ReactRouter

什么是路由

> 路由的概念来源于服务端，在服务端中路由描述的是 URL 与处理函数之间的映射关系。
> 在 Web 前端单页应用 SPA(Single Page Application)中，路由描述的是 URL 与 UI 之间的映射关系，这种映射是单向的，即 URL 变化引起 UI 更新（无需刷新页面）。

路由原理

- 如何改变 URL 却不引起页面刷新?
- 如何检测 URL 变化了?

两种路由

hash 实现

```
1 const HashChangeEventType = 'hashchange';
```

```
window.addEventListener(HashChangeEventType, function() {
  console.log('The hash has changed!')
}, false);
//或者
window.onhashchange =()=>{
 console.log('The hash has changed!')
};
```

history 实现

```
// 监听活动历史记录条目更改
const PopStateEventType = 'popstate';
function handlePop() {
    //监听页面popstate发生改变
}

window.addEventListener(PopStateEventType, handlePop);
```

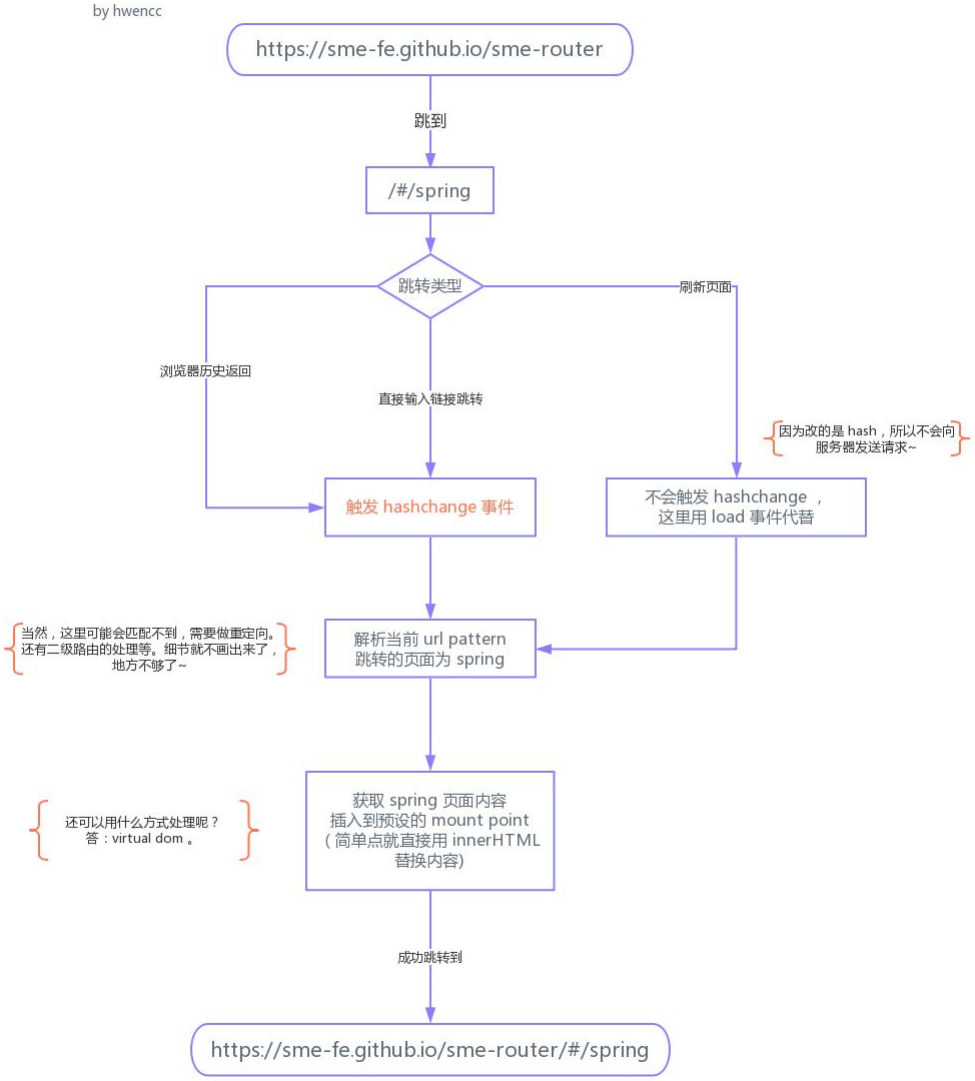
```
history.pushState({page: 2}, "title 2", "?page=2")
history.replaceState({page: 3}, "title 3", "?page=3")
```

