link: null
title: 珠峰架构师成长计划
description: vite.config.ts
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=133 sentences=406, words=3406

1.基础知识

1.1 Reflect

1.1.1 set

- 静态方法 Reflect.set() 工作方式就像在一个对象上设置一个属性

```
Reflect.set(target, propertyKey, value)
```

1.1.2 get

- Reflect.get()方法与从 对象 (target[propertyKey]) 中读取属性类似，但它是通过一个函数执行来操作的。

```
Reflect.get(target, propertyKey)
```

1.2 Proxy

```
let obj ={name:'zhufeng'};
let proxyObj = new Proxy(obj,{
  set(target,key,value){
    console.log(target,key,value);
    return Reflect.set(target,key,value);
  },
  get(target,key){
    console.log(target,key);
   return Reflect.get(target,key);
  }
});
console.log(proxyObj.name);
proxyObj.name = 'jiagou';
```

1.3 decorator

- 修饰器(Decorator)是一个函数，用来修改类的行为

```
function logger(target) {
    console.log(target);
}
@logger
class Person {}
```

2.Mobx

- mobx (https://mobx.js.org/README.html)
- 中文 (https://zh.mobx.js.org/README.html)
- 任何可以从应用状态中派生出来的值都应该被自动派生出来
- MobX 是一个身经百战的库，它通过运用透明的函数式响应编程使状态管理变得简单和可扩展

2.1 安装

```
pnpm create vite
pnpm install @babel/core @babel/plugin-proposal-decorators @babel/plugin-proposal-class-properties
pnpm install mobx mobx-react
```

2.2 vite.config.ts

vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
export default defineConfig({
  plugins: [react({
    babel: {
      plugins: [
        ["@babel/plugin-proposal-decorators", { legacy: true }],
        ["@babel/plugin-proposal-class-properties", { loose: true }],
      ],
    },
  })]
})
```

2.3 jsconfig.json

jsconfig.json

```
{
    "compilerOptions": {
        "experimentalDecorators": true
    }
}
```

2.4 main.tsx

src\main.tsx

```
import {observable} from 'mobx';
console.log(observable);
```

3.observable

3.1 main.jsx

src\main.jsx

```
import {observable} from './mobx';
const proxyObj = observable({name:'1'});
console.log(proxyObj);
```

3.2 mobx\index.jsx

src\mobx\index.jsx

```
export {default as observable} from './observable';
```

3.3 observable.jsx

src\mobx\observable.jsx

