HOLY JS/...

2026 Autumn

Всё, что нужно знать о React Compiler



Антон Непша

Copy page

React Compiler

Introduction

Learn what React Compiler does and how it automatically optimizes your React application by handling memoization for you, eliminating the need for manual `useMemo`, `useCallback`, and `React.memo`.

Installation

Get started with installing React Compiler and learn how to configure it with your build tools.

Факты о React Compiler

Факты о React Compiler

- React Compiler нет в React 19

Факты о React Compiler

- React Compiler нет в React 19
- React Compiler не включен по умолчанию в @vitejs/plugin-react

Факты о React Compiler

- React Compiler нет в React 19
- React Compiler не включен по умолчанию в @vitejs/plugin-react
- React Compiler не обернёт все компоненты в React.memo

Факты о React Compiler

- React Compiler нет в React 19
- React Compiler не включен по умолчанию в @vitejs/plugin-react
- React Compiler не обернёт все компоненты в React.memo
- Когда вы его включите, всё сломается. И вы этого не заметите.

Мемоизация

```
const Component = ({ item }) => {

  const status = getStatus(item)

  return (
    <span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(4);
  const { item } = t0;
  let t1;
  let t2;
  if ($[0] !== item) {
    const status = getStatus(item);
    t1 = status;
    t2 = i18next.t(status);
...
```

Расширяем props

```jsx
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```jsx
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    }
...
```

Факты о React Compiler

- React Compiler нет в React 19
- React Compiler не включен по умолчанию в @vitejs/plugin-react
- React Compiler не обернёт все компоненты в React.memo
- Когда вы его включите, всё сломается. И вы этого не заметите.
- React Compiler не гарантирует мемоизацию

Обо мне

- 12 лет в IT, 10 лет в Сбере
- Разрабатывал WEB СберБанк Онлайн

Обо мне

- 12 лет в IT, 10 лет в Сбере
- Разрабатывал WEB СберБанк Онлайн
- React Compiler – моя любимая тема





Факты о React Compiler

- React Compiler нет в React 19
- React Compiler не включен по умолчанию в @vitejs/plugin-react
- React Compiler не обернёт все компоненты в React.memo
- Когда вы его включите, всё сломается. И вы этого не заметите.
- React Compiler не гарантирует мемоизацию



React Compiler нет в React 19

Из чего состоит React Compiler

babel-plugin-react-compiler

Build time

react-compiler-runtime

Runtime

Из чего состоит React Compiler

babel-plugin-react-compiler

Build time

react-compiler-runtime

Runtime

Из чего состоит React Compiler

babel-plugin-react-compiler

- Babel-плагин
- Меняет ваши исходники на уровне AST
- Добавляет хук useMemoCache в ваши компоненты

react-compiler-runtime

Runtime

Какие версии вышли после 1.0.0

```
$ npm view babel-plugin-react-compiler versions
```

Никакие версии не вышли после 1.0.0

```
  '0.0.0-experimental-a8e64ef-20260402',
  '0.0.0-experimental-cabb246-20260106',
  '0.0.0-experimental-cabb246-20260123',
  '0.0.0-experimental-ce9113a-20251015',
  '0.0.0-experimental-d664a4c-20250313',
  '0.0.0-experimental-fe727a3-20250909',
  '0.0.0-experimental-ff4e0fd-20260324',
  '0.0.0',
  '1.0.0',
  '19.0.0-beta-201e55d-20241215',
  '19.0.0-beta-ebf51a3-20250411',
  '19.1.0-rc.1',
  '19.1.0-rc.2',
  '19.1.0-rc.3',
  '19.1.0-rc.1-rc-af1b7da-20250421'
]
```

Из чего состоит React Compiler

babel-plugin-react-compiler

- Babel-плагин
- Меняет ваши исходники на уровне AST
- Добавляет хук useMemoCache в ваши компоненты
- Всё так же 1.0.0, как и в 2025
- Много experimental версий

react-compiler-runtime

Runtime

Из чего состоит React Compiler

babel-plugin-react-compiler

- Babel-плагин
- Меняет ваши исходники на уровне AST
- Добавляет хук useMemoCache в ваши компоненты

react-compiler-runtime

- Исходники хука useMemoCache

Из чего состоит React Compiler

babel-plugin-react-compiler

- Babel-плагин
- Меняет ваши исходники на уровне AST
- Добавляет хук useMemoCache в ваши компоненты

react-compiler-runtime

- Исходники хука useMemoCache

Из чего состоит React Compiler

babel-plugin-react-compiler

- Babel-плагин
- Меняет ваши исходники на уровне AST
- Добавляет хук useMemoCache в ваши компоненты

