Vue双向数据绑定原理：

利用Object.defineProperty()方法，在vue初始化实例时对属性执行getter/setter转换，每个组件实例都对应一个watcher实例，他会在组件使用render函数渲染的过程中把接触过的数据属性(就是data对象里的数据属性)记录为依赖，之后当依赖项的getter或者setter触发时会同时watcher，watcher就会促使关联的组件重新渲染

对于对象：Vue 无法检测 property 的添加或移除。

对于数组：Vue 不能检测以下数组的变动：（1）当你利用索引直接设置一个数组项时（2）当你修改数组的长度时（可用splice解决）

事件循环机制：

为什么会有事件循环机制：JavaScript是单线程的，就是一个任务执行万才能执行后一个任务，如果某个执行事件太长，就容易造成阻塞，为了解决这个问题，JavaScript引入了事件循环机制

事件循环是什么：JavaScript单线程任务被分为同步任务和异步任务，在主线程中执行完同步任务后，主线程不断的从任务队列中读取事件，这个过程是循环不断的，这种运行机制就是事件循环

同步任务：立即执行的任务，在主线程上排队执行，前一个任务执行完毕，才能执行后一个任务

异步任务：异步执行的任务，不进入主线程，而是在异步任务有了结果后，将注册的回调函数放入任务队列中，等待主线程空闲的时候读取执行

异步函数在相应辅助线程中处理完成后，即异步函数达到触发条件了，就把回调函数推入任务队列中，而不是说注册一个异步任务就会被放在这个任务队列中

异步任务又分为宏任务和微任务

常见宏任务：1.定时器（setTimeout,setInterval,setImmediate）2.srcipt(可以理解未外层同步代码) 3.事件绑定，4ajax ,5 I/O(node.js的) 6.postMessage, MessageChannel5

常见微任务：1.promise(promise.then promise.catch promise.finally) 2.process.nextTick(node.js里的)都是微任务 3.Object.observe(已废弃；Proxy对象替代)4.MutationObserver

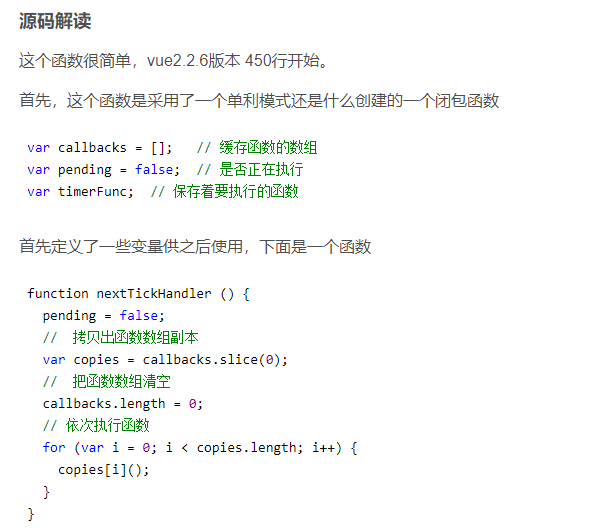
**Promise的then等方法是微任务，而Promise中的代码是同步任务，并且，process的nextTick的执行顺序优先于Promise的then等方法，因为process.nextTick是直接告诉浏览器说要尽快执行，而不是放入队列**

微任务总是先于宏任务执行，也就是说，这三种任务的执行顺序是：同步任务>微任务>宏任务

this.$nextTick()作用：在下次 DOM 更新循环结束之后执行延迟回调

[vue之nextTick全面解析\_aikouhui4965的博客-CSDN博客](https://blog.csdn.net/aikouhui4965/article/details/102124372?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-2-102124372-blog-112428797.pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-2-102124372-blog-112428797.pc_relevant_3mothn_strategy_and_data_recovery)

Vue在更新DOM时时异步执行的，只要侦听到数据变化，Vue将开启一个队列，并缓冲在同一事件循环中发生的所有数据变更。如果同一个watcher被多次出发，置灰被推入到队列中一次



$nextTick的目的就是把传进来的函数延迟到dom更新后再使用，所以这里使用js的方法来做到这一点

$nextTick的工作原理主要是使用js事件循环中的Promise.then…MutationObser 和setTimeout