React Router V6.4 （history 对象篇）

@稀土掘金技术社区

react router 层级

之前有过基础篇是涉及的底层api，React Router 的架构是针对 web 平台和移动端平台：

- react-router-dom
- react-native

层级

- 底层 api 层：web api 层级
- history 层（是建立在 location web api 和 history web api） 通用封装
- router 层：封装了创建 router 的方法和 router 对象
- react-router 层：正对 React 组件封装了一层，注意这里还不区别平台的
- react-router-dom/react-router-native 层: 平台层（针对不同的平台封装）

history 对象详解

action/path/location

action 表示当前 history 的进行的动作：`POP/PUSH/REPLACE` 三种不同的 action 变体

```ts 复制代码
export enum Action {
  Pop = "POP",
  Push = "PUSH",
  Replace = "REPLACE",
}
```

path 包含 `pathname/search/hash`

```ts 复制代码
export interface Path {
  pathname: string;
  search: string;
  hash: string;
}
```

提供了创建和解析函数：`createPath/parsePath`

location 包含 `path & state/key`

```ts 复制代码
export interface Location extends Path {
  state: any;
  key: string; // 使用 createKey 方法创建
}
```

提供了创建函数 `createLocation`

有了三个对象，就可以开始 history 封装之旅了，

History

- 类型

```ts 复制代码
export interface History {
  readonly action: Action;
  readonly location: Location;
  createHref(to: To): string;
  encodeLocation(to: To): Path;
```

```
  push(to: To, state?: any): void;
  replace(to: To, state?: any): void;
  go(delta: number): void;
  listen(listener: Listener): () => void;
}
```

我们看到 History 中包含了 Action/Location 这两对象，

- craeteHref 创建一个完整的 url 地址
- encodeLocation 正对 location 的解析
- history 的跳转函数 push 取代 pushState, replace 取代 replaceState, 以及 go 函数跳转
- listen 监听当前 location 对象的变化

高层 History 封装

histoy 中包含了三种 History 对象：

- MemoryHistory 对应的创建函数 createMemoryHistory，用于非 dom 环境，react-native 和测试环境
- BrowserHistory 对应的创建函数 createBrowserHistory， 用于现代浏览器由 html5 history api 提供基础
- HashHistory 对应的创建函数 createHashHistory 用于旧款浏览器

MemoryHistory

MemoryHistory 实际开发过程中使用的会很少

比 History 对象多了索引

createBrowserHistory/createHashHistory 函数

createBrowserHistory/createHashHistory 函数基于 getUrlBasedHistory ， 提供不同的：

- getLocation
- createHref
- validateLocation
- options

属性实现不同的 history 对象

```ts
function getUrlBasedHistory(
  getLocation: (window: Window, globalHistory: Window["history"]) => Location,
  createHref: (window: Window, to: To) => string,
  validateLocation: ((location: Location, to: To) => void) | null,
  options: UrlHistoryOptions = {}
){ /**/ }
```

createMemoryHistory

- 创建 memoery 模式history

```js
export function createMemoryHistory(
  options: MemoryHistoryOptions = {}
): MemoryHistory {
  let { initialEntries = ["/"], initialIndex, v5Compat = false } = options;
  let entries: Location[]; // Declare so we can access from createMemoryLocation
  entries = initialEntries.map((entry, index) =>
    createMemoryLocation(
      entry,
      typeof entry === "string" ? null : entry.state,
      index === 0 ? "default" : undefined
    )
  );
  let index = clampIndex(
    initialIndex == null ? entries.length - 1 : initialIndex
  );
  let action = Action.Pop;
  let listener: Listener | null = null;

  function clampIndex(n: number): number {
    return Math.min(Math.max(n, 0), entries.length - 1);
  }
  function getCurrentLocation(): Location {
    return entries[index];
  }
  function createMemoryLocation(
    to: To,
    state: any = null,
    key?: string
  ): Location {
    let location = createLocation(
      entries ? getCurrentLocation().pathname : "/",
      to,
      state,
```

```
      key
    );
    warning(
      location.pathname.charAt(0) === "/",
      `relative pathnames are not supported in memory history: ${JSON.stringify(
        to
      )}`
    );
    return location;
  }

  let history: MemoryHistory = {
    get index() {
      return index;
    },
    get action() {
      return action;
    },
    get location() {
      return getCurrentLocation();
    },
    createHref(to) {
      return typeof to === "string" ? to : createPath(to);
    },
    encodeLocation(to: To) {
      let path = typeof to === "string" ? parsePath(to) : to;
      return {
        pathname: path.pathname || "",
        search: path.search || "",
        hash: path.hash || "",
      };
    },
    push(to, state) {
      action = Action.Push;
      let nextLocation = createMemoryLocation(to, state);
      index += 1;
      entries.splice(index, entries.length, nextLocation);
      if (v5Compat && listener) {
        listener({ action, location: nextLocation });
      }
    },
    replace(to, state) {
      action = Action.Replace;
      let nextLocation = createMemoryLocation(to, state);
      entries[index] = nextLocation;
      if (v5Compat && listener) {
        listener({ action, location: nextLocation });
      }
    },
```

```
    go(delta) {
      action = Action.Pop;
      index = clampIndex(index + delta);
      if (listener) {
        listener({ action, location: getCurrentLocation() });
      }
    },
    listen(fn: Listener) {
      listener = fn;
      return () => {
        listener = null;
      };
    },
  };

  return history;
}
```

