nextTick

Vue 在更新 DOM 时是异步执行的。 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` 被多次触发，只会被推入到队列中一次。**Vue 在内部对异步队列尝试使用原生的 Promise.then、MutationObserver 和 setImmediate，如果执行环境不支持，则会采用 setTimeout(fn, 0) 代替。**

为了在数据变化之后等待 Vue 完成更新 DOM，可以在数据变化之后立即使用 Vue.nextTick(callback)。 这样回调函数将在 DOM 更新完成后被调用。例如：

```
<template>
  <div>
    <ul ref="list">
      <li v-for="item of list" :key="item.id">
        <span>name: {{ item.name }}</span>
      </li>
    </ul>
    <button @click="onAddItem">添加两项</button>
    <div ref="message" class="message">{{ message }}</div>
    <button @click="onChangeMessage">改变message</button>
  </div>
</template>

<script>
export default {
  name: 'NextTick',
  data() {
    return {
      list: [
        {id: '1', name: 'zhangsan'},
        {id: '2', name: 'lisi'},
      ],
      message: "before message"
    }
  },
  methods: {
    async onAddItem() {
      this.list.push(`${new Date().getTime()}`);
      this.list.push(`${new Date().getTime()}`);
      console.log("before-nextTick", this.$refs.list.childNodes.length); // 获取
ul 的子节点的数量
      await this.$nextTick();
      console.log("after-nextTick", this.$refs.list.childNodes.length);
    },
    async onChangeMessage() {
      this.message = 'updated';
      console.log("before-nextTick, message has updated",
this.$refs.message.textContent === 'updated'); // false
      await this.$nextTick();
      console.log("after-nextTick, message has updated",
```

```
  this.$refs.message.textContent === 'updated'); // true
      }
    }
  }
  </script>
```

第一次点击“添加两项”，输出 `before-nextTick, 2` 和 `before-nextTick, 4`; 第二次点击“添加两项”，输出 `before-nextTick, 4` 和 `before-nextTick, 6`; 第三次点击“添加两项”，输出 `before-nextTick, 6` 和 `before-nextTick, 8`;

由输出结果可知：

- **更新 data 数据后，没有立即更新 DOM；**
- DOM 的更新是异步执行的，等待多个数据修改完成后，一次更新 DOM；
- 在数据变化之后立即，使用 `Vue.nextTick(callback)` 可以确保 Vue 完成了更新 DOM；
- `Vue.nextTick(callback)` 返回的是一个 `Promise` 对象。