第 3 章 计算属性和数据监听

在 Vue 组件的模板中，可以通过插值表达式来输出模型变量的运算结果，例如以下代码中的{{a*b+b*c+a*c}}是插值表达式：

```
<p>a={{a}},b={{b}},c={{c}}</p>
<p>插值表达式的取值：{{a*b+b*c+a*c}}  </p>
```

如果模板中嵌入了大量包含复杂运算的插值表达式，会影响模板的可读性，而且插值表达式不能包含流程控制逻辑，比如不能包含 if-else 条件判断或者 for 循环等。

为了弥补插值表达式的不足，Vue 提供了以下三种替代方案：

（1）用 Vue 的计算属性来替代插值表达式。

（2）用 Vue 的 watch 选项来替代插值表达式。

（3）用方法调用来替代插值表达式。

本章会介绍以上三种方式的具体用法，并且比较它们的优缺点。

3.1 计算属性

Vue 组件的 computed 选项用来定义计算属性，例程 3-1 的 calculate.html 演示了 computed 选项的基本用法。

例程 3-1 calculate.html

```
<div id="app">
  <p>a={{a}},b={{b}},c={{c}}</p>
  <p>插值表达式的取值：{{a*b+b*c+a*c}}  </p>
  <p>计算属性的取值：{{ result }}  </p>
</div>

<script>
  const vm=Vue.createApp({
    data(){
      return { a:10,b:20,c:30 }
    },

    computed:{
      //定义 result 计算属性
      result(){
        console.log('get result property')
        return  this.a*this.b+this.b*this.c+this.a*this.c
      }
    }
```

```
    }).mount('#app')
  </script>
```

在 calculate.html 中，{{ a*b+b*c+a*c }}和{{result}}能输出同样的结果。显然，{{result}}使得模板更加简洁。result 就是根组件的计算属性，它的定义方式如下：

```
computed:{
  result(){    //result 计算属性的 get 函数
    console.log('get result property')
    return  this.a*this.b+this.b*this.c+this.a*this.c
  }
}
```

Vue 框架会通过调用 result()函数来计算 result 的取值。这个 result()函数相当于 result 计算属性的 get 函数。

通过浏览器访问 calculate.html，会得到如图 3-1 所示的网页。{{ a*b+b*c+a*c }}和{{result}}的取值都是 1100。

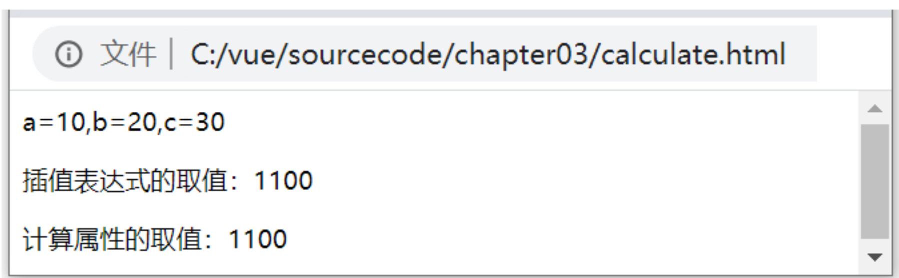

图 3-1 calculate.html 的网页

3.1.1 读写计算属性

在本章 3.1 节开头的 calculate.html 中，result()函数用来读取 result 计算属性的值，相当于 result 计算属性的 get 函数。在例程 3-2 的 fullname.html 中，为 fullName 计算属性同时提供了 get 和 set 函数。

例程 3-2 fullname.html

```
<div id="app">
  <p>First name: <input type="text" v-model="firstName"></p>
  <p>Last name: <input type="text" v-model="lastName"></p>
  <p>Full  name: <input type="text" v-model="fullName"></p>
  <p>{{ fullName }}</p>
</div>

<script>
  const vm = Vue.createApp({
    data() {
      return {
        firstName: 'Tom',
        lastName: "Smith"
```

```
          }
        },

        computed: {
          fullName: {
            get () {  //get 函数
              console.log('call get')
              return this.firstName + ' ' + this.lastName
            },
            set (newValue) {   //set 函数
              console.log('call set')
              console.log('原先的 fullName:'+this.fullName)
              var names = newValue.split(' ')
              this.firstName = names[0]
              this.lastName = names[names.length - 1]
            }
          }
        }
      }).mount('#app')
    </script>
```

通过浏览器访问 fullname.html，会得到如图 3-2 所示的网页。

图 3-2 fullname.html 的网页

为了跟踪 fullname 计算属性的 get 和 set 函数的调用时机，在这两个函数中都会向控制台输出一些日志。

如图 3-3 所示，如果修改网页上 firstName 或 lastName 输入框的值，会看到 get 函数被调用。如果修改 fullName 输入框的值，会看到 set 函数以及 get 函数先后被调用。

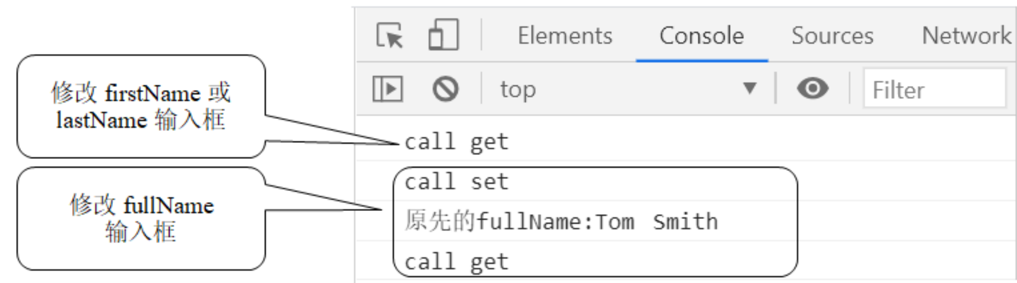

图 3-3 fullName 计算属性的 get 和 set 函数的调用时机

firstName 变量、lastName 变量以及 fullName 计算属性之间存在依赖关系。从图 3-3 可以看出，fullName 计算属性的 get 函数和 set 函数的作用如下：