```
import {isObject} from './utils';
import {object} from './observableobject';
function createObservable(v) {
    if (isObject(v)) {
        return object(v)
    }
}
export default createObservable;
```

3.4 observableobject.jsx

src\mobx\observableobject.jsx

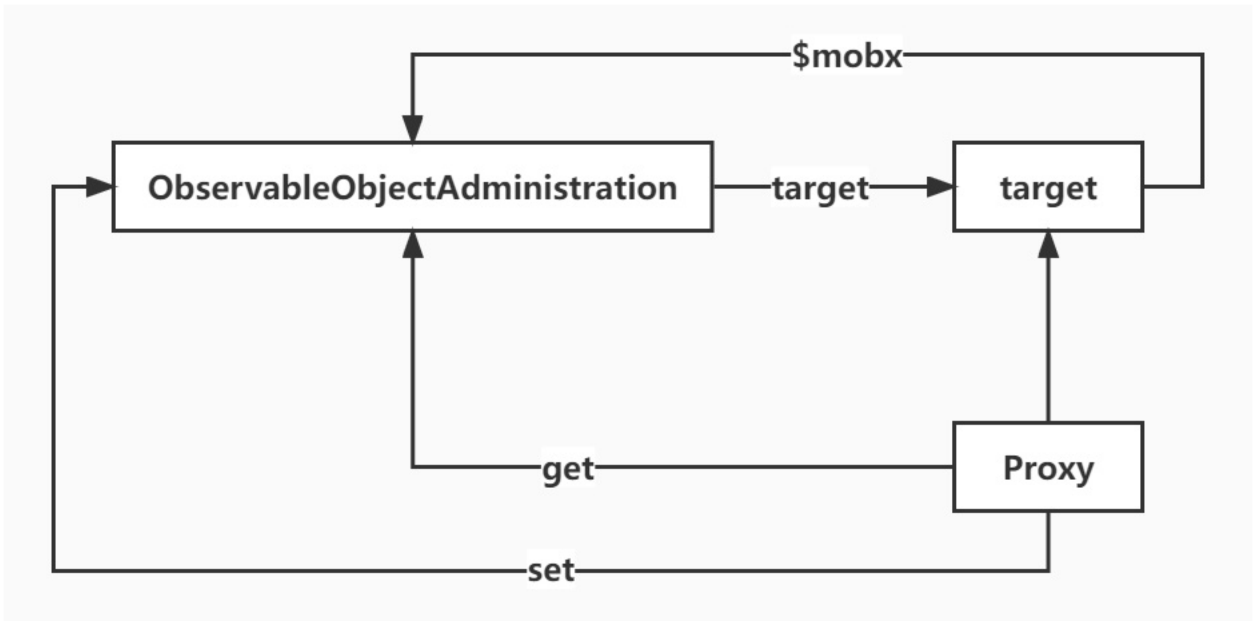
```
export function object(target) {
   return target;
}
```

3.5 utils.jsx

src\mobx\utils.jsx

```
export function isObject(value) {
  return value !== null && typeof value === "object"
}
```

4.asDynamicObservableObject



4.1 src\mobx\utils.jsx

src\mobx\utils.jsx

```
+export const $mobx = Symbol("mobx administration")
+let mobxGuid = 0;
+export function getNextId() {
+    return ++mobxGuid
+}
+
+export function addHiddenProp(object, propName, value) {
+    Object.defineProperty(object, propName, {
+        enumerable: false,
+        writable: true,
+        configurable: true,
+        value
+    })
+}
export function isObject(value){
    return value !== null && typeof value
}
+export function getAdm(target) {
+    return target[$mobx]
+}
```

4.2 observableobject.jsx

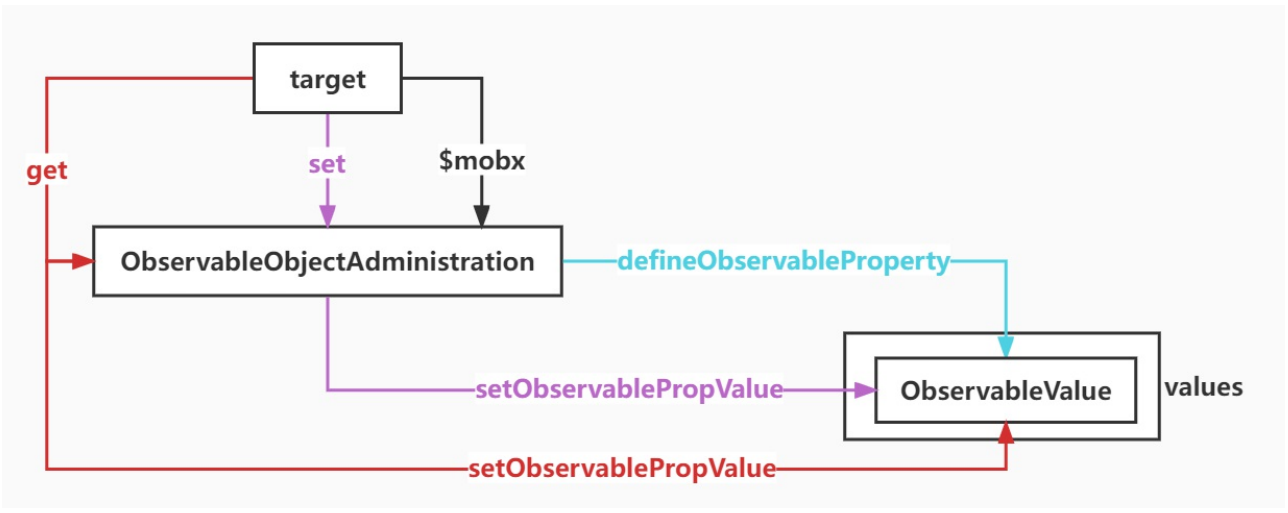
src\mobx\observableobject.jsx