react-compiler-runtime

- Исходники хука useMemoCache

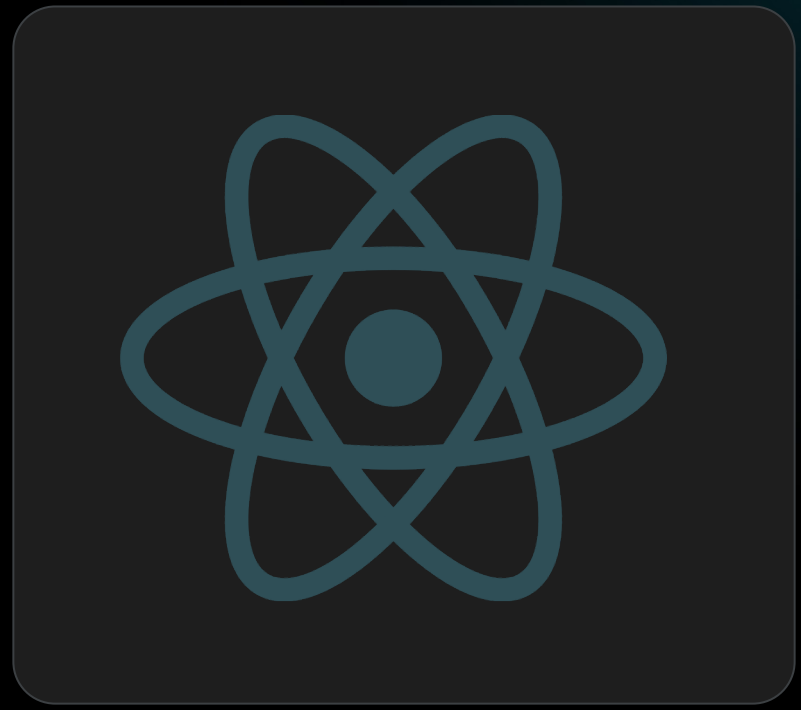
Для работы React Compiler

Нужно всего лишь два пакета

babel-plugin-react-compiler



react-compiler-runtime

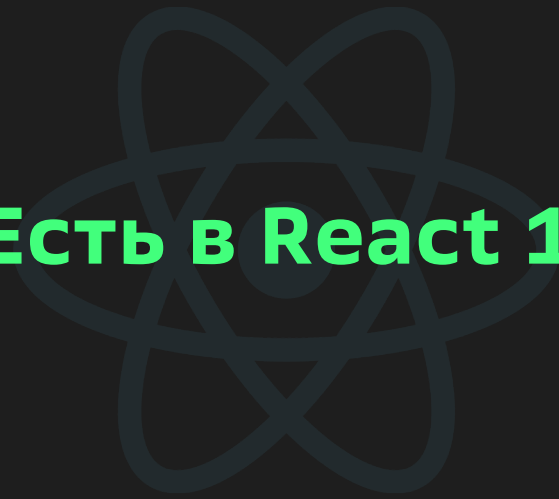


Если у вас React 19

babel-plugin-react-compiler

Устанавливаем отдельно

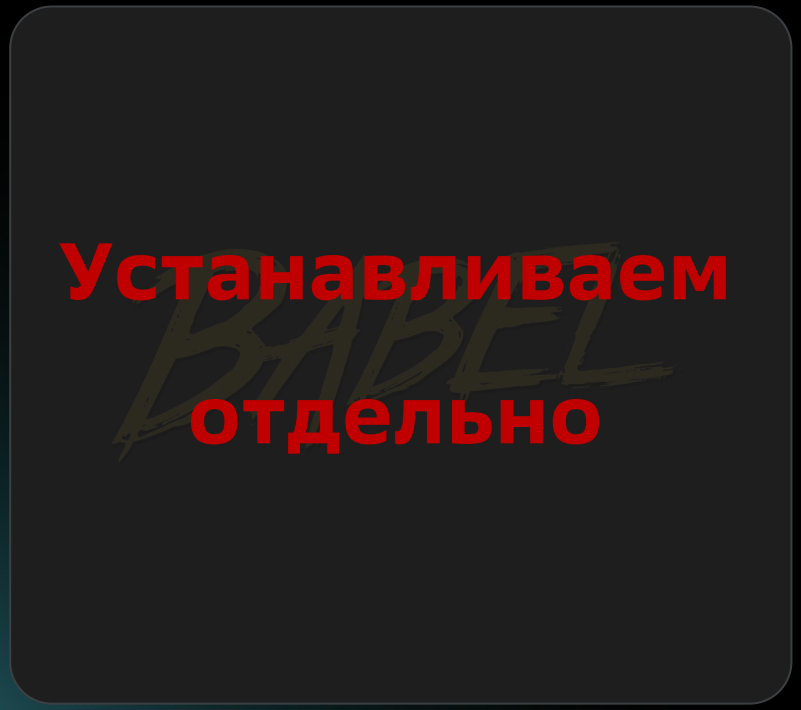
react-compiler-runtime



Есть в React 19

Если у вас React 17 или 18

babel-plugin-react-compiler



react-compiler-runtime



Если вы любите Rust

@vitejs/plugin-react@6.1

babel-plugin-react-compiler



react-compiler-runtime

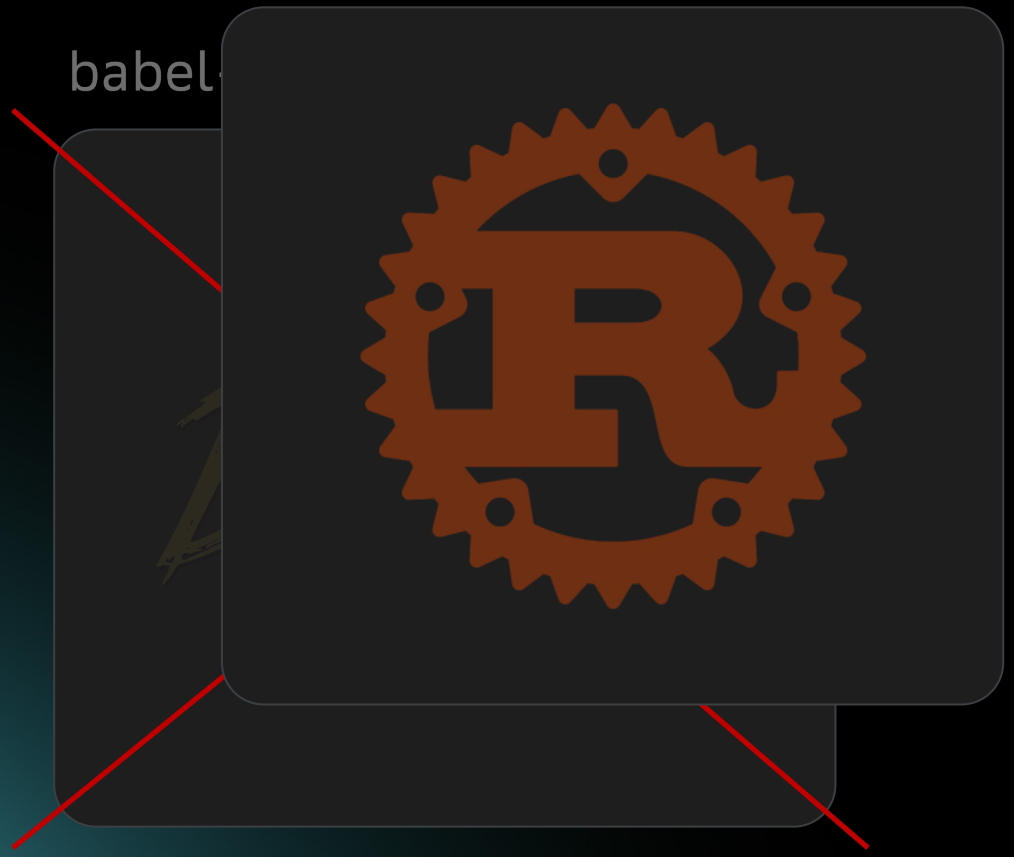
Есть в React 19

Если вы любите Rust

@vitejs/plugin-react@6.1

oxc-transform-react

babel-



react-compiler-runtime

Есть в React 19

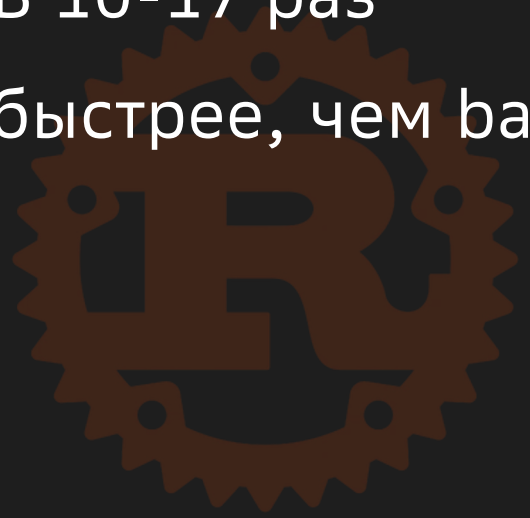
Если вы любите Rust

@vitejs/plugin-react@6.1

oxc-transform-react

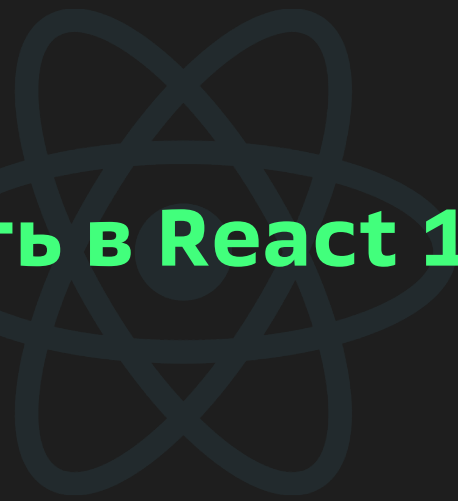
babel

- В 10-17 раз быстрее, чем babel



react-compiler-runtime

Есть в React 19



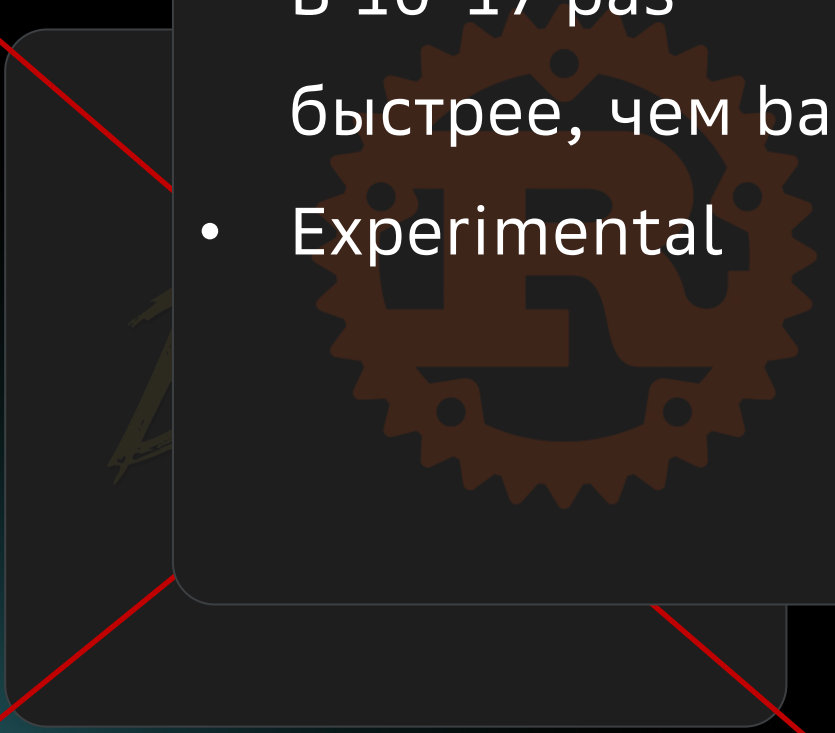
Если вы любите Rust

@vitejs/plugin-react@6.1

oxc-transform-react

babel

- В 10-17 раз быстрее, чем babel
- Experimental



react-compiler-runtime

Есть в React 19

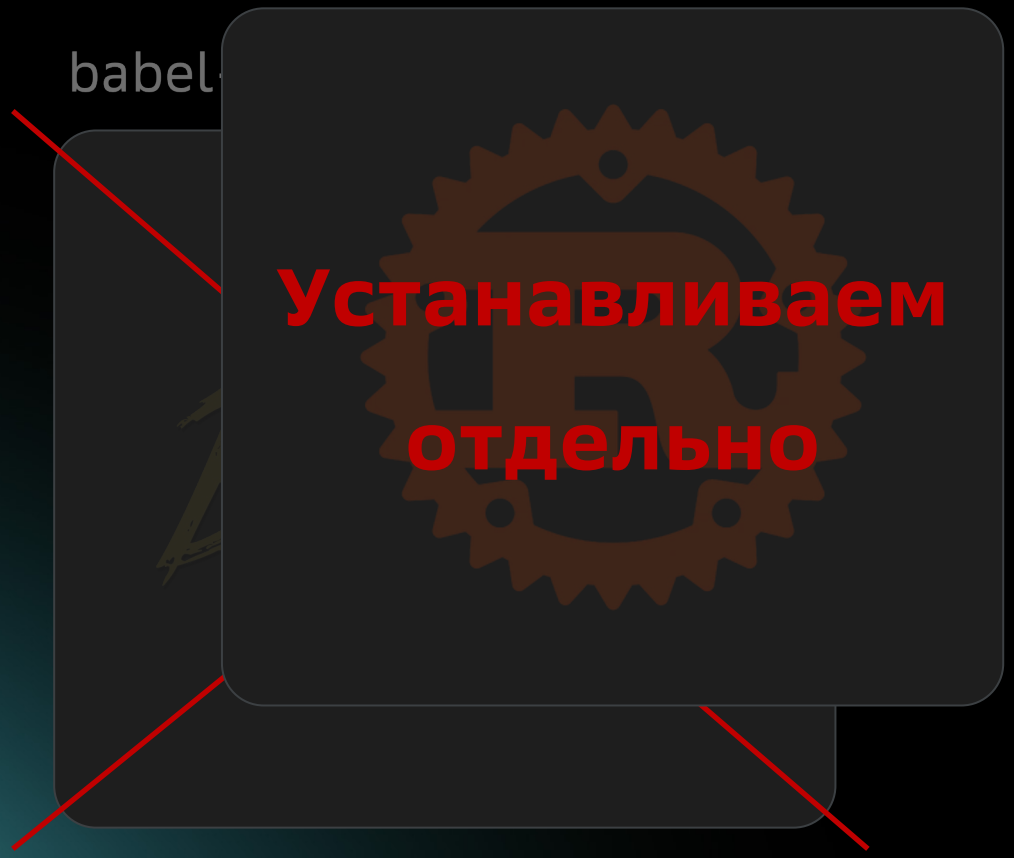


Если вы любите Rust

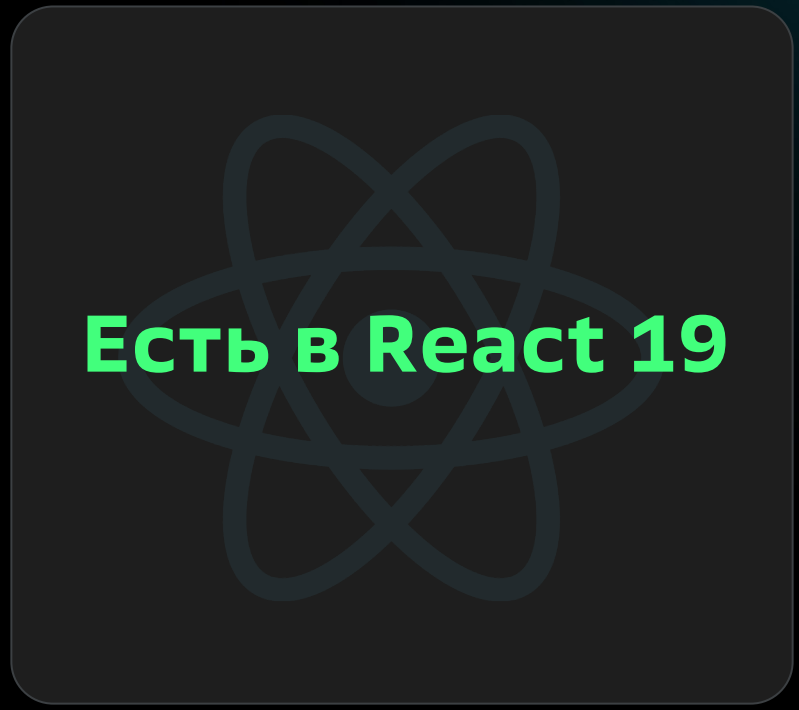
@vitejs/plugin-react@6.1

oxc-transform-react

babel-



react-compiler-runtime



React Compiler

Rust React Compiler

> [!WARNING] Native React Compiler support is experimental.

The `compiler` option uses `oxc-transform-react`, a Rust port of **React Compiler**, which must be installed as an optional peer dependency:

```
npm install -D oxc-transform-react
```

```
// vite.config.js
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react({ compiler: true })],
})
```

The `compiler` option also accepts **React Compiler options**:

```
react({ compiler: { compilationMode: 'annotation' } })
```

React Compiler не включен по умолчанию в @vitejs/plugin-react

React Compiler

Rust React Compiler

> [!WARNING] Native React Compiler support is experimental.

The `compiler` option uses `oxc-transform-react`, a Rust port of React Compiler, which must be installed as an optional peer dependency:

```
npm install -D oxc-transform-react
```

```
// vite.config.js
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react({ compiler: true })],
})
```

The `compiler` option also accepts React Compiler options:

```
react({ compiler: { compilationMode: 'annotation' } })
```

Подключение по-старому

```
export default defineConfig({
   plugins: [
      react({
         babel: {
            plugins: [
               ["babel-plugin-react-compiler", { target: "18" }],
               ...otherPlugins
            ],
         },
      })
   ]
})
```

vite.config.js

Подключение по-новому

```
import react, { reactCompilerPreset } from '@vitejs/plugin-react'
import babel from '@rolldown/plugin-babel'

export default defineConfig({
  plugins: [
    react(),
    babel({
      presets: [
        reactCompilerPreset({ target: '18' })
      ]
    })
  ],
})
```

vite.config.js

Подключение Rust-версии

oxc-transform-react

```
import react from '@vitejs/plugin-react'

export default defineConfig({
   plugins: [
      react({
         compiler: true
      })
   ]
})
```

vite.config.js

Подключение Rust-версии

oxc-transform-react

```
import react from '@vitejs/plugin-react'

export default defineConfig({
   plugins: [
      react({
         compiler: { target: '18' }
      })
   ]
})
```

vite.config.js

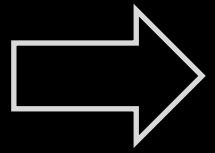
React Compiler не обернёт ваши компоненты в React.memo

Механизм работы

```
export function MyComponent(props) {
  return (
    <div>Hello {props.name}</div>
  );
}
```



babel-plugin-react-compiler
/ oxc-transform-react



```
import { c as _c } from "react/compiler-runtime";

export function MyComponent(props) {
  const $ = _c(2);
  let t0;
  if ($[0] !== props.name) {
    t0 = <div>Hello {props.name}</div>;
    $[0] = props.name;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
  return t0;
}
```

Reactive Function / Reactive Scope

Reactive Function

```
import { c as _c } from "react/compiler-runtime";
export function MyApp(props) {
  const $ = _c(2);
  let t0;
  if ($[0] !== props.name) {
    t0 = <div>Hello {props.name}</div>;
    $[0] = props.name;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
  return t0;
}
```

Reactive Function / Reactive Scope

Reactive Function

```
import { c as _c } from "react/compiler-runtime";
export function MyApp(props) {
  const $ = _c(2);
  let t0;
  if ($[0] !== props.name) {
    t0 = <div>Hello {props.name}</div>;
    $[0] = props.name;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
  return t0;
}
```

Reactive Scope

```