（1）当 firstName 和 lastName 变量被更新，get 函数会更新 fullName 计算属性。

（2）当 fullName 计算属性被更新，set 函数会更新 firstName 和 lastName 变量。

在 fullName 计算属性的 set 函数中，newValue 参数表示更新后的 fullName 计算属性的值，this.fullName 是更新前的值：

```
set (newValue) {   //set 函数
  console.log('call set')
  console.log('原先的 fullName:'+this.fullName)
  var names = newValue.split(' ')
  this.firstName = names[0]
  this.lastName = names[names.length - 1]
}
```

3.1.2 比较计算属性和方法

Vue 组件的计算属性和方法都可以进行逻辑复杂的运算。在例程 3-3 的 compare.html 中，result1 计算属性以及 getResult1()方法都能得到同样的运算结果，并且它们都依赖变量 a；result2 计算属性以及 getResult2()方法都能得到同样的运算结果，并且它们都依赖变量 b。

例程 3-3 compare.html

```
<div id="app">
  <p>a={{a}},b={{b}}</p>
  <p>a: <input type="text" v-model="a"></p>
  <p>计算属性的取值: {{ result1 }}, {{ result2 }} </p>
  <p>调用方法的结果: {{ getResult1() }}, {{getResult2() }} </p>
</div>

<script>
  const vm=Vue.createApp({
    data(){
      return { a:100,b:20 }
    },
```

```
      computed:{
        result1(){   //get 函数
          console.log('call result1()')
          return  Math.sqrt(this.a)  //计算变量 a 的平方根
        },
        result2(){   //get 函数
          console.log('call result2()')
          return  Math.pow(this.b,3)  //计算变量 b 的三次方
        }
      },

      methods:{
        getResult1(){
          console.log('call getResult1()')
          return  Math.sqrt(this.a)  //计算变量 a 的平方根
        },
        getResult2(){
          console.log('call getResult2()')
          return  Math.pow(this.b,3)  //计算变量 b 的三次方
        }
      }
    }).mount('#app')
  </script>
```

通过浏览器访问 compare.html，会得到如图 3-4 所示的网页。

图 3-4　compare.html 的网页

为了跟踪 result1 和 result2 计算属性的 get 函数，以及 getResult1()和 getResult2()方法的调用时机，在这些函数和方法中都会向控制台输出一些日志。

在 compare.html 网页的输入框中修改变量 a 的值，如图 3-5 所示，从控制台输出的日志可以看出，Vue 框架会调用 result1 计算属性的 get 函数，并且调用 getResult1()和 getResult2()方法。

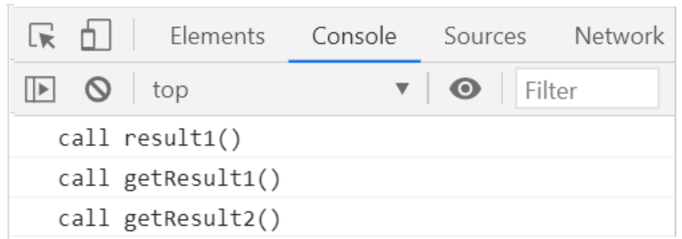

图 3-5 更新变量 a 后的输出日志

Vue 框架会把计算属性存放在专门的缓存中。由于计算属性 result2 不依赖变量 a，因此当变量 a 被更新时，Vue 框架无需调用 result2 计算属性的 get 函数，而是直接从缓存中读取 result2 的值，用来渲染 DOM 中的{{result2}}插值表达式。

由此可见，Vue 框架对计算属性的 get 函数的调用做了性能优化。只有当计算属性所依赖的变量被更新，Vue 框架才会调用它的 get 函数。在本范例中，变量 a 被更新，Vue 框架只需调用 resutl1 计算属性的 get 函数，重新运算 resutl1 计算属性的值。

而对于 getResult1()和 getResult2()方法，当变量 a 被更新时，Vue 框架无法知道上一次这两个方法的取值，因此在渲染 DOM 时，会分别调用这两个方法，重新计算{{getResult1()}}和{{getResult2()}}的取值。

3.1.3 用计算属性过滤数组

在遍历数组时，如果希望只输出数组中符合特定条件的元素，那么可以用计算属性来过滤数组。在例程 3-4 的 array.html 中，passStudents 和 unPassStudents 是两个计算属性，它们对 students 数组变量进行了过滤。passStudents 计算属性表示所有成绩及格的学生，unPassStudents 计算属性表示所有成绩不及格的学生。

例程 3-4 array.html

```
<div id="app">
  <h1>及格同学</h1>
  <ul>
    <li v-for="student in passStudents">
      {{student.name}}:{{student.score}}
    </li>
  </ul>
  <h1>不及格同学</h1>
  <ul>
    <li v-for="student in unPassStudents">
      {{student.name}}:{{student.score}}
    </li>
  </ul>
  </ul>
```

```
</div>

<script>
  const vm= Vue.createApp({
    data(){
      return {
        students: [
          {name: '张飞', score:56},
          {name: '周瑜', score:68},
          {name: '李进', score:92},
          {name: '鲁智深', score:75},
          {name: '程咬金', score:48},
          {name: '诸葛亮', score:99},
        ]
      }
    },

    computed: {
      passStudents() {  //get 函数
        return this.students.filter(student=>student.score >= 60)
      },
      unPassStudents(){  //get 函数
        return this.students.filter(student=> student.score <60)
      }
    }
  }).mount('#app')
</script>
```