```
+import { getNextId, addHiddenProp,getAdm ,$mobx} from './utils';
+export class ObservableObjectAdministration {
+    constructor(target, values, name) {
+        this.target = target;
+        this.values = values;
+        this.name = name;
+    }
+    get(key) {
+       return this.target[key]
+    }
+    set(key, value) {
+       return this.target[key]=value;
+   }
+}
+export function asObservableObject(target) {
+    const name = `ObservableObject@${getNextId()}`;
+    const adm = new ObservableObjectAdministration(
+       target,
+       new Map(),
+       name
+   )
+   addHiddenProp(target, $mobx, adm)
+   return target;
+}
+const objectProxyTraps = {
+    get(target, name) {
+        return getAdm(target).get(name)
+    },
+    set(target, name, value) {
+        return getAdm(target).set(name, value);
+    }
+}
+export function asDynamicObservableObject(target) {
+    asObservableObject(target);
+    const proxy = new Proxy(target, objectProxyTraps)
+    return proxy;
+}
export function object(target) {
+   const observableObject = asDynamicObservableObject({});
+    console.log(observableObject);
    return target;
}
```

5.extendObservable

5.1 src\mobx\observableobject.jsx

src\mobx\observableobject.jsx



```
import { getNextId, addHiddenProp, getAdm, $mobx } from './utils';
+export class ObservableValue {
+   constructor(value) {
+      this.value = value;
+   }
+   get() {
+      return this.value;
+   }
+   setNewValue(newValue) {
+      this.value = newValue
+   }
+}
export class ObservableObjectAdministration {
   constructor(target, values, name) {
      this.target = target;
      this.values = values;
      this.name = name;
   }
   get(key) {
      return this.target[key];
   }
   set(key, value) {
      return this.target[key] = value;
   }
+  extend(key, descriptor) {
+     this.defineObservableProperty(key, descriptor.value)
+  }
+  getObservablePropValue(key) {
+     return this.values.get(key).get()
+  }
+  setObservablePropValue(key, newValue) {
+     const observable = this.values.get(key)
+     observable.setNewValue(newValue)
+     return true;
+  }
+  defineObservableProperty(key, value) {
+     const descriptor = {
+        configurable: true,
+        enumerable: true,
+        get() {
+           return this[$mobx].getObservablePropValue(key)
+        },
+        set(value) {
+           return this[$mobx].setObservablePropValue(key, value)
+        }
+     }
+     Object.defineProperty(this.target, key, descriptor)
+     const observable = new ObservableValue(value)
+     this.values.set(key, observable)
+  }
}
export function asObservableObject(target) {
   const name = `ObservableObject@${getNextId()}`;
   const adm = new ObservableObjectAdministration(
      target,
      new Map(),
      name
   )
   addHiddenProp(target, $mobx, adm)
   return target;
}
const objectProxyTraps = {
   get(target, name) {
      return getAdm(target).get(name)
   },
   set(target, name, value) {
      return getAdm(target).set(name, value);
   }
}
export function asDynamicObservableObject(target) {
   asObservableObject(target);
   const proxy = new Proxy(target, objectProxyTraps)
   return proxy;
}
+export function extendObservable(proxyObject, properties) {
+   const descriptors = Object.getOwnPropertyDescriptors(properties)
+   const adm = proxyObject[$mobx]
+   Reflect.ownKeys(descriptors).forEach(key => {
+      adm.extend(key, descriptors[key])
+   })
+   return proxyObject;
+}
export function object(target) {
   const observableObject = asDynamicObservableObject({});
+  return extendObservable(observableObject, target);
}
```

6.autorun

6.1 main.jsx

src\main.jsx

