니다.
- 모드와 관계없이 "use no memo" 지시어가 있는 함수는 항상 건너뛴니다.

---

## 사용법

### 기본 추론 모드

기본 'infer' 모드는 React의 규칙을 따르는 대부분의 코드베이스에서 잘 작동합니다.

```
{
  compilationMode: 'infer'
}
```

이 모드에서는 다음 함수들이 컴파일됩니다.

```
// ✅ 컴파일됨: 컴포넌트처럼 이름이 지어졌고 JSX를 반환함
function Button(props) {
  return <button>{props.label}</button>;
}

// ✅ 컴파일됨: Hook처럼 이름이 지어졌고 Hook을 호출함
function useCounter() {
  const [count, setCount] = useState(0);
  return [count, setCount];
}

// ✅ 컴파일됨: 명시적인 지시어
function expensiveCalculation(data) {
  "use memo";
  return data.reduce(/* ... */);
}

// ❌ 컴파일되지 않음: 컴포넌트/Hook 패턴이 아님
function calculateTotal(items) {
  return items.reduce((a, b) => a + b, 0);
}
```

## 어노테이션 모드를 사용한 점진적 도입

점진적 마이그레이션을 위해 'annotation' 모드를 사용하여 표시된 함수만 컴파일합니다.

```
{
  compilationMode: 'annotation'
}
```

그런 다음 컴파일할 함수를 명시적으로 표시합니다.

```
// 이 함수만 컴파일됩니다
function ExpensiveList(props) {
  "use memo";
  return (
    <ul>
      {props.items.map(item => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
}

// 지시어가 없으면 컴파일되지 않습니다
function NormalComponent(props) {
  return <div>{props.content}</div>;
}
```

## Flow 문법 모드 사용하기

코드베이스가 TypeScript 대신 Flow를 사용하는 경우입니다.

```
{
  compilationMode: 'syntax'
}
```

그런 다음 Flow의 컴포넌트 문법을 사용합니다.

```
// 컴파일됨: Flow 컴포넌트 문법
component Button(label: string) {
  return <button>{label}</button>;
}

// 컴파일됨: Flow Hook 문법
hook useCounter(initial: number) {
  const [count, setCount] = useState(initial);
```

```
    return [count, setCount];
  }

  // 컴파일되지 않음: 일반 함수 문법
  function helper(data) {
    return process(data);
  }
```

## 특정 함수 제외하기

컴파일 모드와 관계없이 "use no memo" 를 사용하여 컴파일을 건너뛸 수 있습니다.

```
function ComponentWithSideEffects() {
  "use no memo"; // 컴파일 방지

  // 이 컴포넌트는 메모이제이션되어서는 안 되는 사이드 이펙트가 있습니다
  logToAnalytics('component_rendered');

  return <div>Content</div>;
}
```

## 문제 해결

### 'infer' 모드에서 컴포넌트가 컴파일되지 않는 경우

'infer' 모드에서는 컴포넌트가 React의 규칙을 따르는지 확인하세요.

```
// ❌ 컴파일되지 않음: 소문자 이름
function button(props) {
  return <button>{props.label}</button>;
}

// ✅ 컴파일됨: PascalCase 이름
function Button(props) {
  return <button>{props.label}</button>;
}
```

// ❌ 컴파일되지 않음: JSX를 생성하거나 Hook을 호출하지 않음

```
function useData() {  
  return window.localStorage.getItem('data');  
}
```

// ✅ 컴파일됨: Hook을 호출함

```
function useData() {  
  const [data] = useState(() => window.localStorage.getItem('data'));  
  return data;  
}
```

< [이전  
설정](#)

[다음  
gating](#) >

## Meta Open Source

Copyright © Meta Platforms, Inc

uwu?

## React 학습하기

[빠르게 시작하기](#)

[설치하기](#)

[UI 표현하기](#)

[상호작용성 더하기](#)

[State 관리하기](#)

[탈출구](#)

## 커뮤니티

[행동 강령](#)

[팀 소개](#)

[문서 기여자](#)

[감사의 말](#)

## API 참고서

[React APIs](#)

[React DOM APIs](#)

## 더 보기

[블로그](#)

[React Native](#)

[개인 정보 보호](#)

[약관](#)