通过浏览器访问 array.html，会得到如图 3-6 所示的网页。

图 3-6 array.html 的网页

3.1.4 计算属性实用范例：实现购物车

对于购物网站应用，需要在前端管理购物车。购物车中包含了所选购的商品的名字、数量、单价和小计（数量*单价）的信息，还包含了所有选购商品的总金额的信息。用户可以

修改选购商品的数量，还可以删除选购的商品。

例程 3-5 的 shoppingcart.html 实现了购物车。cartItems 数组变量表示用户选购的所有商品条目。在模板中通过 v-for 指令来遍历 cartItems 数组变量：

```
<tr v-for="(item,index) in cartItems">……</tr>
```

cartItems 数组中的每个商品条目表示一个商品的具体购买信息，在购物车网页上会显示商品的具体购买信息：

（1）序号：{{index+1}}

（2）名称：{{item.name}}

（3）价格：{{ currency(item.price,2) }}

（4）数量：{{item.count}}

（5）小计：{{ currency(item.count*item.price,2) }}

以上 currency()方法用于对金额数字进行格式化，保留两位小数，并且会在数字开头加上货币符号“￥”。

在遍历了 cartItems 数组中的所有商品后，还会在网页上显示所有选购商品的总金额：

```
总金额：{{ currency( total,2 ) }}
```

以上 total 是计算属性，它的定义如下：

```
computed:{
  total(){  //计算属性 total 表示总金额
    var sum=0;
    for(let i=0;i<this.cartItems.length;i++){
      sum+=parseFloat(this.cartItems[i].price)
                *parseFloat(this.cartItems[i].count)
    }
    return sum
  }
}
```

以上 total()函数遍历 cartItems 数组变量，计算所有选购商品的总金额。由于 total()函数依赖 cartItems 数组变量，因此当用户在网页上修改了 cartItems 数组变量中商品的购买数量，或者删除了某个商品时，Vue 框架就会调用 total()函数，从而同步更新 total 计算属性的值。

例程 3-5 shoppingcart.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>购物车</title>
    <script src="vue.js"></script>
  </head>
```

```
<body>
  <div id="cart">
    <table border="1" width="600" style="margin: 0 auto;">
      <tr>
        <th>序号</th>
        <th>名称</th>
        <th>价格</th>
        <th>数量</th>
        <th>小计</th>
        <th>操作</th>
      </tr>
      <tr v-for="(item,index) in cartItems">
        <td>{{index+1}}</td>
        <td>{{item.name}}</td>
        <td>{{ currency(item.price,2) }}</td>
        <td>
          <!-- 递增数量的按钮 -->
          <button v-on:click="item.count<=0?0:item.count-=1">
            -
          </button>

          <!-- 设置数量的输入框 -->
          <input type="text" v-model.number="item.count"
             @keyup="item.count=item.count<=0?0:item.count"/>

          <!-- 递减数量的按钮 -->
          <button v-on:click="item.count+=1">+</button>
        </td>
        <td>{{ currency(item.count*item.price,2) }}</td>
        <td>
          <button @click="remove(index)">移除</button>
        </td>
      </tr>
      <tr>
        <td colspan="6" align="right">
           总金额: {{ currency( total,2 ) }}
        </td>
      </tr>
    </table>
  </div>

  <script>
    const vm= Vue.createApp({
      data() {
        return {
          cartItems:[
            {id:10001,name:'足球',price:105.5,count:10},
            {id:10002,name:'跳绳',price:8.8,count:2},
            {id:10003,name:'呼啦圈',price:21.6,count:5},
          ]
        }
      },
```

```
        computed:{
          total(){  //total 计算属性表示总金额
            var sum=0;
            for(let i=0;i<this.cartItems.length;i++){
              sum+=parseFloat(this.cartItems[i].price)
                         *parseFloat(this.cartItems[i].count)
            }
            return sum
          }
        },

        methods:{
           remove(index){  //删除购物车的一个商品
             if(confirm('你确定要删除吗?')){
               this.cartItems.splice(index,1)
             }
           },

          //对金额进行格式化
          //参数 v 表示需要格式化的金额
          //参数 n 表示需要保留的小数位数
          currency(v,n){
            if(!v){  //如果 v 为空，就退出
              return ""
            }
            //增加货币符号"￥"，并且保留 n 位小数，默认保留 2 位小数
           return "￥"+v.toFixed(n||2)
          }
        }
      }).mount('#cart')

