Ref 的概念

proxy 代理的目标必须是非原始值，所以 reactive 不支持原始值类型。所以我们需要将原始值类型进行包装。

```
const flag = ref(false);
effect(() => {
  document.body.innerHTML = flag.value ? 30 : "姜文";
});
setTimeout(() => {
  flag.value = true;
}, 1000);
```

1.Ref & ShallowRef

```
function createRef(rawValue, shallow) {
  return new RefImpl(rawValue, shallow); // 将值进行装包
}
// 将原始类型包装成对象，同时也可以包装对象 进行深层代理
export function ref(value) {
  return createRef(value, false);
}
// 创建浅ref 不会进行深层代理
export function shallowRef(value) {
  return createRef(value, true);
}
```

```
function toReactive(value) { // 将对象转化为响应式的
    return isObject(value) ? reactive(value) : value
}
class RefImpl {
    public _value;
    public dep;
    public __v_isRef = true;
    constructor(public rawValue, public _shallow) {
        this._value = _shallow ? rawValue : toReactive(rawValue); // 浅ref不需要再次代理
    }
    get value(){
        if(activeEffect){
            trackEffects(this.dep || (this.dep = new Set)); // 收集依赖
        }
        return this._value;
```

```
    }
    set value(newVal){
        if(newVal !== this.rawValue){
            this.rawValue = newVal;
            this._value = this._shallow ? newVal : toReactive(newVal);
            triggerEffects(this.dep); // 触发更新
        }
    }
}
```

2.toRef & toRefs

响应式丢失问题

```
const state = reactive({ name: "jw", age: 30 });
let person = { ...state };
effect(() => {
  document.body.innerHTML = person.name + "今年" + person.age + "岁了";
});
setTimeout(() => {
  person.age = 31;
}, 1000);
```

> *如果将响应式对象展开则会丢失响应式的特性*

```
class ObjectRefImpl {
    public __v_isRef = true
    constructor(public _object, public _key) { }
    get value() {
        return this._object[this._key];
    }
    set value(newVal) {
        this._object[this._key] = newVal;
    }
}
export function toRef(object, key) { // 将响应式对象中的某个属性转化成ref
    return new ObjectRefImpl(object, key);
}
export function toRefs(object) { // 将所有的属性转换成ref
    const ret = Array.isArray(object) ? new Array(object.length) : {};
    for (const key in object) {
        ret[key] = toRef(object, key);
    }
    return ret;
}
```

```
let person = { ...toRefs(state) }; // 解构的时候将所有的属性都转换成ref即可
effect(() => {
  document.body.innerHTML =
    person.name.value + "今年" + person.age.value + "岁了";
});
setTimeout(() => {
  person.age.value = 31;
}, 1000);
```

3.自动脱 ref

```
let person = proxyRefs({ ...toRefs(state) });
effect(() => {
  document.body.innerHTML = person.name + "今年" + person.age + "岁了";
});
setTimeout(() => {
  person.age = 31;
}, 1000);
```

```
export function proxyRefs(objectWithRefs) {
  // 代理的思想，如果是ref 则取ref.value
  return new Proxy(objectWithRefs, {
    get(target, key, receiver) {
      let v = Reflect.get(target, key, receiver);
      return v.__v_isRef ? v.value : v;
    },
    set(target, key, value, receiver) {
      // 设置的时候如果是ref,则给ref.value赋值
      const oldValue = target[key];
      if (oldValue.__v_isRef) {
        oldValue.value = value;
        return true;
      } else {
        return Reflect.set(target, key, value, receiver);
      }
    },
  });
}
```