import { c as _c } from "react/compiler-runtime";
export function MyApp(props) {
  const $ = _c(2);
  let t0;
  if ($[0] !== props.name) {
    t0 = <div>Hello {props.name}</div>;
    $[0] = props.name;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
  return t0;
}
```

Отличие от React.memo

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    }
...
```

Отличие от React.memo

```
const Component = React.memo(
{ item, name }) => {

   const status = getStatus(item)

   return (
      <span>
         <span>{name}</span>
         {i18next.t(status)}
      </span>
   )
}
)
```

```
const Component = React.memo((t0) => {
   const $ = _c(5);
   const { item, name } = t0;
   let t1;
   if ($[0] !== item || $[1] !== name) {
      const status = getStatus(item);
      let t2;
      if ($[3] !== name) {
         t2 = <span>{name}</span>;
         $[3] = name;
         $[4] = t2;
      }
...
)
```

Когда вы включите React Compiler, всё сломается

Подключение React Compiler

```
export default defineConfig({
   plugins: [
      react({
         babel: {
            plugins: [
               ["babel-plugin-react-compiler", { target: "18" }],
               ...otherPlugins
            ],
         },
      })
   ]
})
```

vite.config.js

Настройка React Compiler

```
export default defineConfig({
  plugins: [
    react({
      babel: {
        plugins: [
          ["babel-plugin-react-compiler", { target: "18" }],
          ...otherPlugins
        ],
      },
    })
  ]
})
```

vite.config.js

Настройка React Compiler

```
const options = { target: "18" }

const defaultOptions = {
   compilationMode: "infer",
   panicThreshold: "none",
   environment: parseEnvironmentConfig({}).unwrap(),
   logger: null,
   gating: null,
   noEmit: false,
   dynamicGating: null,
   eslintSuppressionRules: null,
   flowSuppressions: true,
   ignoreUseNoForget: false,
   sources: (filename) => filename.indexOf("node_modules") === -1,
   enableReanimatedCheck: true,
   customOptOutDirectives: null,
   target: "19",
};
const result = {...defaultOptions, ...options};
```

vite.config.js

Настройка React Compiler

```
const options = { target: "18" }

const defaultOptions = {
  compilationMode: "infer",
  panicThreshold: "none",
  environment: parseEnvironmentConfig({}).unwrap(),
  logger: null,
  gating: null,
  noEmit: false,
  dynamicGating: null,
  eslintSuppressionRules: null,
  flowSuppressions: true,
  ignoreUseNoForget: false,
  sources: (filename) => filename.indexOf("node_modules") === -1,
  enableReanimatedCheck: true,
  customOptOutDirectives: null,
  target: "19",
};
const result = {...defaultOptions, ...options};
```

vite.config.js

react@19.3
Overview
Hooks
Components
APIs

react-dom@19.3
Hooks
Components
APIs
Client APIs
Server APIs
Static APIs

React Compiler
Configuration
compilationMode
gating

Reference

`compilationMode`

Controls the strategy for determining which functions the React Compiler will optimize.

Type

```
'infer' | 'syntax' | 'annotation' | 'all'
```

Default value

`'infer'`

Options

- **`'infer'`** (default): The compiler uses intelligent heuristics to identify React components and hooks:
  - Functions explicitly annotated with `"use memo"` directive
  - Functions that are named like components (PascalCase) or hooks (`use` prefix) AND create JSX and/or call other hooks
- **`'annotation'`**: Only compile functions explicitly marked with the `"use memo"` directive. Ideal for incremental adoption.
- **`'syntax'`**: Only compile components and hooks that use Flow's component and hook syntax.

ON THIS PA
Overview
Reference
compila
Usage
Default i
Incremen
Using Flo
Opting o
Troublesh
Compone
mode

Compilation Mode = infer

Скомпилирует все функции, которые:

> Содержат внутри директивы "use memo" или "use forget"

Compilation Mode = infer

Скомпилирует все функции, которые:

- Содержат внутри директивы "use memo" или "use forget"
- Начинаются на use[A-Z0-9] и внутри себя создают JSX или вызывают хук

Compilation Mode = infer

Скомпилируется

```
const useCustomState = (input) => {
  return useState(input)
}
```

Молча пропустит

```
const useeeeeeee = (input) => {
  return useState(input)
}
```

Compilation Mode = infer

Скомпилируется

```
const useCustomState = (input) => {
  return useState(input)
}
```

```
const use1 = () => {
  use2()
}
```

Молча пропустит

```
const useeeeeeee = (input) => {
  return useState(input)
}
```

Compilation Mode = infer

Скомпилируется

```
const useCustomState = (input) => {
  return useState(input)
}
```

```
const use1 = () => {
  use2()
}
```

```
const useJSX = (input) => {
  const a = <div/>
}
```

Молча пропустит

```
const useeeeeeee = (input) => {
  return useState(input)
}
```

Input

```
1  const useCustomState = (input) => {
2    return useState(input)
3  }
```

- Output

