React源码解析01

课堂主题

深入理解React原理

资源

1. React中文网
2. React源码

课堂目标

1. 掌握createElement/Component/render三个核心api

知识点

顶层目录

当克隆 React 仓库之后，你们将看到一些顶层目录：

- **源代码在/packages/, 每个包的 src 子目录是你最需要花费精力的地方。**
- `fixtures` 包含一些给贡献者准备的小型 React 测试项目。
- `build` 是 React 的输出目录。源码仓库中并没有这个目录，但是它会在你克隆 React 并且第一次构建它之后出现。

React核心api

react

```
const React = {
  Children: {
    map,
    forEach,
    count,
    toArray,
    only,
  },

  createRef,
  Component,
  PureComponent,

  createContext,
  forwardRef,
  lazy,
  memo,

  useCallback,
  useContext,
  useEffect,
  useImperativeHandle,
  useDebugValue,
  useLayoutEffect,
  useMemo,
```

```
  useReducer,
  useRef,
  useState,

  Fragment: REACT_FRAGMENT_TYPE,
  StrictMode: REACT_STRICT_MODE_TYPE,
  Suspense: REACT_SUSPENSE_TYPE,

  createElement: __DEV__ ?
createElementWithValidation : createElement,
  cloneElement: __DEV__ ?
cloneElementWithValidation : cloneElement,
  createFactory: __DEV__ ?
createFactoryWithValidation : createFactory,
  isValidElement: isValidElement,

  version: ReactVersion,

  unstable_ConcurrentMode:
REACT_CONCURRENT_MODE_TYPE,
  unstable_Profiler: REACT_PROFILER_TYPE,

  __SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_
FIRED: ReactSharedInternals,
};

// Note: some APIs are added with feature
flags.
// Make sure that stable builds for open source
```

```
// don't modify the React object to avoid
deopts.
// Also let's not expose their names in stable
builds.

if (enableStableConcurrentModeAPIs) {
  React.ConcurrentMode =
REACT_CONCURRENT_MODE_TYPE;
  React.Profiler = REACT_PROFILER_TYPE;
  React.unstable_ConcurrentMode = undefined;
  React.unstable_Profiler = undefined;
}

export default React;
```

核心精简后：

```
const React = {
    createElement,
    Component
}
```

react-dom 主要是render逻辑

最核心的api：

React.createElement：创建虚拟DOM

React.Component：实现自定义组件

ReactDOM

`render()`

```
ReactDOM.render(element, container[, callback])
```

在提供的 `container` 里渲染一个 React 元素，并返回对该组件的引用（或者针对无状态组件返回 `null`）。

当首次调用时，容器节点里的所有 DOM 元素都会被替换，后续的调用则会使用 React 的 DOM 差分算法（DOM diffing algorithm）进行高效的更新。

如果提供了可选的回调函数，该回调将在组件被渲染或更新之后被执行。

> 注意：
>
> 使用 `ReactDOM.render()` 对服务端渲染容器进行 hydrate 操作的方式已经被废弃，并且会在 React 17 被移除。作为替代，请使用 `hydrate()`。

JSX

[在线尝试]

1. 什么是JSX

   语法糖

   React 使用 JSX 来替代常规的 JavaScript。

   JSX 是一个看起来很像 XML 的 JavaScript 语法扩展。

2. 为什么需要JSX

   - 开发效率：使用 JSX 编写模板简单快速。
   - 执行效率：JSX编译为 JavaScript 代码后进行了优化，执行更快。
   - 类型安全：在编译过程中就能发现错误。

3. 原理：babel-loader会预编译JSX为React.createElement(...)

4. 与vue的异同：

   - react中虚拟dom+jsx的设计一开始就有，vue则是演进过程中才出现的
   - jsx本来就是js扩展，转义过程简单直接的多；vue把template编译为render函数的过程需要复杂的编译器转换字符串-ast-js函数字符串

JSX预处理前：

REACT 编辑器 显示JSX

```
class App extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}, I am {2 + 2} years old
      </div>
    )
  }
}

ReactDOM.render(
  <App name="React" />,
  mountNode
)
```

JSX预处理后：

REACT 编辑器 显示JSX

```
class App extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name,
      ", I am ",
      2 + 2,
      " years old"
    )
  }
}

ReactDOM.render(React.createElement(App, { name: "React" }),
mountNode)
```

使用自定义组件的情况:

```
import React, { Component } from "react";
import ReactDOM from "react-dom";
import "./index.css";

function FuncCmp(props) {
  return <div>name: {props.name}</div>;
}

class ClassCmp extends Component {
  render() {
    return <div>name: {this.props.name}</div>;
  }
}

const jsx = (
  <div>
    <p>我是内容</p>
    <FuncCmp name="我是function组件" />
    <ClassCmp name="我是class组件" />
  </div>
);

console.log("jsx", jsx);
ReactDOM.render(jsx,
document.getElementById("root"));
```

build后

```
function FuncCmp(props) {
  return React.createElement(
    "div",
    null,
    "name: ",
    props.name
  );
}
class ClassCmp extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "name: ",
      this.props.name
    );
  }
}
let jsx = React.createElement(
  "div",
  null,
  "    ",
  React.createElement(
    "div",
    { className: "border" },
    "我是内容"
  ),
  "    ",
```

```
  React.createElement(FuncCmp, { name: "我是function组件" }),
  "    ",
  React.createElement(ClassCmp, { name: "我是class组件" }),
  "  "
);

ReactDOM.render(jsx,
document.getElementById('root'));
```