```
+import { observable, autorun } from './mobx';
const proxyObj = observable({ name: 1 });
console.log(proxyObj);
+autorun(() => {
+    console.log(proxyObj.name);
+});
+proxyObj.name=2;
```

6.2 mobx\index.jsx

src\mobx\index.jsx

```
export { default as observable } from './observable';
+export { default as autorun } from './autorun';
```

6.3 utils.jsx

src\mobx\utils.jsx

```
export const $mobx = Symbol("mobx administration")
let mobxGuid = 0;
export function getNextId() {
    return ++mobxGuid
}

export function addHiddenProp(object, propName, value) {
    Object.defineProperty(object, propName, {
        enumerable: false,
        writable: true,
        configurable: true,
        value
    })
}
export function isObject(value) {
    return value !== null && typeof value
}
export function getAdm(target) {
    return target[$mobx]
}
+export const globalState = {
+    pendingReactions: []
+}
```

6.4 autorun.jsx

src\mobx\autorun.jsx

```
import { getNextId } from './utils';
import { Reaction } from './reaction';
function autorun(view) {
    const name = "Autorun@" + getNextId();
    const reaction = new Reaction(
        name,
        function () {
            view();
        }
    )
    reaction.schedule()
}
export default autorun;
```

6.5 reaction.jsx

src\mobx\reaction.jsx

```
import { getNextId, globalState } from './utils';
export class Reaction {
    constructor(name = "Reaction@" + getNextId(), onInvalidate) {
        this.name = name;
        this.onInvalidate = onInvalidate;
    }
    schedule() {
        globalState.pendingReactions.push(this)
        runReactions()
    }
    runReaction() {
        this.onInvalidate();
    }
}
export function runReactions() {
    const allReactions = globalState.pendingReactions
    let reaction;
    while (reaction = allReactions.shift()) {
        reaction.runReaction()
    }
}
```

7.observing

7.1 src\mobx\autorun.jsx

src\mobx\autorun.jsx

```
import { getNextId } from './utils';
import { Reaction } from './reaction';
function autorun(view) {
    const name = "Autorun@" + getNextId();
    const reaction = new Reaction(
        name,
        function () {
+           this.track(view)
        }
    )
    reaction.schedule()
}
export default autorun;
```

7.2 src\mobx\utils.jsx

src\mobx\utils.jsx

```
export const $mobx = Symbol("mobx administration")
let mobxGuid = 0;
export function getNextId() {
    return ++mobxGuid
}

export function addHiddenProp(object, propName, value) {
    Object.defineProperty(object, propName, {
        enumerable: false,
        writable: true,
        configurable: true,
        value
    })
}
export function isObject(value) {
    return value !== null && typeof value
}
export function getAdm(target) {
    return target[$mobx]
}
export const globalState = {
    pendingReactions: [],
+   trackingDerivation: null
}
```

7.3 src\mobx\reaction.jsx

src\mobx\reaction.jsx

```
import { getNextId, globalState } from './utils';
export class Reaction {
    constructor(name = "Reaction@" + getNextId(), onInvalidate) {
        this.name = name;
        this.onInvalidate = onInvalidate;
+       this.observing = [];
    }
+   track(fn) {
+       globalState.trackingDerivation = this
+       fn.call();
+       globalState.trackingDerivation = null;
+       bindDependencies(this)
+   }
    schedule() {
        globalState.pendingReactions.push(this)
        runReactions()
    }
    runReaction() {
        this.onInvalidate();
    }
}
+function bindDependencies(derivation) {
+    const { observing } = derivation;
+    observing.forEach(observable => {
+        observable.observers.add(derivation)
+    });
+}
export function runReactions() {
    const allReactions = globalState.pendingReactions
    let reaction;
    while (reaction = allReactions.shift()) {
        reaction.runReaction()
    }
}
```

7.4 observableobject.jsx

src\mobx\observableobject.jsx