    </script>
  </body>
</html>
```

通过浏览器访问 shoppingcart.html，会得到如图 3-7 所示的购物车网页。

ⓘ 文件 | C:/vue/sourcecode/chapter03/shoppingcart.html

序号	名称	价格	数量	小计	操作
1	足球	￥105.50	- 10 +	￥1055.00	移除
2	跳绳	￥8.80	- 2 +	￥17.60	移除
3	呼啦圈	￥21.60	- 5 +	￥108.00	移除
				总金额：￥1180.60	

图 3-7 shoppingcart.html 的网页

在 shoppingcart.html 的网页上增加或删除某个商品的数量，总金额以及小计会同步更新。如果移除某个商品，总金额也会同步更新。

shoppingcart.html 同时运用了插值表达式、方法和计算属性。从这个范例也可以总结出这些方式的使用场合：

（1）对于简单的运算表达式，可以使用插值表达式，比如用 {{ currency(item.count*item.price,2) }}表示商品的小计金额。

（2）如果运算逻辑复杂，并且运算过程不通用，可以使用计算属性，例如用 total 计算属性表示总金额。

（3）如果运算逻辑复杂，并且运算过程很通用，可以使用方法。例如用 currency()方法对金额数字进行格式化。它的参数可以是商品单价、小计金额和总金额。

从语义上来区分，计算属性具有特定的属性特征，例如 total 计算属性表示总金额。而方法实现了通用的运算功能，例如 currency()方法可以对所有金额数字进行通用的格式化。

3.2 数据监听

如果 Vue 组件的一个变量 num 会被频繁更新，并且当变量 num 每次被更新时，需要进行一系列耗时的操作，比如访问远程服务器的资源，或者通过复杂耗时的运算更新那些依赖变量 num 的其他变量（比如 result 变量）。在这种情况下，可以通过 Vue 框架的数据监听器 Watcher 来实现对变量 num 的监听。

Vue 的 watch 选项会通过 Watcher 来监听数据。例程 3-6 的 mywatch.html 演示了 watch 选项的基本用法。

例程 3-6 mywatch.html

```
<div id="app">
  <p><input v-model="num" /></p>
  <p>{{ result }}</p>
</div>

<script>
  const vm=Vue.createApp({
    data(){
      return{  num: 0, result: 0  }
    },
    methods:{
      //睡眠方法，参数 numberMillis 是睡眠的毫秒数
      sleep(numberMillis) {
        var now = new Date();
        var exitTime = now.getTime() + numberMillis
        while (true) {
          now = new Date()
```

```
            if (now.getTime() > exitTime)
              return;
          }
        }
      },
      watch: {
        //当 num 变量被更新，就会调用此函数
        //newNum 参数表示更新后的 num 变量
        //oldNum 参数表示更新前的 num 变量
        num(newNum, oldNum) {   //num 变量的监听函数
          this.sleep(2000)  //睡眠 2 秒
          this.result=Math.sqrt(newNum)   //计算平方根
        }
      }
    }).mount('#app')
  </script>
 </body>
</html>
```

在 mywatch.html 中，Vue 应用实例的 watch 选项中有一个 num()函数，负责监听 num 变量。当 num 变量被更新，Vue 的数据监听器就会调用这个 num()函数。

通过浏览器访问 mywatch.html，会得到如图 3-8 所示的网页。在网页的 num 变量的输入框输入新的数字，Vue 的数据监听器就会调用 num()函数，更新 result 变量的值。

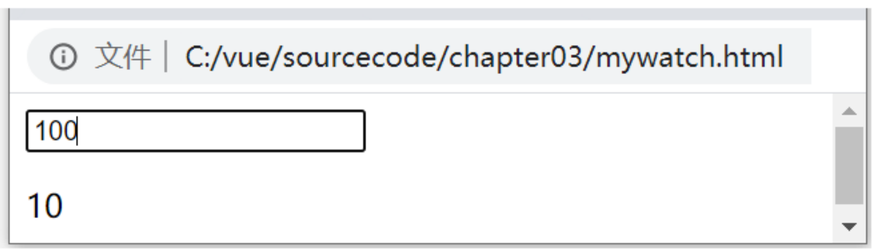

图 3-8　mywatch.html 的网页

除了通过 watch 选项来监听特定数据，还可以调用$watch()方法来监听数据。本书第 11 章的 11.12.1 节（导航后抓取）的 ItemPostFetch.vue 演示了$watch()方法的用法。

3.2.1　用 Web Worker 执行数据监听中的异步操作

对于 3.2 节的例程 3-6 的 mywatch.html，当用户在网页的 num 变量的输入框输入新的数字时，num()函数就会被 Vue 的数据监听器调用。num()函数会先调用 sleep(2000)方法睡眠 2 秒，通过这种睡眠的方式来模拟耗时的操作。

num()函数是由浏览器的负责执行 JavaScript 脚本的主线程来执行的。当主线程执行 sleep(2000)方法睡眠时，网页处于卡死状态，不能响应用户的任何操作。只有当主线程执行完 num()函数，重新更新了网页，网页才能继续响应用户的操作。

如果希望用户始终可以和网页进行顺畅地交互，不会出现网页卡死的情况，可以通过一个额外的线程来异步执行耗时的操作。本节会利用 HTML5 中的 Web Worker 线程来执行耗时操作。

首先创建一个 longtask.js 文件（文件名可以任意取），参见例程 3-7，它的 onmessage 函数包含了 Worker 线程接收到主线程发送的数据时所执行的操作。

例程 3-7　longtask.js

```
//睡眠函数，参数 numberMillis 是睡眠的毫秒数
function sleep(numberMillis) {……}

//当 Worker 线程接收到主线程发送的数据时，调用此函数
onmessage=function(event){
  var num=event.data  //读取主线程发送过来的数据
  sleep(2000)  //睡眠 2 秒，模拟耗时的操作
  var result=Math.sqrt(num)  //求平方根
  postMessage(result)   //向主线程发送运算结果
}
```

在 onmessage 函数中，event.data 表示主线程发送过来的 num 变量。postMessage(result) 用于向主线程发送 result 变量。

例程 3-8 的 mywatch-async.html 会通过 Worker 线程来执行耗时操作。

例程 3-8　mywatch-async.html