实现三大接口：React.createElement, React.Component, ReactDom.render

CreateElement

将传入的节点定义转换为vdom。

src/index.js

```
// import React, { Component } from "react";
// import ReactDOM from "react-dom";

import React from "./kreact/";
import ReactDOM from "./kreact/ReactDOM";
```

```
import "./index.css";

function FuncCmp(props) {
  return <div>name: {props.name}</div>;
}

class ClassCmp extends React.Component {
  constructor(props) {
    super(props);
    this.state = { counter: 0 };
  }
  clickHandle = () => {
    console.log("clickHandle");
  };
  render() {
    const { counter } = this.state;
    return (
      <div>
        name: {this.props.name}
        <p>counter: {counter}</p>
        <button onCclick={this.clickHandle}>点
击</button>
        {[0, 1, 2].map(item => {
          return <FuncCmp name={"我是function组
件" + item} key={item} />;
        })}
      </div>
    );
  }
```

```
}

let jsx = (
  <div>
    <div className="border">我是内容</div>
    <FuncCmp name="我是function组件" />
    <ClassCmp name="我是class组件" />
  </div>
);
ReactDOM.render(jsx,
document.getElementById("root"));
```

- 创建./src/kkreact/index.js，它需要包含createElement方法

```
import { Component } from "./Component";

function createElement(type, props,
...children) {
  props.children = children;
  //判断组件类型
  let vtype;
  if (typeof type === "string") {
    // 原生标签
    vtype = 1;
  } else if (typeof type === "function") {
    // 类组件 函数组件
    vtype = type.isReactComponent ? 3 : 2;
```

```
  }
  return {
    vtype,
    type,
    props,
  };
}

const React = {
  createElement,
  Component,
};

export default React;
```

- 修改index.js实际引入kreact，测试

```
import React from "./kreact/";
```

> createElement被调用时会传入标签类型type，标签属性props及若干子元素children

> index.js中从未使用React类或者其任何接口，为何需要导入它?
>
> JSX编译后实际调用React.createElement方法，所以只要出现JSX的文件中都需要导入React类

ReactDOM.render

- 创建react-dom.js
- 需要实现一个render函数，能够将vdom渲染出来，这里先打印vdom结构

```
import { initVnode } from "./virtual-dom";

function render(vnode, container) {
  const node = initVnode(vnode, container);
  container.appendChild(node);
}

export default { render };
```

实现Component

要实现class组件，需要添加Component类，Component.js

```
//import { diff } from "./virtual-dom";

class Component {
  static isReactComponent = {};
  constructor(props) {
    this.props = props;
    this.$cache = {};
    this.state = {};
  }
  setState(nextState) {
```

```
    //this.state = { ...this.state,
...nextState };
    //this.forceUpdate();
  }
  /*forceUpdate() {
    const { $cache: cache } = this;
    const { vnode, node } = cache;
    let newVnode = this.render();
    let newNode = diff(cache, newVnode);
    this.$cache = { ...this.$cache, vnode:
newVnode, node: newNode };
  }*/
}

export default Component;
```

> 浅层封装，setState现在只是一个占位符

组件类型判断

传递给createElememnt的组件有三种组件类型，1：dom组件， 2. class组件， 3. 函数组件，使用vtype属性标识

转换vdom为真实dom

./virtual-dom.js

```
export function initVnode(vnode, container) {
  const { vtype } = vnode;
```

```
  let node = null;
  if (!vtype) {
    node = initTxtNode(vnode, container);
  } else if (vtype === 1) {
    node = initHtmlNode(vnode, container);
  } else if (vtype === 2) {
    //class component
    node = initClassNode(vnode, container);
  } else if (vtype === 3) {
    //function component
    node = initFuncNode(vnode, container);
  }
  return node;
}
function initTxtNode(vnode, container) {
  let node = document.createTextNode(vnode);
  return node;
}

function initHtmlNode(vnode, container) {
  const { type, props } = vnode;
  let node = document.createElement(type);
  const { children, ...rest } = props;
  children.map(item => {
    node.appendChild(initVnode(item, node));
  });
  Object.keys(rest).map(key => {
    if (key === "className") {
      node.setAttribute("class", rest[key]);
```

```
    } else if (key.slice(0, 2) === "on") {
      node.addEventListener("click",
rest[key]);
    }
  });
  return node;
}
function initClassNode(vnode, container) {
  const { type, props } = vnode;
  let component = new type(props);
  let node = initVnode(component.render(),
container);
  let cache = {
    vnode,
    node,
    parentNode: container,
  };
  component.$cache = cache;
  return node; //initVnode(node, container);
}
function initFuncNode(vnode, container) {
  const { type, props } = vnode;
  let node = type(props);
  return initVnode(node, container);
}

/*export function diff(cache, newVnode) {
  const { parentNode, vnode, node } = cache;
```

```
  const newNode = initVnode(newVnode,
parentNode);
  parentNode.replaceChild(newNode, node);
  return newNode;
}*/
```

执行渲染，kreact-dom.js

```
import { initVnode } from "./virtual-dom";

function render(vnode, container) {
  //vnode->node
  const node = initVnode(vnode, container);
  container.appendChild(node);
}

export default { render };
```

总结：

1. webpack+babel编译时，替换JSX为React.createElement(type,props,...children)
2. 所有React.createElement()执行结束后得到一个JS对象即vdom，它能够完整描述dom结构
3. ReactDOM.render(vdom, container)可以将vdom转换为dom并追加到container中
4. 实际上，转换过程需要经过一个diff过程，比对出实际更新补丁操作dom

回顾