vue3

Vue 3 升级内容

- 全部用 ts 重写（响应式、vdom、模板编译等）
- 性能提升，代码量减少
- 会调整部分 API

proxy实现响应式

Proxy 基本使用

```
const data = {
    name: 'zhangsan',
    age: 20,
}
// const data = ['a', 'b', 'c']
```



```
const proxyData = new Proxy(data, {
    get(target, key, receiver) {
        const result = Reflect.get(target, key, receiver)
        console.log('get', key)
        return result // 返回结果
    },
    set(target, key, val, receiver) {
        const result = Reflect.set(target, key, val, receiver)
        console.log('set', key, val)
        return result // 是否设置成功
    },
    deleteProperty(target, key) {
        const result = Reflect.deleteProperty(target, key)
        console.log('delete property', key)
        return result // 是否删除成功
    }
})
```

Reflect 作用

- 和 Proxy 能力一一对应
- 规范化、标准化、函数式
- 替代掉 Object 上的工具函数

Proxy 实现响应式

- 深度监听，新能更好
- 可监听 新增/删除 属性
- 可监听数组变化

总结

- Proxy 能规避 Object.defineProperty 的问题
- Proxy 无法兼容所有浏览器，无法 polyfill

```
// proxy-observe.js
```

```
// 创建响应式
function reactive(target = {}) {
    if (typeof target !== 'object' || target == null) {
        // 不是对象或数组，则返回
        return target
    }

    // 代理配置
    const proxyConf = {
        get(target, key, receiver) {
            // 只处理本身（非原型的）属性
            const ownKeys = Reflect.ownKeys(target)
            if (ownKeys.includes(key)) {
                console.log('get', key) // 监听
            }

            const result = Reflect.get(target, key, receiver)

            // 深度监听
            // 性能如何提升的？
            return reactive(result)
        },
        set(target, key, val, receiver) {
            // 重复的数据，不处理
            if (val === target[key]) {
                return true
            }

            const ownKeys = Reflect.ownKeys(target)
            if (ownKeys.includes(key)) {
                console.log('已有的 key', key)
            } else {
                console.log('新增的 key', key)
            }

            const result = Reflect.set(target, key, val, receiver)
            console.log('set', key, val)
            // console.log('result', result) // true
            return result // 是否设置成功
        },
        deleteProperty(target, key) {
            const result = Reflect.deleteProperty(target, key)
            console.log('delete property', key)
            // console.log('result', result) // true
            return result // 是否删除成功
        }
    }

    // 生成代理对象
    const observed = new Proxy(target, proxyConf)
    return observed
}

// 测试数据
const data = {
    name: 'zhangsan',
    age: 20,
```

```
    info: {
        city: 'beijing',
        a: {
            b: {
                c: {
                    d: {
                        e: 100
                    }
                }
            }
        }
    }
}

const proxyData = reactive(data)
```