```
const useCustomState = (input) => {
  return useState(input);
};
```

SourceMap
EnvironmentConfig
HIR
PruneMaybeThrows
DropManualMemoization
InlineImmediatelyInvokedFunctionExpressions
MergeConsecutiveBlocks
SSA
EliminateRedundantPhi
ConstantPropagation



React Compiler Playground

Show Internals

Reset

Share

Input

```
1  const useCustomState = (input) => {
2    return useState(input)
3  }
```

- Output

```
const useCustomState = (input) => {
  return useState(input);
};
```

SourceMap

EnvironmentConfig

HIR

PruneMaybeThrows

DropManualMemoization

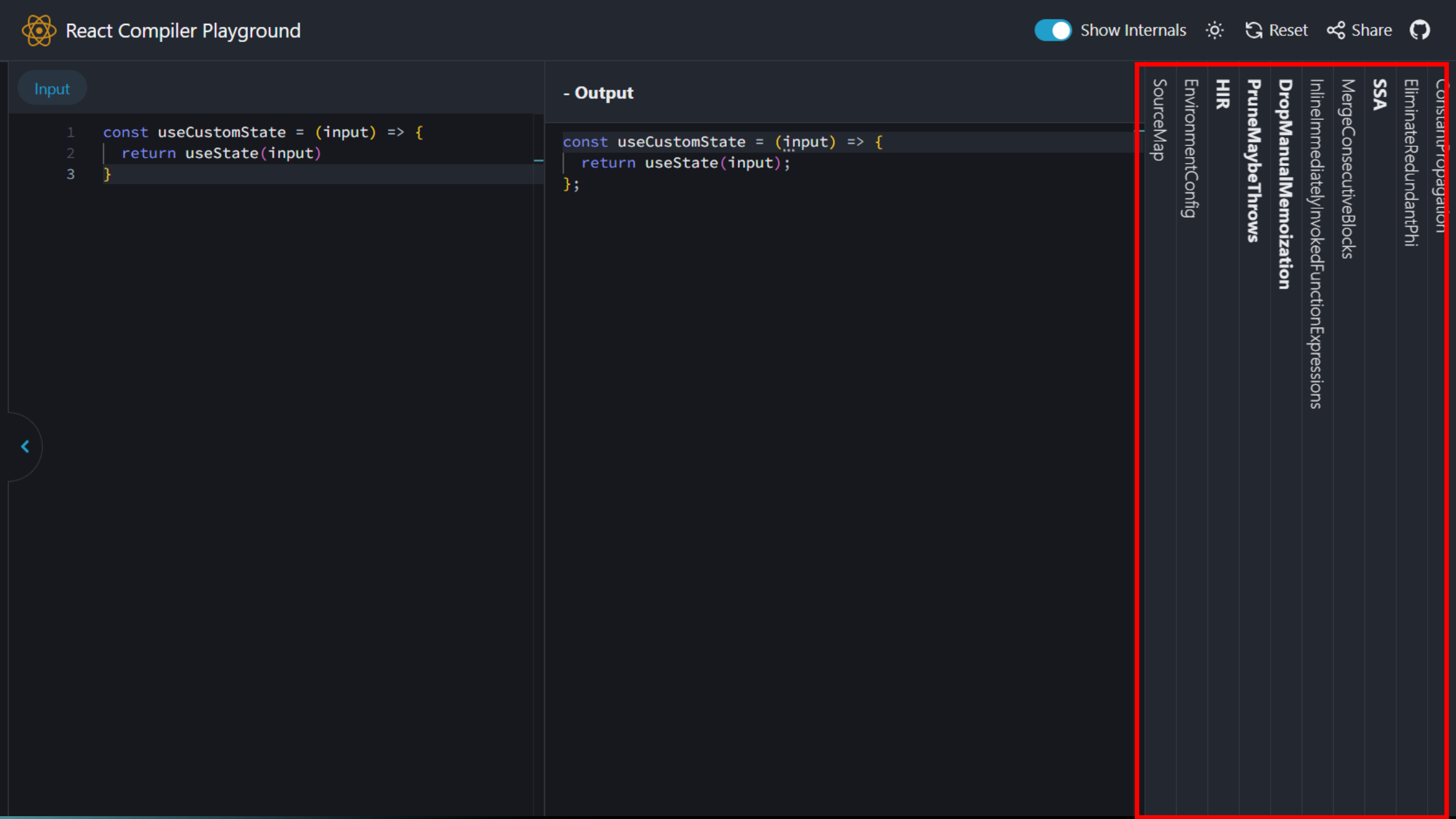
InlineImmediatelyInvokedFunctionExpressions

MergeConsecutiveBlocks

SSA

EliminateRedundantPhi

ConstantPropagation


React Compiler Playground
Show Internals
Reset
Share
Input
1 const useCustomState = (input) => {
2 return useState(input)
3 }
- Output
const useCustomState = (input) => {
return useState(input);
};
ConstantPropagation
EliminateRedundantPhi
SSA
MergeConsecutiveBlocks
InlineImmediatelyInvokedFunctionExpressions
DropManualMemoization
PruneMaybeThrows
HIR
EnvironmentConfig
SourceMap

PruneHoistedContexts
RenameVariables
StabilizeBlockIds
ExtractScopeDeclarationsFromDestructuring
PromoteUsedTemporaries
PruneUnusedLValues
PropagateEarlyReturns
PruneAlwaysInvalidatingScopes
MergeReactiveScopesThatInvalidateTogether
PruneUnusedScopes
PruneNonReactiveDependencies
PruneNonEscapingScopes
PruneUnusedLabels
BuildReactiveFunction
PropagateScopeDependenciesHIR
FlattenScopesWithHooksOrUseHIR
FlattenReactiveLoopsHIR
BuildReactiveScopeTerminalsHIR
MergeOverlappingReactiveScopesHIR
AlignReactiveScopesToBlockScopesHIR
PruneUnusedLabelsHIR
AlignObjectMethodScopes
AlignMethodCallScopes
OutlineFunctions
MemoizeFbtAndMacroOperandsInSameScope
InferReactiveScopeVariables
RewriteInstructionKindsBasedOnReassignment
InferReactivePlaces
InferMutationAliasingRanges
DeadCodeElimination
InferMutationAliasingEffects
AnalyseFunctions
OptimizePropsMethodCalls
InferTypes
ConstantPropagation
EliminateRedundantPhi
SSA
MergeConsecutiveBlocks
InlineImmediatelyInvokedFunctionExpressions
DropManualMemoization
PruneMaybeThrows
HIR
EnvironmentConfig
SourceMap

Input

```
1  const useMyFunction = () => {
2    return myFunction()
3  }
```

- Output

```
const useMyFunction = () => {
  return myFunction();
};
```

SourceMap

Compilation Mode = infer

Скомпилируется

```
const use123 = (input) => {
  return useState(input)
}
```

```
const useHook = () => {
  use2()
}
```

```
const useJSX = (input) => {
  const a = <div/>
}
```

Молча пропустит

```
const useeeeeeee = (input) => {
  return useState(input)
}
```

```
const useMyFunction = (input) => {
  return myFunction(input)
}
```

Compilation Mode = infer

Скомпилирует все функции, которые:

- Содержат внутри директивы "use memo" или "use forget"
- Начинаются на use[A-Z0-9] и внутри себя создают JSX или вызывают хук

Compilation Mode = infer

Скомпилирует все функции, которые:

- Содержат внутри директивы "use memo" или "use forget"
- Начинаются на use[A-Z0-9] и внутри себя создают JSX или вызывают хук
- Названы с большой буквы, внутри себя создают JSX или вызывают хук, входящие параметры подходят под сигнатуры компонентов

Compilation Mode = infer

Скомпилируется

```
const MyComponent = (props) => (
  <div>
    <span>Hello</span>
  </div>
)
```

Compilation Mode = infer

Скомпилируется

```
const MyComponent = (props) => (
  <div>
    <span>Hello</span>
  </div>
)
```

Молча пропустит

```
const MyComponent = (props, test123) => (
  <div>
    <span>Hello</span>
  </div>
)
```

Compilation Mode = infer

Скомпилируется

```
const MyComponent = (props) => (
  <div>
    <span>Hello</span>
  </div>
)
```

```
const T_T_____HELLO = () => (
  <div>
    <span>Hello</span>
  </div>
)
```

Молча пропустит

```
const MyComponent = (props, test123) => (
  <div>
    <span>Hello</span>
  </div>
)
```

Compilation Mode = infer

Скомпилируется

```
const MyComponent = (props) => (
  <div>
    <span>Hello</span>
  </div>
)
```

```
const T_T_____HELLO = () => (
  <div>
    <span>Hello</span>
  </div>
)
```

```
const o_o = memo(() => (
  <div />
))
```

Молча пропустит

```
const MyComponent = (props, test123) => (
  <div>
    <span>Hello</span>
  </div>
)
```

Compilation Mode = infer

Скомпилирует все функции, которые:

- Содержат внутри директивы "use memo" или "use forget"
- Начинаются на use[A-Z0-9] и внутри себя создают JSX или вызывают хук
- Названы с большой буквы, внутри себя создают JSX или вызывают хук, входящие параметры подходят под сигнатуры компонентов
- Анонимные функции, переданные в memo() или forwardRef(), внутри себя создают JSX или вызывают хук

Compilation Mode = infer

```
const Items = ({ items }) => {
  return items.map(() => (
    <Item />
  ))
}
```

Compilation Mode = infer

Молча пропустит

```
const Items = ({ items }) => {
  return items.map(() => (
    <Item />
  ))
}
```

Compilation Mode = infer

```
const Items2 = ({ items, test }) => {
  return items.map((item) => {
    const formatted = useMemo(() => {
      return item.getItemName()
    }, [item, test])


    return (
      <Item name={formatted} />
    )
  })
}
```

Compilation Mode = infer

Молча пропустит