```
<div id="app">
  <p><input v-model="num" /></p>
  <p>{{ result }}</p>
</div>

<script>
  const vm=Vue.createApp({
    data(){
      return{  num: 0, result: 0 }
    } ,

    watch: {
      // 当 num 变量被更新，就会调用此方法
      num(newNum, oldNum) {   // num 变量的监听函数
        this.result='正在运算,请稍后...'
        //创建 Worker 线程
        var worker=new Worker('longtask.js')

        //注册监听接收 Worker 线程发送数据的函数
        worker.onmessage=(event)=>this.result=event.data

        //向 Worker 线程发送数据
        worker.postMessage(newNum)
      }
```

```
        }
      }).mount('#app')
    </script>
```

在 num()函数中，浏览器的主线程先通过以下语句为 result 变量赋予一个临时取值：

```
this.result='正在运算,请稍后...'
```

接着主线程通过“new Worker('longtask.js')”语句创建了 Worker 线程。接下来执行以下语句注册用于监听接收数据的 onmessage()函数：

```
worker.onmessage=(event)=>this.result=event.data
```

当主线程接收到 Worker 线程发送的数据时，就会执行 worker.onmessage()函数中的“this.result=event.data”语句，event.data 表示 Worker 线程发送的数据。

主线程接着向 Worker 线程发送 newNum 变量：

```
 worker.postMessage(newNum)
```

当笔者在创作此书时，各个浏览器对 Web Worker 的支持程度不一样。如果在 Chrome 浏览器中访问本地的 mywatch-async.html，然后在网页的输入框修改 num 变量的值，浏览器会产生以下错误：

```
Uncaught (in promise) DOMException: Failed to construct 'Worker':
Script at 'file:///C:/vue/sourcecode/chapter03/longtask.js'
cannot be accessed from origin 'null'.
```

这是因为 Chrome 出于安全的原因，不允许使用本地的 Web Worker 线程。笔者把该范例发布到了 JavaThinker.net 网站上，网址如下：

```
www.javathinker.net/vue/mywatch-async.html
```

通过浏览器访问上述网址，就可以正常访问 mywatch-async.html。在网页的输入框修改 num 变量的值，网页不会卡死，主线程会先显示 result 变量的临时取值，参见图 3-9。过 2 秒后，主线程再显示由 Worker 线程运算得到的 result 变量。

图 3-9　网页显示 result 变量的临时取值

图 3-10 展示了主线程和 Worker 线程的通信以及交换数据的过程。

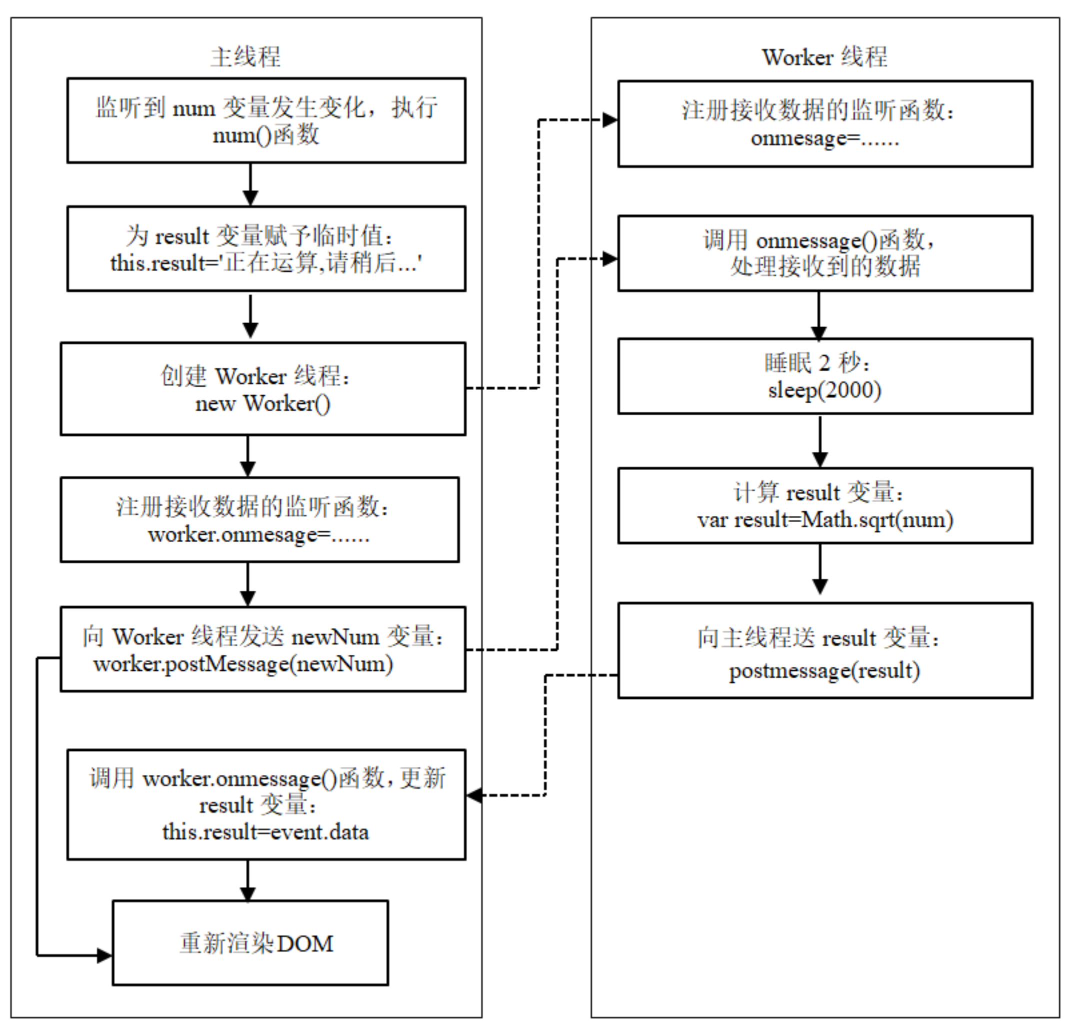

图 3-10 主线程和 Worker 线程的通信以及交换数据的过程

从图 3-10 可以看出，当主线程通过 worker.postmessage(newNum)方法，向 Worker 线程发送 newNum 变量，就会触发 Worker 线程执行 longtask.js 中的 onmessage()函数。当 Worker 线程通过 postmessage(result)方法，向主线程发送 result 变量，就会触发主线程执行 worker.onmessage()函数。无论是主线程还是 Worker 线程，都可以通过 event.data 来读取对方发送的数据。

3.2.2 在 watch 选项中调用方法

在 watch 选项中还可以调用方法。例如在例程 3-9 的 score.html 中，如果 score 变量被更新，Vue 的数据监听器就会调用 judge()方法。

例程 3-9 score.html