```
import { getNextId, addHiddenProp, getAdm, $mobx, globalState } from './utils';
export class ObservableValue {
   constructor(value) {
      this.value = value;
+     this.observers = new Set();
   }
   get() {
+     reportObserved(this)
      return this.value;
   }
   setNewValue(newValue) {
      this.value = newValue;
   }
}
+export function reportObserved(observable) {
+   const derivation = globalState.trackingDerivation
+   if (derivation !== null) {
+      derivation.observing.push(observable);
+   }
+}
export class ObservableObjectAdministration {
   constructor(target, values, name) {
      this.target = target;
      this.values = values;
      this.name = name;
   }
   get(key) {
      return this.target[key];
   }
   set(key, value) {
      return this.target[key] = value;
   }
   extend(key, descriptor) {
      this.defineObservableProperty(key, descriptor.value)
   }
   getObservablePropValue(key) {
      return this.values.get(key).get()
   }
   setObservablePropValue(key, newValue) {
      const observable = this.values.get(key)
      observable.setNewValue(newValue)
      return true;
   }
   defineObservableProperty(key, value) {
      const descriptor = {
         configurable: true,
         enumerable: true,
         get() {
            return this[$mobx].getObservablePropValue(key)
         },
         set(value) {
            return this[$mobx].setObservablePropValue(key, value)
         }
      }
      Object.defineProperty(this.target, key, descriptor)
      const observable = new ObservableValue(value)
      this.values.set(key, observable)
   }
}
export function asObservableObject(target) {
   const name = `ObservableObject@${getNextId()}`;
   const adm = new ObservableObjectAdministration(
      target,
      new Map(),
      name
   )
   addHiddenProp(target, $mobx, adm)
   return target;
}
const objectProxyTraps = {
   get(target, name) {
      return getAdm(target).get(name)
   },
   set(target, name, value) {
      return getAdm(target).set(name, value);
   }
}
export function asDynamicObservableObject(target) {
   asObservableObject(target);
   const proxy = new Proxy(target, objectProxyTraps)
   return proxy;
}
export function extendObservable(proxyObject, properties) {
   const descriptors = Object.getOwnPropertyDescriptors(properties)
   const adm = proxyObject[$mobx]
   Reflect.ownKeys(descriptors).forEach(key => {
      adm.extend(key, descriptors[key])
   })
   return proxyObject;
}
export function object(target) {
   const observableObject = asDynamicObservableObject({});
   return extendObservable(observableObject, target);
}
```

8. propagateChanged

8.1 observableobject.jsx

src\mobx\observableobject.jsx