```
const Items2 = ({ items, test }) => {
   return items.map((item) => {
      const formatted = useMemo(() => {
         return item.getItemName()
      }, [item, test])


      return (
         <Item name={formatted} />
      )
   })
}
```

Compilation Mode = infer

Скомпилирует все функции, которые:

- Содержат внутри директивы "use memo" или "use forget"
- Начинаются на use[A-Z0-9] и внутри себя создают JSX или вызывают хук
- Названы с большой буквы, внутри себя создают JSX или вызывают хук, входящие параметры подходят под сигнатуры компонентов
- Анонимные функции, переданные в memo() или forwardRef(), внутри себя создают JSX или вызывают хук
- Создание JSX или вызовы хуков должны быть непосредственно в теле функции

Импорты не считаются

```
import { Button } from '../components'

const MemoizedButton = React.memo(Button)
```

Содержимое соседних функций не учитывается

```
const myFunc = (value) => {
  return `${value}`
}


const useHook = (foo) => useMemo(() => {
  return myFunc(foo)
}, [foo])
```

```
const myFunc = (value) => {
  return `${value}`;
};

const useHook = (foo) => {
  const $ = _c(2);
  let t0;
  if ($[0] !== foo) {
    t0 = myFunc(foo);
    $[0] = foo;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
  return t0;
};
```

Compilation Mode = infer

Скомпилирует все функции, которые:

- Содержат внутри директивы "use memo" или "use forget"
- Начинаются на use[A-Z0-9] и внутри себя создают JSX или вызывают хук
- Названы с большой буквы, внутри себя создают JSX или вызывают хук, входящие параметры подходят под сигнатуры компонентов
- Анонимные функции, переданные в memo() или forwardRef(), внутри себя создают JSX или вызывают хук
- Создание JSX или вызовы хуков должны быть непосредственно в теле функции
- Каждая функция компилируется индивидуально, без учёта импортов и содержимого соседних функций

react@19.3
Overview
Hooks
Components
APIs

react-dom@19.3
Hooks
Components
APIs
Client APIs
Server APIs
Static APIs

React Compiler
Configuration
compilationMode
gating

Reference

`compilationMode`

Controls the strategy for determining which functions the React Compiler will optimize.

Type

```
'infer' | 'syntax' | 'annotation' | 'all'
```

Default value

`'infer'`

Options

- **`'infer'`** (default): The compiler uses intelligent heuristics to identify React components and hooks:
  - Functions explicitly annotated with `"use memo"` directive
  - Functions that are named like components (PascalCase) or hooks (`use` prefix) AND create JSX and/or call other hooks
- **`'annotation'`**: Only compile functions explicitly marked with the `"use memo"` directive. Ideal for incremental adoption.
- **`'syntax'`**: Only compile components and hooks that use Flow's component and hook syntax.

ON THIS PA
Overview
Reference
compila
Usage
Default i
Incremen
Using Flo
Opting o
Troublesh
Compone
mode

Вспоминаем баг с мемоизацией

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    }
...
```

Есть ещё такой

```
const Component = ({ items }) => {

   const total = useMemo(
      () => heavyFunction(items),
      [items]
   )

   return (
      <span>{total * 2}</span>
   )
}
```

```
const Component = (t0) => {
   const $ = _c(2);
   const { items } = t0;
   const total = heavyFunction(items);

   const t1 = total * 2;
   let t2;
   if ($[0] !== t1) {
      t2 = <span>{t1}</span>;
      $[0] = t1;
      $[1] = t2;
   } else {
      t2 = $[1];
   }
   return t2;
};
```

compilationMode: "infer" опасен

```
const options = { target: "18" }

const defaultOptions = {
  compilationMode: "infer",
  panicThreshold: "none",
  environment: parseEnvironmentConfig({}).unwrap(),
  logger: null,
  gating: null,
  noEmit: false,
  dynamicGating: null,
  eslintSuppressionRules: null,
  flowSuppressions: true,
  ignoreUseNoForget: false,
  sources: (filename) => filename.indexOf("node_modules") === -1,
  enableReanimatedCheck: true,
  customOptOutDirectives: null,
  target: "19",
};
const result = {...defaultOptions, ...options};
```

vite.config.js

А он включен по умолчанию

oxc-transform-react

```
import react from '@vitejs/plugin-react'

export default defineConfig({
   plugins: [
      react({
         compiler: true
      })
   ]
})
```

vite.config.js

А он включен по умолчанию

oxc-transform-react

```
import react from '@vitejs/plugin-react'

export default defineConfig({
   plugins: [
      react({
         compiler: {
            compilation_mode: "annotation"
         }
      })
   ]
})
```

vite.config.js

Compilation mode = "annotation"

```
const Component = ({ item, name }) => {
  "use memo"
   const status = getStatus(item)

   return (
      <span>
        <span>{name}</span>
        {i18next.t(status)}
      </span>
   )
}
```

Об ошибках вы не узнаете

```
const options = { target: "18" }

const defaultOptions = {
  compilationMode: "infer",
  panicThreshold: "none",
  environment: parseEnvironmentConfig({}).unwrap(),
  logger: null,
  gating: null,
  noEmit: false,
  dynamicGating: null,
  eslintSuppressionRules: null,
  flowSuppressions: true,
  ignoreUseNoForget: false,
  sources: (filename) => filename.indexOf("node_modules") === -1,
  enableReanimatedCheck: true,
  customOptOutDirectives: null,
  target: "19",
};
const result = {...defaultOptions, ...options};
```

vite.config.js

ED
t
n
piler
n
Adoption
and Troubleshooting
CT
the UI

Is it safe to use?

React Compiler is now stable and has been tested extensively in production. While it has been used in production at companies like , rolling out the compiler to production for your app will depend on the health of your codebase and how well you've followed the Rules of React.

What build tools are supported?

React Compiler can be installed across several build tools such as Babel, Vite, Metro, and Rsbuild.

React Compiler is primarily a light Babel plugin wrapper around the core compiler, which was designed to be decoupled from Babel itself. While the initial stable version of the compiler will remain primarily a Babel plugin, we are working with the swc and oxc teams to build first class support for React Compiler so you won't have to add Babel back to your build pipelines in the future.

all_errors – чтобы видеть ошибки в build-time

oxc-transform-react

```
import react from '@vitejs/plugin-react'

export default defineConfig({
   plugins: [
      react({
         compiler: {
            compilation_mode: "annotation",
            panic_threshold: "all_errors"
         }
      })
   ]
})
```

vite.config.js

gating

oxc-transform-react

```
import react from '@vitejs/plugin-react'

export default defineConfig({
   plugins: [
      react({
         compiler: {
            compilation_mode: "annotation",
            panic_threshold: "all_errors",
            gating: {
               source: "blablabla",
               importSpecifierName: "myFunction"
            }
         }
      })
   ]
})
```

vite.config.js

Без gating

Component

```
const MyComponent = ({ name }) => {
  return (
    <span>Hello, {name}</span>
  )
}
```

Compiler Output

```
import { c as _c } from "react/compiler-runtime";

const MyComponent = (t0) => {
  const $ = _c(2);
  const { name } = t0;
  let t1;
  if ($[0] !== name) {
    t1 = <span>Hello, {name}</span>;
    $[0] = name;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  return t1;
};
```

Что даёт gating

Component

```
const MyComponent = ({ name }) => {
   return (
      <span>Hello, {name}</span>
   )
}
```

vite.config.ts

```
...
gating: {
   source: "blablabla",
   importSpecifierName: "myFunction"
}
```

Что даёт gating

Component

```
const MyComponent = ({ name }) => {
  return (
    <span>Hello, {name}</span>
  )
}
```

vite.config.ts

```
...
gating: {
  source: "blablabla",
  importSpecifierName: "myFunction"
}
```

Compiler Output

```
import { myFunction } from "blablabla";
import { c as _c } from "react/compiler-runtime";
```

Что даёт gating

Component

```
const MyComponent = ({ name }) => {
  return (
    <span>Hello, {name}</span>
  )
}
```

vite.config.ts

```
...
gating: {
  source: "blablabla",
  importSpecifierName: "myFunction"
}
```

Compiler Output

```
import { myFunction } from "blablabla";
import { c as _c } from "react/compiler-runtime";

const MyComponent = myFunction() ? (t0) => {
  const $ = _c(2);
  const { name } = t0;
  let t1;
  if ($[0] !== name) {
    t1 = <span>Hello, {name}</span>;
    $[0] = name;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  return t1;
} : ({ name }) => {
  return <span>Hello, {name}</span>;
};
```

Что даёт gating

Component

```
const MyComponent = ({ name }) => {
  return (
    <span>Hello, {name}</span>
  )
}
```

vite.config.ts

```
...
gating: {
  source: "blablabla",
  importSpecifierName: "myFunction"
}
```

Compiler Output

```
import { myFunction } from "blablabla";
import { c as _c } from "react/compiler-runtime";

const MyComponent = myFunction() ? (t0) => {
  const $ = _c(2);
  const { name } = t0;
  let t1;
  if ($[0] !== name) {
    t1 = <span>Hello, {name}</span>;
    $[0] = name;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  return t1;
} : ({ name }) => {
  return <span>Hello, {name}</span>;
};
```

Выполняется в каждом компоненте

Что даёт gating

Component

```
const MyComponent = ({ name }) => {
  return (
    <span>Hello, {name}</span>
  )
}
```

vite.config.ts

```
...
gating: {
  source: "blablabla",
  importSpecifierName: "myFunction"
}
```

Compiler Output

```
import { myFunction } from "blablabla";
import { c as _c } from "react/compiler-runtime";

const MyComponent = myFunction() ? (t0) => {
  const $ = _c(2);
  const { name } = t0;
  let t1;
  if ($[0] !== name) {
    t1 = <span>Hello, {name}</span>;
    $[0] = name;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  return t1;
} : ({ name }) => {
  return <span>Hello, {name}</span>;
};
```

Увеличивает вес сборки

React Compiler не гарантирует мемоизацию

Хук useMemo

input

```
import { myHeavyCalculation } from './utils'

const useMyHook = ({ foo, bar }) => useMemo(() => {
   return foo * myHeavyCalculation(bar)
}, [foo, bar])
```