```
<div id="app">
  <p><input v-model="score" /></p>
  <p>{{ result }}</p>
</div>
```

```
<script>
  const vm=Vue.createApp({
    data(){
      return{  score: '', result: ''  }
    } ,

   methods:{
      judge() {
        if(this.score>=60)
           this.result='及格'
        else
           this.result='不及格'
       }
    },

   watch: {
      score: 'judge'    //调用 judge()方法
    }
  }).mount('#app')
</script>
```

下面对 judge()方法做一些修改，使它通过 JavaScript 语言的 setTimeout()函数来执行异步操作：

```
judge() {
  this.result="正在运算,请稍后..."
  setTimeout( ()=>{
    if(this.score>=60)
      this.result='及格'
    else
      this.result='不及格'
  },2000)  //延迟 2 秒后再执行运算
}
```

以上 judge()方法先给 result 变量赋予了一个临时值“正在运算,请稍后...”，然后利用 setTimeout()函数设置了异步操作：过两秒后计算 result 变量的取值。judge()方法产生的运行效果是，网页上首先显示“正在运算,请稍后...”，过两秒后再显示 reuslt 变量的实际取值。

3.2.3　比较同步操作和异步操作

按照多个操作之间的执行顺序划分，可分为同步操作和异步操作。所谓同步操作，是指一个操作执行完后，才能执行另一个操作，例如对于组件的 watch 选项的以下 num()函数：

```
num(newNum, oldNum) {   //num 变量的监听函数
  this.result="正在运算,请稍后..."       //第一行
  this.sleep(2000)  //第二行
  this.result=Math.sqrt(newNum)  //第三行
}
```

num()方法中的三行代码都是同步操作，主线程依次执行完以上三行代码，才能执行其

他的操作，例如依据当前的 result 变量重新渲染 DOM。因此，尽管在 num()函数中把 result 变量的值更新了两次，但是在网页上只看到最终更新后的 result 变量的值，而无法看到 result 变量的临时值“正在运算,请稍后...”。

所谓异步操作，是指多个操作可以各自独立运行。前端的异步操作有两种执行方式：

（1）多线程执行方式。通过多个线程来同时执行不同的异步操作。这就好比由两个人分别烧开水和扫地，可以同时进行。3.2.1 节就是通过单独的 Web Worker 线程来执行耗时的计算任务，而主线程会执行 Vue 框架的主流程。值得注意的是，早期的 JavaScript 版本并不支持多线程，因为这种运行方式会增加客户端的运行负荷，如果 JavaScript 脚本设计不合理，还会给客户机器带来安全隐患。

（2）单线程执行方式。把多个异步操作放在一个异步队列中，由主线程以轮询的方式来执行异步队列中的异步操作。这就好比一个人同时进行烧开水和扫地两个操作，当水未烧开的时候就扫地，同时会经常观测水的状态，如果水已经烧开，就把水倒进水壶，再继续扫地。

3.2.2 节介绍的 setTimeout()函数就是通过单线程方式来执行异步操作。对于 3.2.2 节的以下 judge()函数：

```
judge() {
  this.result="正在运算,请稍后..."   //第一行
  setTimeout( ()=>{                  //第二行
    if(this.score>=60)
      this.result='及格'
    else
      this.result='不及格'
  },2000)  //延迟 2 秒后再执行运算
}
```

judge()函数中的第一行和第二行代码是同步操作，但是 setTimeout()函数中的第一个参数指定的函数包含一段异步操作，这段异步操作会放在异步队列中，主线程过两秒后再执行这段异步操作。在这两秒时间内，主线程可以执行 Vue 框架的其他操作，比如渲染 DOM，把网页上的{{result}}渲染为“正在运算,请稍后...”。等过了两秒，主线程执行了上述计算 result 变量实际取值的异步操作后，再重新渲染 DOM，再次更新网页上的{{result}}。

3.2.4　深度监听

默认情况下，当 Vue 的 watch 选项监听一个对象时，不会监听对象的属性的变化。如果希望监听对象的属性变化，可以在 watch 选项中把 deep 属性设为 true，这样就会支持深度监听。

在例程 3-10 的 student.html 中，Vue 的 watch 选项会监听 student 对象，由于 deep 属性设为 true，当 student.score 属性被更新，watch 选项中的 handler()函数也会被执行。

例程 3-10 student.html

```
<div id="app">
  <p><input v-model="student.score" /></p>
  <p>{{ result }}</p>
</div>

