Computed 实现原理

接受一个 getter 函数，并根据 getter 的返回值返回一个不可变的响应式 ref 对象。

- 计算属性的getter只有当取值时才会执行。
- 计算属性是具备缓存的，如果依赖的值不发生变化，不会重新执行getter。
- 计算属性也是一个effect，内部也具备依赖收集的功能。

```
import { reactive, effect, computed } from './reactivity.esm.js'
const state = reactive({ flag: true, name: 'jw', age: 30 });
const aliasName = computed(() => {
  console.log('computed-run')
  return '**' + state.name + '**'
})
const runner = effect(() => {
  console.log('effect-run')
  console.log(aliasName.value)
  console.log(aliasName.value)
  console.log(aliasName.value)
});
setTimeout(() => {
  state.name = 'Mr Jiang'
}, 1000)
```

1.计算属性实现

```
import { isFunction } from "@vue/shared";
import { activeEffect, ReactiveEffect, trackEffects, triggerEffects }
from "./effect";

class ComputedRefImpl {
    public effect; // 计算属性是基于effect实现的
    public _value; // 缓存计算属性的执行结果
    public dep;    // 记录依赖的effect
    public __v_isRef = true; // 计算属性是一个ref
    public _dirty = true; // 标识此computed依赖的值是否发生变化
    constructor(getter,public setter) {
        this.effect = new ReactiveEffect(getter,()=>{
            if(!this._dirty){
                // 依赖的值变化更新dirty并触发更新
                this._dirty = true;
                triggerEffects(this.dep);
            }
```

```
        });
    }
    get value(){ // 取值的时候进行依赖收集
        if(activeEffect){
            trackEffects(this.dep || (this.dep = new Set));
        }
        if(this._dirty){ // 如果是脏值，执行函数
            this._dirty = false;
            this._value = this.effect.run();
        }
        return this._value;
    }
    set value(newValue){
        this.setter(newValue)
    }
}
export function computed(getterOrOptions) {
    const onlyGetter = isFunction(getterOrOptions); // 传入的是函数就是
getter
    let getter;
    let setter;
    if (onlyGetter) {
        getter = getterOrOptions;
        setter = () => { }
    } else {
        getter = getterOrOptions.get;
        setter = getterOrOptions.set;
    }
    // 创建计算属性
    return new ComputedRefImpl(getter, setter)
}
```

> *创建 ReactiveEffect 时，传入scheduler函数，稍后依赖的属性变化时调用此方法！*

2.收集依赖的effect

```
export function trackEffects(dep) {
  // 收集dep 对应的effect
  let shouldTrack = !dep.has(activeEffect);
  if (shouldTrack) {
    dep.add(activeEffect);
    activeEffect.deps.push(dep);
  }
}
```

3.触发依赖的effect更新

```
export function triggerEffects(dep) {
  const effects = [...dep];
  effects.forEach((effect) => {
    if (effect !== activeEffect) {
      if (effect.scheduler) {
        // 如果有调度函数则执行调度函数
        effect.scheduler();
      } else {
        effect.run();
      }
    }
  });
}
```