```
import { getNextId, addHiddenProp, getAdm, $mobx, globalState } from './utils';
export class ObservableValue {
   constructor(value) {
      this.value = value;
      this.observers = new Set();
   }
   get() {
      reportObserved(this)
      return this.value;
   }
   setNewValue(newValue) {
      this.value = newValue;
+     propagateChanged(this)
   }
}
+export function propagateChanged(observable) {
+   const observers = observable.observers;
+   observers.forEach(observer => {
+      observer.onBecomeStale()
+   })
+}
export function reportObserved(observable) {
   const derivation = globalState.trackingDerivation
   if (derivation !== null) {
      derivation.observing.push(observable);
   }
}
export class ObservableObjectAdministration {
   constructor(target, values, name) {
      this.target = target;
      this.values = values;
      this.name = name;
   }
   get(key) {
      return this.target[key];
   }
   set(key, value) {
+     if (this.values.has(key)) {
+        return this.setObservablePropValue(key, value)
+     }
   }
   extend(key, descriptor) {
      this.defineObservableProperty(key, descriptor.value)
   }
   getObservablePropValue(key) {
      return this.values.get(key).get()
   }
   setObservablePropValue(key, newValue) {
      const observable = this.values.get(key)
      observable.setNewValue(newValue)
      return true;
   }
   defineObservableProperty(key, value) {
      const descriptor = {
         configurable: true,
         enumerable: true,
         get() {
            return this[$mobx].getObservablePropValue(key)
         },
         set(value) {
            return this[$mobx].setObservablePropValue(key, value)
         }
      }
      Object.defineProperty(this.target, key, descriptor)
      const observable = new ObservableValue(value)
      this.values.set(key, observable)
   }
}
export function asObservableObject(target) {
   const name = `ObservableObject@${getNextId()}`;
   const adm = new ObservableObjectAdministration(
      target,
      new Map(),
      name
   )
   addHiddenProp(target, $mobx, adm)
   return target;
}
const objectProxyTraps = {
   get(target, name) {
      return getAdm(target).get(name)
   },
   set(target, name, value) {
      return getAdm(target).set(name, value);
   }
}
export function asDynamicObservableObject(target) {
   asObservableObject(target);
   const proxy = new Proxy(target, objectProxyTraps)
   return proxy;
}
export function extendObservable(proxyObject, properties) {
   const descriptors = Object.getOwnPropertyDescriptors(properties)
   const adm = proxyObject[$mobx]
   Reflect.ownKeys(descriptors).forEach(key => {
      adm.extend(key, descriptors[key])
   })
   return proxyObject;
}
export function object(target) {
   const observableObject = asDynamicObservableObject({});
   return extendObservable(observableObject, target);
}
```

8.2 reaction.jsx

src\mobx\reaction.jsx

```
import { getNextId, globalState } from './utils';
export class Reaction {
    constructor(name = "Reaction@" + getNextId(), onInvalidate) {
        this.name = name;
        this.onInvalidate = onInvalidate;
        this.observing = [];
    }
    track(fn) {
        globalState.trackingDerivation = this
        fn.call();
        globalState.trackingDerivation = null;
        bindDependencies(this)
    }
    schedule() {
        globalState.pendingReactions.push(this)
        runReactions()
    }
    runReaction() {
        this.onInvalidate();
    }
+   onBecomeStale() {
+       this.schedule()
+   }
}
function bindDependencies(derivation) {
    const { observing } = derivation;
    observing.forEach(observable => {
        observable.observers.add(derivation)
    });
}
export function runReactions() {
    const allReactions = globalState.pendingReactions
    let reaction;
    while (reaction = allReactions.shift()) {
        reaction.runReaction()
    }
}
```

9. useObserver

9.1 main.jsx

src\main.jsx

```
import { createRoot } from "react-dom/client";
import Counter from "./Counter";
const rootElement = document.getElementById("root");
const root = createRoot(rootElement);
root.render(<Counter />);
```

9.2 Counter.jsx

src\Counter.jsx

```
import React from 'react';
import { makeAutoObservable } from 'mobx';
import { useObserver } from 'mobx-react';
class Store {
    number = 1
    constructor() {
        makeAutoObservable(this, {}, { autoBind: true });
    }
    add() {
        this.number++;
    }
}
let store = new Store();
export default function () {
    return useObserver(() => (
        <div>
            <p>{store.number}p>
            <button onClick={store.add}>+button>
        div>
    ));
};
```

9.3 mobx-react\index.jsx

src\mobx-react\index.jsx