<script>
  const vm=Vue.createApp({
    data(){
      return{
        student: { name:'Tom',score: '98'} ,
        result: ''
      }
    } ,

    watch: {
      student: {  //监听 student 对象
        handler(newStudent,oldStudent){
          if(this.student.score>=60)
            this.result='及格'
          else
            this.result='不及格'
        },
        deep: true  //启用深度监听
      }
    }
  }).mount('#app')
</script>
```

通过浏览器访问 student.html 网页，在输入框修改 student.score 属性的值，Vue 的数据监听器会调用 handler()函数，更新 result 变量。

当 Vue 的数据监听器深度监听一个对象时，不管对象的属性嵌套了多少层，只要属性发生变化，就会被监听。

3.2.5 立即监听

通过浏览器访问 3.2.4 节的例程 3-10 的 student.html 时，会看到网页上一开始显示{{student.score}}的值为 98，而{{result}}的值为“”。因为这时候 Vue 的数据监听器还没有监听到 student.score 属性的变化，因此不会调用 watch 选项中的 handler()函数。

在 Vue 组件的生命周期中，如果希望在它的初始化阶段，Vue 框架就会调用一次 watch 选项中的 handler()函数，为 result 变量赋值，那么可以把 watch 选项的 immediate 属性设为 true。

下面对 student.html 做如下修改，增加“immediate: true”的语句：

```
watch: {
  student: {
    handler(newStudent,oldStudent){……},
    immediate: true,
    deep: true
  }
}
```

再次通过浏览器访问 student.html，会看到网页上{{student.score}}的初始值为 98，{{result}}的初始值为“及格”。

3.2.6　比较计算属性和数据监听 watch 选项

计算属性和数据监听 watch 选项表面上看能完成一些相同的功能。例如当一个变量发生变化时，两者都能更新那些依赖这个变量的其他数据。

但是，由于计算属性和数据监听 watch 选项有不同的特长，所以它们有着不同的使用场合。watch 选项最擅长的是执行耗时的异步操作，本章 3.2.1、3.2.2 和 3.3.3 节已经对此做了介绍。而在只需同步更新变量的场合，使用计算属性能使程序代码更加简洁，并且具有更好的运行性能。

例程 3-11 的 fullname-watch.html 在 watch 选项中监听数据，和 3.1.1 节的例程 3-2 的 fullname.html 具有同样的功能，都能对 firstName、lastName 和 fullName 进行同步更新。

例程 3-11　fullname-watch.html

```
<div id="app">
  <p>First name: <input type="text" v-model="firstName"></p>
  <p>Last name: <input type="text" v-model="lastName"></p>
  <p>Full  name: <input type="text" v-model="fullName"></p>
  <p>{{ fullName }}</p>
</div>

<script>
  const vm = Vue.createApp({
    data() {
      return {
        firstName: 'Tom',
        lastName: 'Smith',
        fullName:  'Tom Smith'
      }
    },

    watch: {
      firstName(newValue) {
        console.log('call firstName(newValue)')
        this.fullName = newValue + ' ' + this.lastName
```

```
          },
          lastName(newValue) {
            this.fullName = this.firstName + ' ' + newValue
          },
          fullName(newValue) {
            var names = newValue.split(' ')
            this.firstName = names[0]
            this.lastName = names[names.length - 1]
          }
        }
      }).mount('#app')
    </script>
```

比较 fullname.html 和 fullname-watch.html，会发现两者有以下区别：

（1）fullname.html 的代码更加简洁。在 fullname.html 的 computed 选项中，只需为 fullName 计算属性定义 get 和 set 函数。

（2）fullname-watch.html 的代码看起来更繁琐，需要在 data 选项中定义 fullName 变量，并且需要在 watch 选项中监听 firstName、lastName 和 fullName 这个三个变量。

下面分别把 fullname.html 和 fullname-watch.html 的模板中显示 fullName 信息的代码注释掉：

