测试代码：chapter02

第一步：数据变化监听

变化监听目前使用的是Object.defineProperty.。Vue3.0会使用ES6的proxy来实现。

demo01:用Object.defineProperty实现一个数据监听的方法。

存在的问题：

1.Object的属性增加或者删除的时候，都不能监听到变化。用vm.$set()，

第二步：收集依赖

1.在get里面收集依赖， 在set里面触发依赖。看demo02.js

每个key对应一个数组，用于存储这个key的依赖，每个依赖假定是一个函数。当数据变化的时候，执行这些函数，实现数据变化，通知依赖数据变更的目的。

暂定依赖函数放在window.target里面。

2.直接把依赖写在数据监听函数里面，不够解耦，需要抽象出一个专门处理依赖的对象Dep.js

接下来在main.js里面运行一下defineReactiveData01.js

3.存在的问题：实际依赖是不可能是target这么简单,并且依赖的来源可以是各种情况，比如表达式，比如写一个watch:

vm.watch('a.b.c',function(newVal,oldVal){

console.log(newVal,oldVal)

})，

还比如是computed的。

那么如何在上面的基础上，抽象一个可以处理各种类型的依赖的类？---watcher

看Watcher.js

简单来说，就是当遇到一个watch或者compute的时候，就新建一个watcher对象，并执行watcher对象的get方法,而因为一旦执行了watcher的get方法，就把整个watcher对象添加到对应的key的依赖中了，就可以实现监听数据变化。而当数据变化的时候，就执行watcher对象的update方法，这样就实现数据响应了。

在main.js里面运行一下demo03.js

第三步：递归监听data所有的key.

到目前为止，我们知道了数据监听的原理，也抽象出了Dep类来专门处理依赖收集，同时也抽象了一个依赖类:watcher。那接下来就是简单的，递归所有的data的key设置数据监听---Observer类。Observer.js

重新写一个干净的defineReactiveData.js。然后在main.js里面运行一下demo04.js

问题：你可以说说vm.$set的实现原理吗？

简化版的$set.js

在main.js里面运行一下demo05.js