```
import React, { useEffect } from 'react';
import { Reaction } from 'mobx';
export function useObserver(fn) {
    const [, setState] = React.useState();
    const forceUpdate = () => setState({});
    const reactionTrackingRef = React.useRef(null);
    if (!reactionTrackingRef.current) {
        const reaction = new Reaction(`observer`, () => {
            forceUpdate();
        });
        reactionTrackingRef.current = { reaction };
    }
    const { reaction } = reactionTrackingRef.current;
    useEffect(() => {
        return () => {
            reactionTrackingRef.current.reaction.dispose();
            reactionTrackingRef.current = null;
        }
    }, []);
    let rendering;
    reaction.track(() => {
        rendering = fn();
    });
    return rendering;
}
```

9.Observer

9.2 Counter.jsx

src\Counter.jsx

```
import React from 'react';
import { makeAutoObservable } from 'mobx';
+import { useObserver, Observer } from 'mobx-react';
class Store {
    number = 1
    constructor() {
        makeAutoObservable(this, {}, { autoBind: true });
    }
    add() {
        this.number++;
    }
}
let store = new Store();
export default function () {
+   return (
+
+           {
+               () => (
+
+                       {store.number}
+                       +
+
+               )
+           }
+
+   )
};
```

9.3 mobx-react\index.jsx

src\mobx-react\index.jsx

```
import React, { useEffect } from 'react';
import { Reaction } from 'mobx';
export function useObserver(fn) {
    const [, setState] = React.useState();
    const forceUpdate = () => setState({});
    const reactionTrackingRef = React.useRef(null);
    if (!reactionTrackingRef.current) {
        const reaction = new Reaction(`observer`, () => {
            forceUpdate();
        });
        reactionTrackingRef.current = { reaction };
    }
    const { reaction } = reactionTrackingRef.current;
    useEffect(() => {
        return () => {
            reactionTrackingRef.current.reaction.dispose();
            reactionTrackingRef.current = null;
        }
    }, []);
    let rendering;
    reaction.track(() => {
        rendering = fn();
    });
    return rendering;
}
+export function Observer({ children }) {
+    return useObserver(children);
+}
```

10.observer

10.1 Counter.jsx

src\Counter.jsx

```
import React from 'react';
import { makeAutoObservable } from 'mobx';
+import { useObserver, Observer, observer } from 'mobx-react';
class Store {
    number = 1
    constructor() {
        makeAutoObservable(this, {}, { autoBind: true });
    }
    add() {
        this.number++;
    }
}
let store = new Store();
+export default observer(function () {
+    return (
+
+            {store.number}
+            +
+
+    )
+});
```

10.2 mobx-react\index.jsx

src\mobx-react\index.jsx

```
import React, { useEffect } from 'react';
import { Reaction } from 'mobx';
export function useObserver(fn) {
    const [, setState] = React.useState();
    const forceUpdate = () => setState({});
    const reactionTrackingRef = React.useRef(null);
    if (!reactionTrackingRef.current) {
        const reaction = new Reaction(`observer`, () => {
            forceUpdate();
        });
        reactionTrackingRef.current = { reaction };
    }
    const { reaction } = reactionTrackingRef.current;
    useEffect(() => {
        return () => {
            reactionTrackingRef.current.reaction.dispose();
            reactionTrackingRef.current = null;
        }
    }, []);
    let rendering;
    reaction.track(() => {
        rendering = fn();
    });
    return rendering;
}
export function Observer({ children }) {
    return useObserver(children);
}
+export function observer(baseComponent) {
+    let observerComponent = (props, ref) => {
+        return useObserver(() => baseComponent(props, ref));
+    };
+    return observerComponent;
+}
```

11.observer class

11.1 Counter.jsx

src\Counter.jsx

```
import React from 'react';
import { makeAutoObservable } from 'mobx';
import { useObserver, Observer, observer } from 'mobx-react';
class Store {
    number = 1
    constructor() {
        makeAutoObservable(this, {}, { autoBind: true });
    }
    add() {
        this.number++;
    }
}
let store = new Store();
+@observer
+class Counter extends React.Component {
+    render() {
+        return (
+
+                {store.number}
+                +
+
+        )
+    }
+}
+export default Counter;
```

11.2 src\mobx-react\index.jsx

src\mobx-react\index.jsx