Хук useMemo

input

```
import { myHeavyCalculation } from './utils'

const useMyHook = ({ foo, bar }) => useMemo(() => {
   return foo * myHeavyCalculation(bar)
}, [foo, bar])
```

output

```
import { myHeavyCalculation } from "./utils";

const useMyHook = (t0) => {
   const { foo, bar } = t0;
   return foo * myHeavyCalculation(bar);
};
```

React Compiler мемоизирует иначе

- Не мемоизирует примитивы

React Compiler не мемоизирует примитивы

input

```
const useMyHook = ({ foo, bar }) => useMemo(() => {
  return 2 * 3
}, [])
```

output

```
const useMyHook = (t0) => {
  return 6;
};
```

React Compiler не мемоизирует примитивы

input

```
const useMyHook = ({ foo, bar }) => useMemo(() => {
  return `${foo} ${bar}`
}, [foo, bar])
```

output

```
const useMyHook = (t0) => {
  const { foo, bar } = t0;
  return `${foo} ${bar}`;
};
```

React Compiler мемоизирует ссылочные типы

input

```
const useMyHook =
({ foo, bar }) => useMemo(() => {
  return {
    foo,
    bar
  }
}, [foo, bar])
```

output

```
const useMyHook = (t0) => {
  const $ = _c(3);
  const { foo, bar } = t0;
  let t1;
  if ($[0] !== bar || $[1] !== foo) {
    t1 = { foo, bar };
    $[0] = bar;
    $[1] = foo;
    $[2] = t1;
  } else {
    t1 = $[2];
  }
  return t1;
};
```

React Compiler мемоизирует ссылочные типы

input

```
const useMyHook2 =
({ foo, bar }) => useMemo(() => {
  return [
    foo,
    bar
  ]
}, [foo, bar])
```

output

```
const useMyHook2 = (t0) => {
  const $ = _c(3);
  const { foo, bar } = t0;
  let t1;
  if ($[0] !== bar || $[1] !== foo) {
    t1 = [foo, bar];
    $[0] = bar;
    $[1] = foo;
    $[2] = t1;
  } else {
    t1 = $[2];
  }
  return t1;
};
```

Результаты функций

input

```
import { myHeavyCalculation } from './utils'

const useMyHook = ({ foo, bar }) => useMemo(() => {
   return myHeavyCalculation(foo, bar)
}, [foo, bar])
```

Compiler не знает, что возвращает функция

input

```
import { myHeavyCalculation } from './utils'

const useMyHook = ({ foo, bar }) => useMemo(() => {
   return myHeavyCalculation(foo, bar)
}, [foo, bar])
```

output

```
import { c as _c } from "react/compiler-runtime";
import { myHeavyCalculation } from "./utils";

const useMyHook= (t0) => {
   const $ = _c(3);
   const { foo, bar } = t0;
   let t1;
   if ($[0] !== bar || $[1] !== foo) {
      t1 = myHeavyCalculation(foo, bar);
      $[0] = bar;
      $[1] = foo;
      $[2] = t1;
   } else {
      t1 = $[2];
   }
   return t1;
};
```

Compiler не знает, что возвращает функция

input

```
import { myHeavyCalculation } from './utils'


const myHeavyCalculation = ( foo, bar ) => {
   return `${foo} ${bar}`
}


const useMyHook = ({ foo, bar }) => useMemo(() => {
   return myHeavyCalculation(foo, bar)
}, [foo, bar])
```

output

```
import { c as _c } from "react/compiler-runtime";

const myHeavyCalculation = (foo, bar) => {
  return `${foo} ${bar}`;
};

const useMyHook = (t0) => {
   const $ = _c(3);
   const { foo, bar } = t0;
   let t1;
   if ($[0] !== bar || $[1] !== foo) {
      t1 = myHeavyCalculation(foo, bar);
      $[0] = bar;
      $[1] = foo;
      $[2] = t1;
   } else {
      t1 = $[2];
   }
   return t1;
};
```

У компонентов тоже

input

```
const myCalculation = ( foo, bar ) => {
   return `${foo} ${bar}`
}


const MyComponent = ({ foo, bar }) => {
   const value = myCalculation(foo, bar)


   return <span>{value}</span>
}
```

output

```
import { c as _c } from "react/compiler-runtime";

const myCalculation = (foo, bar) => {
  return `${foo} ${bar}`;
};

const MyComponent = (t0) => {
   const $ = _c(5);
   const { foo, bar } = t0;
   let t1;
   if ($[0] !== bar || $[1] !== foo) {
      t1 = myCalculation(foo, bar);
      $[0] = bar;
      $[1] = foo;
      $[2] = t1;
   } else {
      t1 = $[2];
   }
...
```

У компонентов тоже

input

```
const myCalculation = (foo, bar) => {
   return { foo, bar }
}


const MyComponent = ({ foo, bar }) => {
   const value = '' + myCalculation(foo, bar)


   return <span>{value}</span>
}
```

output

```
const myCalculation = (foo, bar) => {
   return { foo, bar };
};

const MyComponent = (t0) => {
   const $ = _c(2);
   const { foo, bar } = t0;
   const value = "" + myCalculation(foo, bar);
   let t1;
   if ($[0] !== value) {
      t1 = <span>{value}</span>;
      $[0] = val;
      $[1] = t1;
   } else {
      t1 = $[1];
   }
   return t1;
};
```

React Compiler мемоизирует иначе

- Не мемоизирует примитивы
- Мемоизирует ссылочные типы и результаты функций

Side-effects

input

```
const useMyHook = (foo,bar) => useMemo(() => {
  if (foo.someFlag) {
    console.log(123)
    return someFunction(foo, bar)
  }
  return bar
}, [foo, bar])
```

output

```
const useMyHook = (foo, bar) => {
  const $ = _c(3);
  let t0;
  bb0: {
    if (foo.someFlag) {
      console.log(123);
      let t1;
      if ($[0] !== bar || $[1] !== foo) {
        t1 = someFunction(foo, bar);
        $[0] = bar;
        $[1] = foo;
        $[2] = t1;
      } else {
        t1 = $[2];
      }
      t0 = t1;
      break bb0;
    }
    t0 = bar;
  }
  return t0;
};
```

React Compiler мемоизирует иначе

- Не мемоизирует примитивы
- Мемоизирует ссылочные типы и результаты функций
- Игнорирует наличие Side-эффектов

Добавление props

```
const Component = ({ item }) => {

  const status = getStatus(item)

  return (
    <span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(4);
  const { item } = t0;
  let t1;
  let t2;
  if ($[0] !== item) {
    const status = getStatus(item);
    t1 = status;
    t2 = i18next.t(status);
...
```

Добавление props

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    }
...
```

А так – работает))

```jsx
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
      <span>{name}</span>
    </span>
  )
}
```

```jsx
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item) {
    const status = getStatus(item);
    t1 = i18next.t(status);
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
  }
...
```

1. getStatus может вернуть ссылочный тип

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    }
...
```

Настройка Environment

```
export default defineConfig({
  plugins: [
    react({
      babel: {
        plugins: [
          ["babel-plugin-react-compiler", {
            compilation_mode: "annotation",
            panic_threshold: "all_errors",
            environment: {

            }}],
          ...otherPlugins
        ],
      },
    })
  ]
})
```

vite.config.js

Настройка Environment

```
export default defineConfig({
  plugins: [
    react({
      babel: {
        plugins: [
          ["babel-plugin-react-compiler", {
            compilation_mode: "annotation",
            panic_threshold: "all_errors",
            environment: {
              ...
            }}],
          ...otherPlugins
        ],
      },
    })
  ]
})
```

Ещё 40+ флагов

vite.config.js

moduleTypeProvider

```
environment: {
   moduleTypeProvider(moduleName) {
      if (moduleName == './utils') {
         return {
            kind: "object",
               properties: {
                  getStatus: {
                     kind: "function",
                     positionalParams: "read",
                     restParam: "read",
                     calleeEffect: "read",
                     returnType: { kind: "type", name: "Any"},
                     returnValueKind: "mutable"
                  }
            }
         }
}}}
```

vite.config.js

moduleTypeProvider

```
environment: {
  moduleTypeProvider(moduleName) {
    if (moduleName == './utils') {
      return {
        kind: "object",
          properties: {
            getStatus: {
              kind: "function",
              positionalParams: "read",
              restParam: "read",
              calleeEffect: "read",
              returnType: { kind: "type", name: "Any"},
              returnValueKind: "mutable"
            }
        }
      }
}}}
```

vite.config.js

moduleTypeProvider

```
environment: {
   moduleTypeProvider(moduleName) {
      if (moduleName == './utils') {
         return {
            kind: "object",
               properties: {
                  getStatus: {
                     kind: "function",
                     positionalParams: "read",
                     restParam: "read",
                     calleeEffect: "read",
                     returnType: { kind: "type", name: "Any"},
                     returnValueKind: "mutable"
                  }
            }
         }
}}}
```

Только для указанного источника импорта

vite.config.js

moduleTypeProvider

```
environment: {
   moduleTypeProvider(moduleName) {
      if (moduleName == './utils') {
         return {
            kind: "object",
               properties: {
                  getStatus: {
                     kind: "function",
                     positionalParams: "read",
                     restParam: "read",
                     calleeEffect: "read",
                     returnType: { kind: "type", name: "Any"},
                     returnValueKind: "mutable"
                  }
            }
         }
}}}
```

vite.config.js

- Только для указанного источника импорта
- Перебивает настройки самого компайлера

moduleTypeProvider

```
environment: {
   moduleTypeProvider(moduleName) {
      if (moduleName == './utils') {
         return {
            kind: "object",
               properties: {
                  getStatus: {
                     kind: "function",
                     positionalParams: "read",
                     restParam: "read",
                     calleeEffect: "read",
                     returnType: { kind: "type", name: "Any"},
                     returnValueKind: "mutable"
                  }
            }
         }
}}}
```

