Vue数据响应式原理

测试代码

核心原理

利用`Object.defineProperty()`劫持数据，从而将数值变化转换为dom更新

整体流程

1. 数据拦截处理
2. 依赖收集操作
3. 通知更新

数据拦截处理

递归data中的所有属性并执行数据劫持

initData src\core\instance\state.js

执行observe，执行响应式处理

observe(data, true /* asRootData */)

获取Observer实例

Observer src\core\observer\index.js

辨别传递进来的value类型，如果是对象使用walk遍历，如果是数组使用observeArray遍历

defineReactive(obj, keys[i]) src\core\observer\index.js

> 总结：定义data中所有key，做数据拦截
>
> 将来如果数据变化，则通知更新

依赖收集

目的是要知道data变化后通知谁去更新。

关键角色：Dep, Watcher

data key <=> Dep 1:1

Component <=> Watcher 1:1

依赖收集过程是建立Dep和Watcher之间的对应关系

mountComponent src\core\instance\lifecycle.js

定义更新函数，创建watcher实例

Watcher src\core\observer\watcher.js

执行更新函数的

> 两种创建watcher的情况：
>
> 1. 创建组件时
> 2. 用户编写watch配置项
> 3. 用户调用$watch api

_render src\core\instance\render.js

触发getter，执行依赖收集

depend () src\core\observer\dep.js

addDep (*dep*: Dep) src\core\observer\watcher.js

建立Dep和Watcher之间的关系

通知更新

更新过程通常会造成watcher实例进入异步更新队列，每次更新周期批量执行更新任务。

notify() src\core\observer\dep.js

通知相关的所有watcher执行更新操作

update () src\core\observer\watcher.js

将watcher入队

queueWatcher(watcher) src\core\observer\scheduler.js

将watcher加入到异步更新队列中

nextTick(flushSchedulerQueue)

异步执行队列刷新

timerFunc src\core\util\next-tick.js

根据执行环境定义异步策略

flushSchedulerQueue

刷新队列

watcher.run()

执行组件更新逻辑

数组响应化

拦截数组的变更方法，额外发送更新通知。

Observer src\core\observer\index.js

判断value是否是数组，如果是数组

src\core\observer\array.js

拦截7个数组变更方法，使之可以发送变更通知

> 总结：数组响应化建立在对7个变更方法拦截之上
>
> - 新增、删除、替换元素时只能使用这7个方法去做，不能使用索引

> - 数组里面项初始化做过响应化处理，但要新增和删除key时候需要使用Vue.set/this.$set/Vue.delete/this.$delete