9-1 创建 Fiber 与 FiberRoot

环境

本章节带大家手写 my-mini-react，实现**页面的初次渲染**，先配置下环境 ~

react-reconciler 依赖 shared 和 scheduler，react-dom 依赖 react-reconciler，所以不要忘记安装依赖 ~

安装方式：在根目录下执行 `pnpm i`：

```
gaoshaoyun@gaoshaoyundeMacBook-Pro my-mini-react % pnpm i
```

packages/react-reconciler/package.json

JSON

```json
{
  "name": "react-reconciler",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
```

```
  "license": "ISC",
  "dependencies": {
    "shared": "workspace:*",
    "scheduler": "workspace:*"
  }
}
```

packages/react-dom/package.json

```JSON
{
  "name": "react-dom",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react-reconciler": "workspace:*"
  }
}
```

Fiber

类型

React 源码 js，flow 做类型标记。

packages/react-reconciler/src/ReactFiber.ts

```JavaScript
export type Fiber = {
  // 标记fiber的类型，即描述的组件类型，如原生标签、函数组件、类组件、Fragment等
```

```
  tag: WorkTag;

  // 标记组件在当前层级下的的唯一性
  key: null | string;

  // 组件类型
  elementType: any;

  // 标记组件类型，如果是原生组件，这里是字符串，如果是函数组件，这里是函数，如果是
  type: any;

  // 如果组件是原生标签，DOM；如果是类组件，是实例；如果是函数组件，是null
  // 如果组件是原生根节点，stateNode存的是FiberRoot.  HostRoot=3
  stateNode: any;

  // 父fiber
  return: Fiber | null;

  // 单链表结构
  // 第一个子fiber
  child: Fiber | null;
  // 下一个兄弟fiber
  sibling: Fiber | null;
  // 记录了节点在当前层级中的位置下标，用于diff时候判断节点是否需要发生移动
  index: number;

  // 新的props
  pendingProps: any;
  // 上一次渲染时使用的 props
  memoizedProps: any;

  // 不同的组件的 memoizedState 存储不同
  // 函数组件 hook0
  // 类组件 state
  // HostRoot RootState
  memoizedState: any;

  // Effect
  flags: Flags;

  // 缓存fiber
```

```
  alternate: Fiber | null;
};
```

生成 Fiber

这里创建了几个不同的创建 Fiber 方法：

packages/react-reconciler/src/ReactInternalTypes.ts

JavaScript

```
import { ReactElement } from "shared/ReactTypes";
import { NoFlags } from "./ReactFiberFlags";
import { Fiber } from "./ReactInternalTypes";
import { isStr } from "shared/utils";
import { HostComponent } from "./ReactWorkTags";
import { IndeterminateComponent, WorkTag } from "./ReactWorkTags";

// 创建一个fiber
export function createFiber(
  tag: WorkTag,
  pendingProps: any,
  key: null | string
): Fiber {
  return new FiberNode(tag, pendingProps, key);
}

function FiberNode(tag: WorkTag, pendingProps: any, key: null | string
  // 标记组件类型
  this.tag = tag;
  // 定义组件在当前层级下的唯一性
  this.key = key;
  // 组件类型
  this.elementType = null;
  // 组件类型
  this.type = null;
  // 不同的组件的  stateNode 定义也不同
  // 原生标签：string
  // 类组件：实例
  this.stateNode = null;
```

```
  // Fiber
  this.return = null;
  this.child = null;
  this.sibling = null;
  // 记录了节点在兄弟节点中的位置下标，用于diff时候判断节点是否需要发生移动
  this.index = 0;

  this.pendingProps = pendingProps;
  this.memoizedProps = null;

  // 不同的组件的 memoizedState 指代也不同
  // 函数组件 hook0
  // 类组件 state
  this.memoizedState = null;

  // Effects
  this.flags = NoFlags;

  // 缓存fiber
  this.alternate = null;
}

// 根据 ReactElement 创建Fiber
export function createFiberFromElement(element: ReactElement) {
  const { type, key } = element;
  const pendingProps = element.props;
  const fiber = createFiberFromTypeAndProps(type, key, pendingProps);
  return fiber;
}

// 根据 TypeAndProps 创建fiber
export function createFiberFromTypeAndProps(
  type: any,
  key: null | string,
  pendingProps: any
) {
  let fiberTag: WorkTag = IndeterminateComponent;
  if (isStr(type)) {
    // 原生标签
    fiberTag = HostComponent;
  }
```

```
  const fiber = createFiber(fiberTag, pendingProps, key);
  fiber.elementType = type;
  fiber.type = type;
  return fiber;
}
```

Flags

二进制常量，方便叠加组合。

JavaScript

```
export type Flags = number;

export const NoFlags = /*                      */ 0b0000000000000000000
export const Placement = /*                    */ 0b0000000000000000000
export const Update = /*                       */ 0b0000000000000000000
export const ChildDeletion = /*                */ 0b0000000000000000000
```

WorkTag

组件类型。

packages/react-reconciler/src/ReactWorkTags.ts

JavaScript

```
export type WorkTag =
  | 0
  | 1
  | 2
  | 3
  | 4
  | 5
  | 6
  | 7
  | 8
  | 9
  | 10
```

```
  | 11
  | 12
  | 13
  | 14
  | 15
  | 16
  | 17
  | 18
  | 19
  | 20
  | 21
  | 22
  | 23
  | 24
  | 25
  | 26
  | 27;

export const FunctionComponent = 0;
export const ClassComponent = 1;
export const IndeterminateComponent = 2; // Before we know whether it
export const HostRoot = 3; // Root of a host tree. Could be nested ins
export const HostPortal = 4; // A subtree. Could be an entry point to
export const HostComponent = 5;
export const HostText = 6;
export const Fragment = 7;
export const Mode = 8;
export const ContextConsumer = 9;
export const ContextProvider = 10;
export const ForwardRef = 11;
export const Profiler = 12;
export const SuspenseComponent = 13;
export const MemoComponent = 14;
export const SimpleMemoComponent = 15;
export const LazyComponent = 16;
export const IncompleteClassComponent = 17;
export const DehydratedFragment = 18;
export const SuspenseListComponent = 19;
export const ScopeComponent = 21;
export const OffscreenComponent = 22;
export const LegacyHiddenComponent = 23;
```

```
export const CacheComponent = 24;
export const TracingMarkerComponent = 25;
export const HostHoistable = 26;
export const HostSingleton = 27;
```

FiberRoot

类型

packages/react-reconciler/src/ReactInternalTypes.ts

JavaScript

```
export type Container = Element | Document | DocumentFragment;

export type FiberRoot = {
  containerInfo: Container;
  current: Fiber;
  // 一个准备提交 work-in-progress， HostRoot
  finishedWork: Fiber | null;
};
```

生成 FiberRoot

JavaScript

```
import { createFiber } from "./ReactFiber";
import { Container, Fiber, FiberRoot } from "./ReactInternalTypes";
import { HostRoot } from "./ReactWorkTags";

export function createFiberRoot(containerInfo: Container): FiberRoot {
  const root: FiberRoot = new FiberRootNode(containerInfo);
  const uninitializedFiber: Fiber = createFiber(HostRoot, null, null);
  root.current = uninitializedFiber;
  uninitializedFiber.stateNode = root;
  return root;
}
```

```
export function FiberRootNode(containerInfo) {
  this.containerInfo = containerInfo;
  this.current = null;
  this.finishedWork = null;
}
```