- Только для указанного источника импорта
- Перебивает настройки самого компайлера
- В rust-версии такой настройки нет

vite.config.js

Добавление props

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    }
...
```

Можно поправить так))

```jsx
const Component = ({ item, name }) => {

  const status = String(getStatus(item))

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```jsx
import { c as _c } from "react/compiler-runtime";
const Component = (t0) => {
  const $ = _c(9);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item) {
    t1 = getStatus(item);
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  const status = String(t1);
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
  }
```

Reactive Function / Reactive Scope

Reactive Function

```
import { c as _c } from "react/compiler-runtime";
export function MyApp(props) {
  const $ = _c(2);
  let t0;
  if ($[0] !== props.name) {
    t0 = <div>Hello {props.name}</div>;
    $[0] = props.name;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
  return t0;
}
```

Reactive Scope

```
import { c as _c } from "react/compiler-runtime";
export function MyApp(props) {
  const $ = _c(2);
  let t0;
  if ($[0] !== props.name) {
    t0 = <div>Hello {props.name}</div>;
    $[0] = props.name;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
  return t0;
}
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    } else {
      t2 = $[4];
    }
    t1 = (
      <span>
        {t2}
        {i18next.t(status)}
      </span>
    );
    $[0] = item;
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    } else {
      t2 = $[4];
    }
    t1 = (
      <span>
        {t2}
        {i18next.t(status)}
      </span>
    );
    $[0] = item;
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    } else {
      t2 = $[4];
    }
    t1 = (
      <span>
        {t2}
        {i18next.t(status)}
      </span>
    );
    $[0] = item;
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(5);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item || $[1] !== name) {
    const status = getStatus(item);
    let t2;
    if ($[3] !== name) {
      t2 = <span>{name}</span>;
      $[3] = name;
      $[4] = t2;
    } else {
      t2 = $[4];
    }
    t1 = (
      <span>
        {t2}
        {i18next.t(status)}
      </span>
    );
    $[0] = item;
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  return (
    <span>
      <span>{name}</span>
      {i18next.t(status)}
    </span>
  )
}
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)
  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {i18next.t(statusText)}
    </span>
  )
}
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)
  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)
  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item) {
    const status = getStatus(item);
    t1 = i18next.t(status);
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  const statusText = t1;
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
  }
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)
  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item) {
    const status = getStatus(item);
    t1 = i18next.t(status);
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  const statusText = t1;
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
  }
```

Reactive Scope

```
const Component = ({ item, name }) => {

  const status = getStatus(item)
  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item) {
    const status = getStatus(item);
    t1 = i18next.t(status);
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  const statusText = t1;
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
  }
```

React Compiler мемоизирует иначе

- Не мемоизирует примитивы
- Мемоизирует ссылочные типы и результаты функций
- Игнорирует наличие Side-эффектов
- Сохраняет порядок инструкций

Если порядок так важен...

```
const Component = ({ item, name }) => {

  const status = getStatus(item)
  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  let t1;
  if ($[0] !== item) {
    const status = getStatus(item);
    t1 = i18next.t(status);
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  const statusText = t1;
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
  }
```

Если порядок так важен...

```
const Component = ({ item, name }) => {
  const status = getStatus(item)

  useEffect(someEffect, [item])

  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

???

Если порядок так важен...

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  useEffect(someEffect, [item])

  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  const status = getStatus(item);
  let t1;
  if ($[0] !== item) {
    t1 = [item];
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  useEffect(someEffect, t1);
  const statusText = i18next.t(status);
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
```

React Compiler мемоизирует иначе

- Не мемоизирует примитивы
- Мемоизирует ссылочные типы и результаты функций
- Игнорирует наличие Side-эффектов
- Сохраняет порядок инструкций
- Следит за Rules of React

Если порядок так важен...

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  useEffect(someEffect, [item])

  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  const status = getStatus(item);
  let t1;
  if ($[0] !== item) {
    t1 = [item];
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  useEffect(someEffect, t1);
  const statusText = i18next.t(status);
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
```

А вдруг i18next.t тоже мутирует?

```
const Component = ({ item, name }) => {

  const status = getStatus(item)

  useEffect(someEffect, [item])

  const statusText = i18next.t(status)
  return (
    <span>
      <span>{name}</span>
      {statusText}
    </span>
  )
}
```

```
const Component = (t0) => {
  const $ = _c(7);
  const { item, name } = t0;
  const status = getStatus(item);
  let t1;
  if ($[0] !== item) {
    t1 = [item];
    $[0] = item;
    $[1] = t1;
  } else {
    t1 = $[1];
  }
  useEffect(someEffect, t1);
  const statusText = i18next.t(status);
  let t2;
  if ($[2] !== name) {
    t2 = <span>{name}</span>;
    $[2] = name;
    $[3] = t2;
  } else {
    t2 = $[3];
```

Rules of React важнее мемоизации

```
const Component = ({ item, name }) => {

   const status = getStatus(item)

   useEffect(someEffect, [item])

   const statusText = i18next.t(status)
   return (
      <span>
         <span>{name}</span>
         {statusText}
      </span>
   )
}
```

```
const Component = (t0) => {
   const $ = _c(9);
   const { item, name } = t0;
   let t1;
   if ($[0] !== item) {
      t1 = getStatus(item);
      $[0] = item;
      $[1] = t1;
   } else {
      t1 = $[1];
   }
   const status = t1;
   let t2;
   if ($[2] !== item) {
      t2 = [item];
      $[2] = item;
      $[3] = t2;
   } else {
      t2 = $[3];
   }
   useEffect(someEffect, t2);
   const statusText = status;
   let t3;
   if ($[4] !== name) {
```

Custom Hooks

React Use

NPM V1.0.17 LICENSE MIT JOIN THE COMMUNITY

React Use delivers **production-ready hooks** that solve real-world problems. Built with **TypeScript-first** approach, **SSR compatibility**, and **tree-shaking optimization** - everything you need to build modern React applications. Improve your react applications with our library 📦 designed for comfort and speed.

Documentation

Visit https://reactuse.org to view the full documentation.

Getting Started

```
npm install @siberiacancode/reactuse
```

```
import { useCounter } from '@siberiacancode/reactuse';

const App = () => {
  const counter = useCounter(0);

  return (
    <div>
      <h1>Count: {counter.value}</h1>
      <button onClick={() => counter.inc()}>+1</button>
      <button onClick={() => counter.dec()}>-1</button>
    </div>
  );
};
```

useBreakpoints

```
const useBreakpoints = ( breakpoints, strategy ) => {









}
```

useBreakpoints

```
const useBreakpoints = ( breakpoints, strategy ) => {
   const renderer = useRenderer()

   useEffect(() => {
      const onResize = rerender;
      window.addEventListener('resize', onResize);
      return () => {
         window.removeEventListener('resize', onResize);
      };
   }, []);




}
```

useBreakpoints

```
const useBreakpoints = ( breakpoints, strategy ) => {
   const renderer = useRenderer()

   useEffect(() => {
      const onResize = rerender;
      window.addEventListener('resize', onResize);
      return () => {
         window.removeEventListener('resize', onResize);
      };
   }, []);

   const greaterOrEqual = (breakpoint) => match(`(min-width: ${getValue(breakpoint)})`);

   return {
      greaterOrEqual,
      ...
   }
}
```

useBreakpoints

```
const match = (query) => window.matchMedia(query).matches;


const useBreakpoints = ( breakpoints, strategy ) => {
  const renderer = useRenderer()

  useEffect(() => {
    const onResize = rerender;
    window.addEventListener('resize', onResize);
    return () => {
      window.removeEventListener('resize', onResize);
    };
  }, []);

  const greaterOrEqual = (breakpoint) => match(`(min-width: ${getValue(breakpoint)})`);

  return {
    greaterOrEqual,
    ...
  }
}
```

useRenderInfo

```
export const useRenderInfo = (name, log) => {
  const renderInfoRef = useRef({
    component: name,
    renders: 0,
    timestamp: Date.now(),
    sinceLast: 0
  });

  renderInfoRef.current.renders += 1;
  renderInfoRef.current.timestamp = Date.now();


  return renderInfoRef.current;
};
```

Компонент с useRenderInfo

```
export const MyComponent2 = () => {
  const [count, setCount] = useState(0)
  const info = useRenderInfo("Counter")


  return (
    <>
      <InfoView info={info} />
      <Button onClick={() => setCount(count + 1)} />
    </>
  )
};
```

```
export const MyComponent2 = () => {
  const $ = _c(7);
  const [count, setCount] = useState(0);
  const info = useRenderInfo("Counter");
  let t0;
  if ($[0] !== info) {
    t0 = <InfoView info={info} />;
    $[0] = info;
    $[1] = t0;
  } else {
    t0 = $[1];
  }
...
```

React Compiler мемоизирует иначе

Для чего мы используем
мемоизацию

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

Избегаем перерендера из-за ссылочных типов

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков
- Избегаем повтора сложных вычислений

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков
- Избегаем повтора сложных вычислений

В чём задача React Compiler

Copy page

React Compiler

Introduction

Learn what React Compiler does and how it automatically optimizes your React application by handling memoization for you, eliminating the need for manual `useMemo`, `useCallback`, and `React.memo`.

Installation

Get started with installing React Compiler and learn how to configure it with your build tools.