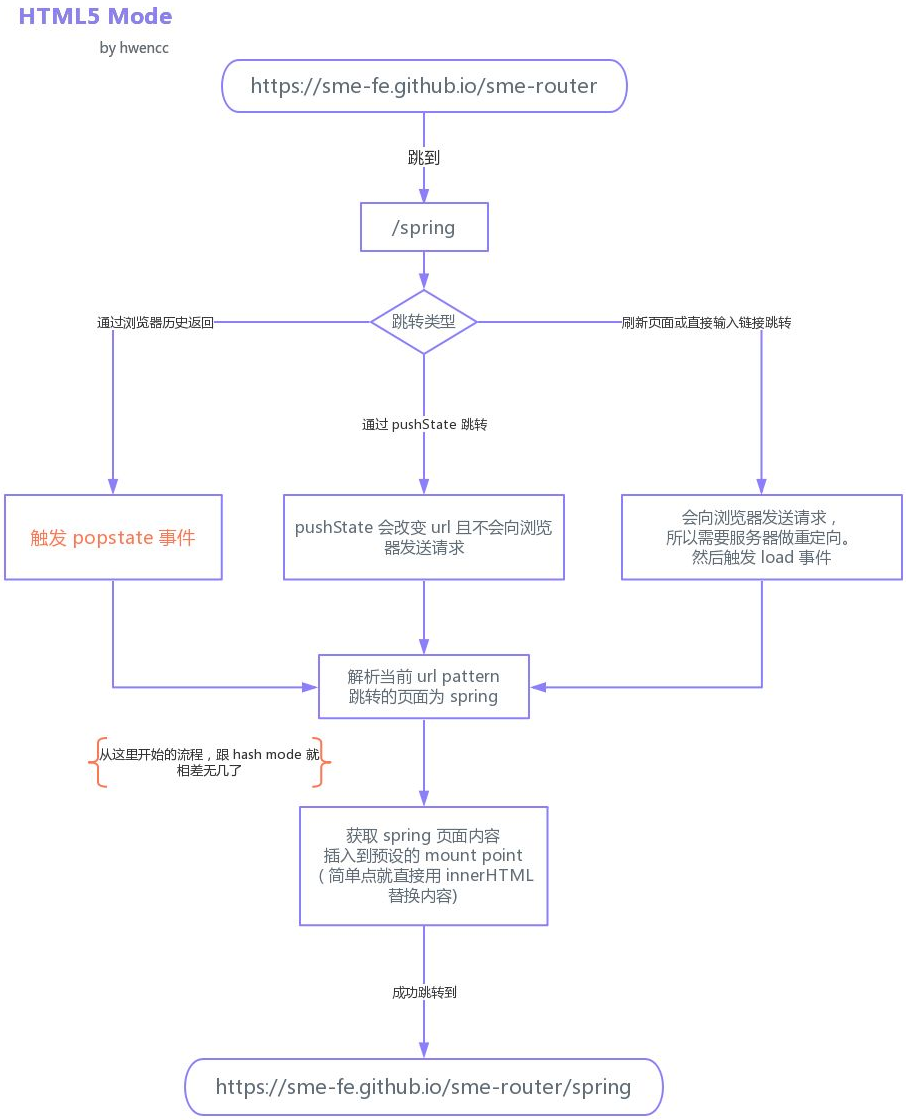
自己去实现一个路由

设计思路

Hash Mode

by hwencc





代码实现

ReactRouter讲解

history // 负责浏览器页面跳转，链接改变

ReactRouter // 负责具体页面组件的渲染

文档

```
1 //import { createBrowserHistory as createHistory } from "histor
  y";
```

```
// BrowserRouter = React.createElement(Router, {history:createHistory()});
// <Router history={createHistory()}></Router>

React.render(
  <BrowserRouter>
      <Layout>
        <Switch>
          <Route exact path="/msg">
            <Message />
          </Route>
          <Route exact path="/index">
            <Index />
          </Route>
          <Route exact path="/about">
            <About />
          </Route>
          <Route path='/test/:id?/:key?' render={(props) => {
            console.log('111router/test', props);
            return props.location.pathname;
          }} />
          <Route path='/param/:a?/:b?' component={(props) => {
            return <Param {...props} />
          }}>
          </Route>
          <Redirect from='/' to='/index' />
        </Switch>
      </Layout>
    </BrowserRouter>
), document.body)
```