```
import React, { useEffect } from 'react';
import { Reaction } from 'mobx';
export function useObserver(fn) {
    const [, setState] = React.useState();
    const forceUpdate = () => setState({});
    const reactionTrackingRef = React.useRef(null);
    if (!reactionTrackingRef.current) {
        const reaction = new Reaction(`observer`, () => {
            forceUpdate();
        });
        reactionTrackingRef.current = { reaction };
    }
    const { reaction } = reactionTrackingRef.current;
    useEffect(() => {
        return () => {
            reactionTrackingRef.current.reaction.dispose();
            reactionTrackingRef.current = null;
        }
    }, []);
    let rendering;
    reaction.track(() => {
        rendering = fn();
    });
    return rendering;
}
export function Observer({ children }) {
    return useObserver(children);
}
export function observer(baseComponent) {
+   if (baseComponent.prototype.isReactComponent) {
+       return makeClassComponentObserver(baseComponent);
+   }
    let observerComponent = (props, ref) => {
        return useObserver(() => baseComponent(props, ref));
    };
    return observerComponent;
}
+export function makeClassComponentObserver(componentClass) {
+    const target = componentClass.prototype
+    const originalRender = target.render
+    target.render = function () {
+        const boundOriginalRender = originalRender.bind(this)
+        const reaction = new Reaction(`render`, () => React.Component.prototype.forceUpdate.call(this))
+        let rendering;
+        reaction.track(() => {
+            rendering = boundOriginalRender();
+        })
+        return rendering
+    }
+    return componentClass
+}
```

12.useLocalObservable [#](#)

12.1 Counter.jsx [#](#)

src\Counter.jsx

```
import React from 'react';
import { useObserver, useLocalObservable } from 'mobx-react';
export default function (props) {
+    const store = useLocalObservable(() => ({
+        number: 1,
+        add() {
+            this.number++;
+        }
+    }));
    return useObserver(() => (

            {store.number}
            +

    ));
};
```

12.2 mobx-react\index.jsx [#](#)

src\mobx-react\index.jsx

```
import React, { useEffect, useState } from 'react';
+import { Reaction, observable } from 'mobx';
export function useObserver(fn) {
    const [, setState] = React.useState();
    const forceUpdate = () => setState({});
    const reactionTrackingRef = React.useRef(null);
    if (!reactionTrackingRef.current) {
        const reaction = new Reaction(`observer`, () => {
            forceUpdate();
        });
        reactionTrackingRef.current = { reaction };
    }
    const { reaction } = reactionTrackingRef.current;
    useEffect(() => {
        return () => {
            reactionTrackingRef.current.reaction.dispose();
            reactionTrackingRef.current = null;
        }
    }, []);
    let rendering;
    reaction.track(() => {
        rendering = fn();
    });
    return rendering;
}
export function Observer({ children }) {
    return useObserver(children);
}
export function observer(baseComponent) {
    if (baseComponent.prototype.isReactComponent) {
        return makeClassComponentObserver(baseComponent);
    }
    let observerComponent = (props, ref) => {
        return useObserver(() => baseComponent(props, ref));
    };
    return observerComponent;
}
export function makeClassComponentObserver(componentClass) {
    const target = componentClass.prototype
    const originalRender = target.render
    target.render = function () {
        const boundOriginalRender = originalRender.bind(this)
        const reaction = new Reaction(`render`, () => Component.prototype.forceUpdate.call + (this))
        let rendering;
        reaction.track(() => {
            rendering = boundOriginalRender();
        })
        return rendering
    }
    return componentClass
}
+export function useLocalObservable(initializer) {
+    return React.useState(() => observable(initializer(), {}, { autoBind: true }))[0];
+}
```