React Compiler мемоизирует иначе

- Не мемоизирует примитивы
- Мемоизирует ссылочные типы и результаты функций
- Игнорирует наличие Side-эффектов
- Сохраняет порядок инструкций
- Следит за Rules of React

React Compiler мемоизирует иначе

- Не мемоизирует примитивы
- Мемоизирует ссылочные типы и результаты функций
- Игнорирует наличие Side-эффектов
- Сохраняет порядок инструкций
- Следит за Rules of React

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков
- Избегаем повтора сложных вычислений

В чём задача React Compiler

- Стабильность ссылочных типов везде

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков
- Избегаем повтора сложных вычислений

В чём задача React Compiler

- Стабильность ссылочных типов везде
- Контроль выполнения Rules of React

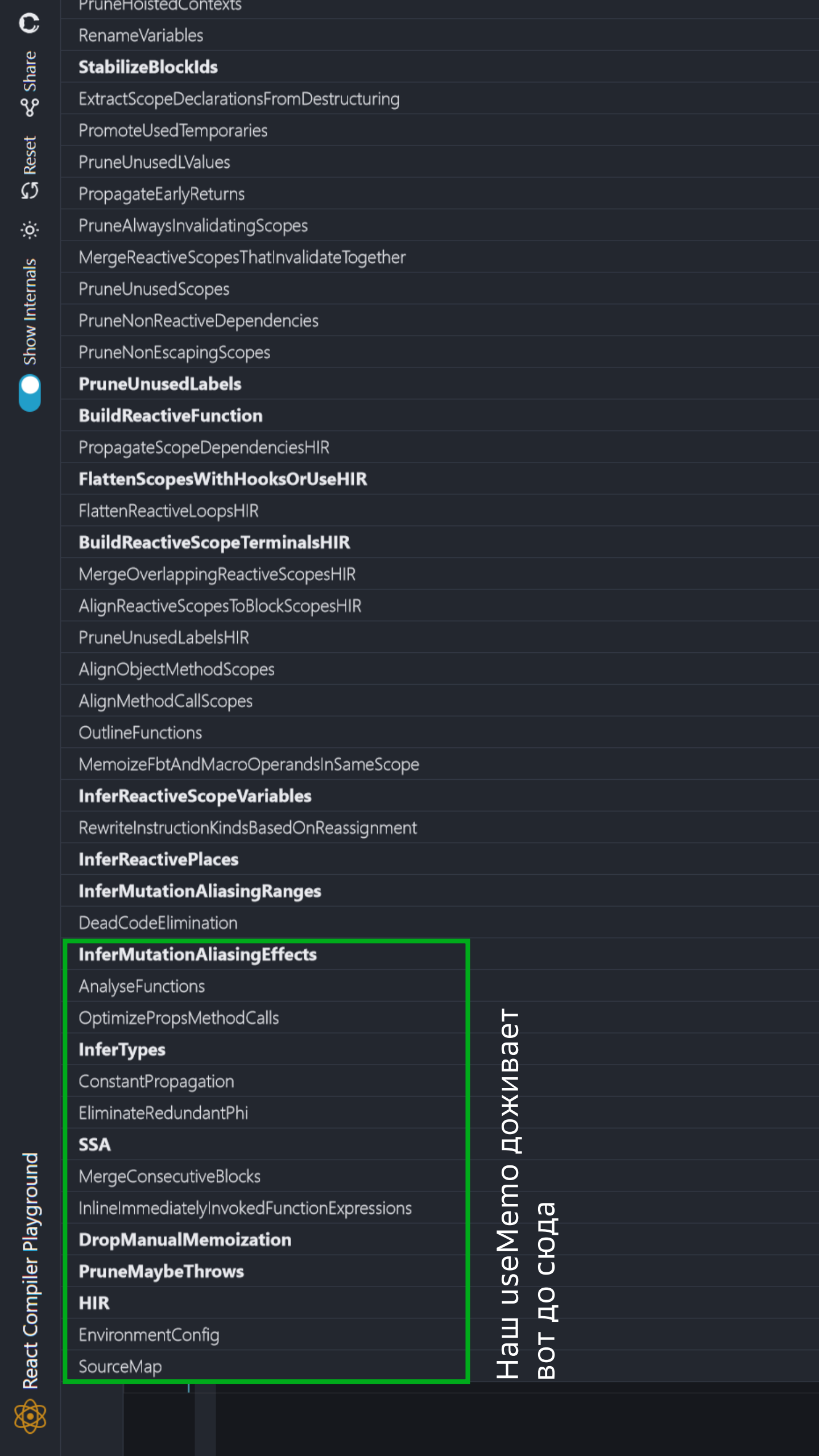
React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков
- Избегаем повтора сложных вычислений

В чём задача React Compiler

- Стабильность ссылочных типов везде
- Контроль выполнения Rules of React
- Ничего не знает о сложности вычислений


PruneHoistedContexts
RenameVariables
StabilizeBlockIds
ExtractScopeDeclarationsFromDestructuring
PromoteUsedTemporaries
PruneUnusedLValues
PropagateEarlyReturns
PruneAlwaysInvalidatingScopes
MergeReactiveScopesThatInvalidateTogether
PruneUnusedScopes
PruneNonReactiveDependencies
PruneNonEscapingScopes
PruneUnusedLabels
BuildReactiveFunction
PropagateScopeDependenciesHIR
FlattenScopesWithHooksOrUseHIR
FlattenReactiveLoopsHIR
BuildReactiveScopeTerminalsHIR
MergeOverlappingReactiveScopesHIR
AlignReactiveScopesToBlockScopesHIR
PruneUnusedLabelsHIR
AlignObjectMethodScopes
AlignMethodCallScopes
OutlineFunctions
MemoizeFbtAndMacroOperandsInSameScope
InferReactiveScopeVariables
RewriteInstructionKindsBasedOnReassignment
InferReactivePlaces
InferMutationAliasingRanges
DeadCodeElimination
InferMutationAliasingEffects
AnalyseFunctions
OptimizePropsMethodCalls
InferTypes
ConstantPropagation
EliminateRedundantPhi
SSA
MergeConsecutiveBlocks
InlineImmediatelyInvokedFunctionExpressions
DropManualMemoization
PruneMaybeThrows
HIR
EnvironmentConfig
SourceMap
Share
Reset
Show Internals
React Compiler Playground
Наш useMemo доживает
вот до сюда

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков
- Избегаем повтора сложных вычислений

В чём задача React Compiler

- Стабильность ссылочных типов везде
- Контроль выполнения Rules of React
- Ничего не знает о сложности вычислений

React Compiler мемоизирует иначе

Для чего мы используем мемоизацию

- Избегаем перерендера из-за ссылочных типов
- Управляем зависимостями хуков
- Избегаем повтора сложных вычислений
- Ничего не знаем о Rules of React

В чём задача React Compiler

- Стабильность ссылочных типов везде
- Контроль выполнения Rules of React
- Ничего не знает о сложности вычислений

Всё, что нужно знать о React Compiler

Всё, что нужно знать о React Compiler

> Не используем настройки по умолчанию

Настройки логирования

```
{
    compilation_mode: "annotation",
    panic_threshold: "all_errors",
    logger: {
      logEvent: (filename, event) => {
        // Лог результата компиляции
      }
    }
}
```

vite.config.js

Пофайловый лог со статусом


```
/Users/anton/projects/compiler-test/src/metric-cell.tsx {
  kind: 'CompileSuccess',
  fnLoc: SourceLocation {
    start: Position { line: 10, column: 5, index: 166 },
    end: Position { line: 26, column: 1, index: 539 },
    filename: undefined,
    identifierName: undefined
  },
  fnName: null,
  memoSlots: 10,
  memoBlocks: 4,
  memoValues: 4,
  prunedMemoBlocks: 0,
  prunedMemoValues: 0
}
```

И ошибки тоже


```
/Users/16689790/projects/compiler-test/src/metric-cell.tsx {
  kind: 'CompileError',
  fnLoc: SourceLocation {
    start: Position { line: 10, column: 5, index: 166 },
    end: Position { line: 26, column: 1, index: 532 },
    filename: undefined,
    identifierName: undefined
  },
  detail: _CompilerDiagnostic {
    options: {
      category: 'PreserveManualMemo',
      severity: 'CannotPreserveMemoization',
      reason: 'Compilation skipped because existing memoization could not be preserved',
      description: 'React Compiler has skipped optimizing this component because the existing manual memoization could not be preserved. The inferred dependencies did not match the manually specified dependencies, which could cause the value to change more or less frequently than expected. The inferred dependency was `product`, but the source dependencies were []. Inferred dependency not present in source.',
      suggestions: null,
      details: [Array]
    }
  }
```

console

Всё, что нужно знать о React Compiler

- Не используем настройки по умолчанию
- Продолжаем писать useMemo

Всё, что нужно знать о React Compiler

- Не используем настройки по умолчанию
- Продолжаем писать useMemo
- Проверяем результат компиляции при каждом изменении

poteto [compiler] Init react-mcp-server (#32859) 0807592 · last year His

Preview Code Blame 22 lines (17 loc) · 752 Bytes Raw

React MCP Server (experimental)

An experimental MCP Server for React.

Development

First, add this file if you're using Claude Desktop: `code ~/Library/Application\ Support/Claude/claude_desktop_config.json`. Copy the absolute path from `which node` and from `react/compiler/react-mcp-server/dist/index.js` and paste, for example:

```
{
  "mcpServers": {
    "react": {
      "command": "/Users/<username>/.asdf/shims/node",
      "args": [
        "/Users/<username>/code/react/compiler/packages/react-mcp-server/dist/index.js"
      ]
    }
  }
}
```

Next, run `yarn workspace react-mcp-server watch` from the `react/compiler` directory and make changes as needed. You will need to restart Claude everytime you want to try your changes.

Input

```
1  const useCustomState = (input) => {
2    return useState(input)
3  }
```

- Output

```
const useCustomState = (input) => {
  return useState(input);
};
```

SourceMap
EnvironmentConfig
HIR
PruneMaybeThrows
DropManualMemoization
InlineImmediatelyInvokedFunctionExpressions
MergeConsecutiveBlocks
SSA
EliminateRedundantPhi
ConstantPropagation

Всё, что нужно знать о React Compiler

- Не используем настройки по умолчанию
- Продолжаем писать useMemo
- Проверяем результат компиляции при каждом изменении
- Компактно используем переменные

Всё, что нужно знать о React Compiler

- Не используем настройки по умолчанию
- Продолжаем писать useMemo
- Проверяем результат компиляции при каждом изменении
- Компактно используем переменные
- Не верим документации