Router:

```
// https://github.com/ReactTraining/react-router/blob/master/packages/react-router/modules/Router.js#L58
// 渲染路由页面， 并注入 history 等对象
render() {
    return (
      <RouterContext.Provider
```

```
        value={{
          history: this.props.history,
          location: this.state.location,
          match: Router.computeRootMatch(this.state.location.pathname),
          staticContext: this.props.staticContext
        }}
      >
        <HistoryContext.Provider
          children={this.props.children || null}
          value={this.props.history}
        />
      </RouterContext.Provider>
    );
  }
```

Route

```
// https://github.com/ReactTraining/react-router/blob/master/packages/react-router/modules/Route.js#L30
// 使用 pathToRegexp 去解析 path 生成正则表达式，然后去math 页面 pathname  获取到页面参数
//path:
//exact:
return (
            <RouterContext.Provider value={props}>
              {props.match
                ? children
                  ? typeof children === "function"
                    ? __DEV__
                      ? evalChildrenDev(children, props, this.props.path)
                      : children(props)
                    : children
                  : component
                  ? React.createElement(component, props)
                  : render
                  ? render(props)
                  : null
```

```
                : typeof children === "function"
                ? __DEV__
                  ? evalChildrenDev(children, props, this.props.path)
                  : children(props)
                : null}
            </RouterContext.Provider>
          );
```

withRouter

```
//You can get access to the history object's properties and the closest <Route>'s match via the withRouter higher-order component. withRouter will pass updated match, location, and history props to the wrapped component whenever it render
// 将history match location 等 传给 组件
// https://github.com/ReactTraining/react-router/blob/master/packages/react-router/modules/withRouter.js#L11
function withRouter(Component) {
  const displayName = `withRouter(${Component.displayName || Component.name})`;
  const C = props => {
    const { wrappedComponentRef, ...remainingProps } = props;

    return (
      <RouterContext.Consumer>
        {context => {
          return (
            <Component
              {...remainingProps}
              {...context}
              ref={wrappedComponentRef}
            />
          );
        }}
      </RouterContext.Consumer>
    );
  };
  C.displayName = displayName;
```

```
  C.WrappedComponent = Component;
  return hoistStatics(C, Component);
}

// 问题1： 为什么 match 改变也能够传递给Component 呢?
```

Switch

```
//Renders the first child <Route> or <Redirect> that matches the location.
class Switch extends React.Component {
  render() {
    return (
      <RouterContext.Consumer>
        {context => {
          const location = this.props.location || context.location;

          let element, match;
          React.Children.forEach(this.props.children, child => {
            if (match == null && React.isValidElement(child)) {
              element = child;

              const path = child.props.path || child.props.from;

              match = path
                ? matchPath(location.pathname, { ...child.props, path })
                : context.match;
            }
          });

          return match
            ? React.cloneElement(element, { location, computedMatch: match })
            : null;
        }}
      </RouterContext.Consumer>
    );
```

```
  }
}

// 用处 This is also useful for animated transitions since the matched <Route> is rendered in the same position as the previous one.
```

Redirect

```
// 配合switch
```

history操作引起页面变化

```
//1. Router 监听location 变化事件，并通过更新state 让组件重新渲染
this.unlisten = props.history.listen(location => {
  if (this._isMounted) {
     this.setState({ location });
  } else {
    this._pendingLocation = location;
  }
});

//2. hisory 利用popstate 监听history 的 state变化
window.addEventListener(PopStateEventType, handlePop);
function applyTx(nextAction: Action) {
    action = nextAction;
    [index, location] = getIndexAndLocation();
    listeners.call({ action, location }); // 这里利用事件回调，告诉Router location 变了
}

// 3. Router 监听到 state 变化后，render 重新执行
// context: https://zh-hans.reactjs.org/docs/context.html
// 每当 Provider(提供者) 的 value 属性发生变化时，所有作为 Provider(提供者) 后代的 consumer(使用者) 组件 都将重新渲染
render() {
```

```
    return (
      <RouterContext.Provider
        value={{
          history: this.props.history,
          location: this.state.location,  //location变了
          match: Router.computeRootMatch(this.state.location.path
name),
          staticContext: this.props.staticContext
        }}
      >
        <HistoryContext.Provider
          children={this.props.children || null}
          value={this.props.history}
        />
      </RouterContext.Provider>
    );
  }
```

手写一个ReactRoter路由

> Talk is cheap! show me the code!

Router

> 负责将location 等全局变量传递给子组件， 并且当location 变化时通知组件重新render

Route

> 判断当前页面url 是否与路由path 匹配，如果匹配则渲染当前路由下内容，并且处理好页面的querystring 等传到内容组件中

作业

1. 剩下没实现的API 实现下；
2. 用hash 的方式实现一个路由；

推荐阅读

https://github.com/wayou/wayou.github.io/issues/16