不同的 history 如何选择

- memoryHistory 是内存模式，使用与 react-native 测试当环境
- browserHistory 适用于现代浏览器
- hashHistory 老浏览器支持 hash 模式

history 对象属性

action

- 获取当前 action

```
get action() {
  return action;
}
```

location

- 获取当前 location 对象

```
get location() {
  return getLocation(window, globalHistory);
}
```

listen

- popstate 监听浏览器跳转行为（go/forword/back）

```js
listen(fn: Listener) {
  if (listener) {
    throw new Error("A history only accepts one active listener");
  }
  window.addEventListener(PopStateEventType, handlePop);
  listener = fn;

  return () => {
    window.removeEventListener(PopStateEventType, handlePop);
    listener = null;
  };
},
```

createHref

- 创建地址

```js
createHref(to) {
  return createHref(window, to); // hash 路由与browser路由不一致
},
```

encodeLocation

- 以与 window.history 相同的方式对位置进行编码

```js
encodeLocation(to) {
  // Encode a Location the same way window.location would
  let url = createClientSideURL(
    typeof to === "string" ? to : createPath(to)
  );
  return {
    pathname: url.pathname,
    search: url.search,
    hash: url.hash,
  };
},
```

go

- 指定跳转地址， go 的封装

```js 复制代码
go(n) {
  return globalHistory.go(n); // let globalHistory = window.history;
},
```

push

- 添加地址 pushState 的封住

```js 复制代码
function push(to: To, state?: any) {
    action = Action.Push;
    let location = createLocation(history.location, to, state);
    if (validateLocation) validateLocation(location, to);

    let historyState = getHistoryState(location);
    let url = history.createHref(location);

    // try...catch because iOS limits us to 100 pushState calls :/
    try {
      globalHistory.pushState(historyState, "", url);
    } catch (error) {
      // They are going to lose state here, but there is no real
      // way to warn them about it since the page will refresh...
      window.location.assign(url);
    }

    if (v5Compat && listener) {
      listener({ action, location: history.location });
    }
  }
```

replace

- 替换当前 replaceState 的封装

```js 复制代码
function replace(to: To, state?: any) {
    action = Action.Replace;
    let location = createLocation(history.location, to, state);
    if (validateLocation) validateLocation(location, to);

    let historyState = getHistoryState(location);
```

```
    let url = history.createHref(location);
    globalHistory.replaceState(historyState, "", url);

    if (v5Compat && listener) {
      listener({ action, location: history.location });
    }
}
```

index()

- 当前记忆 history 的索引

```
get index() {
  return index;
}

// 实现
let index = clampIndex(
    initialIndex == null ? entries.length - 1 : initialIndex
)

function clampIndex(n: number): number {
    return Math.min(Math.max(n, 0), entries.length - 1);
}
```

学习方法推荐

- 使用测试工具测试 api, 官方自己的测试用例也写的可以
- 实际在浏览器中进行调试，测试各种 api,巩固学习

小结

- history 在 v5 之前使用单独的包， v6 之后再 router 包中单独实现。
- history 主要由 location api (pathname/serach/hash) + history api(state/key) 构建新 History 对象的创建
- 提供了三种不同的 history 的类型：browserHistory/hashHistory/memoryHistory，以及对应的创建函数。
- createBrowserHistory 与 createHashHistory 区别主要在于: 创建 location 方式和创建 hashHref 的 方式有所不同