```
<!--
     <p>Full  name: <input type="text" v-model="fullName"></p>
     <p>{{ fullName }}</p>
-->
```

再通过浏览器分别访问 fullname.html 和 fullname-watch.html，修改网页上 firstName 变量的输入框的内容，从浏览器的控制台观察输出日志，会发现对于 fullname.html 中的 fullName 计算属性，由于在模板中不需要显示它，因此它的 get 函数不会被调用。

而对于 fullname-watch.html，只要 firstName 变量发生变化，watch 选项中的 firstName(newValue)函数就会被调用。

由此可见，计算属性比 watch 选项具有更好的运行性能。如果在页面上不需要显示和计算属性有关的数据，那么即使它所依赖的变量发生变化，Vue 框架也不会调用计算属性的 get 函数。

3.3 Vue 的响应式系统的基本原理

Vue 框架能够对数据的更新快速作出响应。Vue 的响应式系统依赖于三个重要的类：

（1）Observer 类：数据观察器，负责观察数据的更新。如果观察到数据更新，就把数据更新消息发送给 Dep 类。

（2）Dep 类：响应式系统的调度器，负责接收来自 Observer 的数据更新消息，并把这个消息发送给相应的 Watcher 对象。

（3）Watcher 类：数据监听器。负责接收来自 Dep 的数据更新消息，执行相应的响应操作。

如图 3-11 所示，当用户更新了一个变量后，Observer 观察到这种更新，就会把更新消息发送给 Dep，Dep 再把更新消息发送给所有依赖这个变量的 Watcher，Watcher 会执行相应的操作，来对变量更新做出响应。

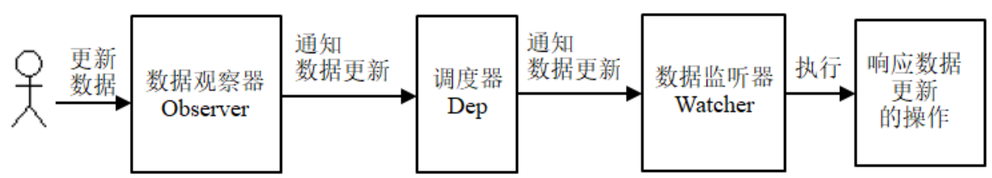

图 3-11 Vue 的响应式系统

数据监听器 Watcher 分为三种：

（1）常规 Watcher（normal-watcher）：对于在组件的 watch 选项中监听的变量，就使用这种 normal-watcher。当被监听的变量发生变化，normal-watcher 会立即执行 watch 选项中的相应函数。

（2）计算属性 Watcher（computed-watcher）：对于在 computed 选项中定义的计算属性，就使用 computed-watcher。对于每个计算属性，都对应一个 computed-watcher 对象。computed-watcher 具有 lazy（懒计算）特性：假定计算属性 b 依赖变量 a，当变量 a 更新时，computed-watcher 并不会立即重新计算 b，而是只有当需要读取 b 时，才会重新计算 b。本章 3.2.6 节的末尾也通过实验演示了这种 lazy 特性。

（3）渲染 Watcher（render-watcher）：每一个 Vue 组件都会有相应的 render-watcher, 当 Vue 组件的 data 选项中的变量或者 computed 选项中的计算属性发生变化时，render-watcher 就会重新渲染组件的 DOM。

以上三种 Watcher 有固定的执行顺序，按照先后顺序分别是：normal-watcher、computed-watcher 和 render-watcher。

三种 Watcher 的执行顺序可以保证数据在业务逻辑上的一致性。例如在例程 3-12 的 relation.html 中，变量 b 依赖变量 a，计算属性 c 依赖变量 b，在 watch 选项中会监听变量 a。在模板中会通过插值表达式{{a}}、{{b}}和{{c}}显示这三个变量的值。这三个变量的关系为：

```
b=a*10
c=b*10
```

例程 3-12　relation.html

```
<div id="app">
  <p><input type="text" v-model="a" /></p>
  <p>{{a}},{{b}},{{c}}</p>
</div>

<script>
  const vue=Vue.createApp({
    data(){
      return {a:0,b:0}
    },
    computed:{
      c(){
        console.log('计算 c')
        return this.b*10   //计算属性 c 依赖变量 b
      }
    },

    watch:{
      a(){
        console.log('开始监听 a')
        this.b=this.a*10  //变量 b 依赖变量 a
        console.log('结束监听 a')
      }
    }

  }).mount('#app')
</script>
```

如图 3-12 所示，在 relation.html 的网页上，修改变量 a 的输入框的值，会看到网页上变量 b 和计算属性 c 都会随之变化。再观察浏览器的控制台输出的日志，会看到 watch 选项以及 computed 选项的函数先后输出以下日志：

```
开始监听 a
结束监听 a
计算 c
```

由此可见，当变量 a 被更新后，Vue 框架先通过 normal-watcher 计算变量 b，再通过 computed-watcher 计算 c，最后通过 render-watcher 渲染 DOM 中的{{a}}、{{b}}和{{c}}，这样就能保证变量 a、b 和 c 的数据一致性。

图 3-12　relation.html 的网页

如果在 watch 选项的监听函数中需要读取计算属性，那么 Vue 框架会立即执行计算属性的 get 函数。下面对 relation.html 的 computed 选项和 watch 选项做如下修改：

```
computed:{
  c(){
    console.log('计算 c')
    return this.a*10     //计算属性 c 依赖变量 a
  }
},

watch:{
  a(){
    console.log('开始监听 a')
    this.b=this.a*this.c    //变量 b 依赖变量 a 和计算属性 c
    console.log('结束监听 a')
  }
}
```

再次通过浏览器访问 relation.html，修改网页中变量 a 的输入框的值，会看到浏览器的控制台输出以下日志：

```
开始监听 a
计算 c
结束监听 a
```

由此可见，当 Vue 框架在执行变量 a 的监听函数时，在执行“this.b=this.a*this.c”语句之前，会先执行计算属性 c 的 get 函数。

3.4　小结

本章主要介绍了 Vue 的 computed 选项和 watch 选项的用法。computed 选项用来定义计算属性，它具有更好的运行性能，只有当计算属性所依赖的变量发生变化，并且在需要读取计算属性的场合，才会执行它的 get 函数。watch 选项用来监听变量，只要被监听的变量发生变化，就会立即执行相应的监听函数，这种监听函数可用来执行耗时的异步操作。

Vue 的响应式系统的三个核心类是：数据观察器 Observer、调度器 Dep 和数据监听器 Watcher。Observer 负责观察数据的更新，Dep 负责调度相应的 Watcher，Watcher 负责执行

相应的操作来响应数据的更新。

Wacher 分为三种：normal-watcher、computed-watcher 和 render-watcher。假定一个 Vue 组件的 data 选项中有一个变量 a，computed 选项中有一个计算属性 b，计算属性 b 依赖变量 a，watcher 选项会监听变量 a。在组件的模板中有一个插值表达式{{a+b}}。当变量 a 发生变化时，三种 Watcher 会依次完成以下操作：

（1）normal-watcher 执行 watcher 选项中变量 a 的监听函数。

（2）computed-watcher 重新计算 b。

（3）render-watcher 重新渲染插值表达式{{a+b}}。

3.5　思考题

1．关于计算属性，以下哪些说法正确？（多选）

（a）计算属性在 computed 选项中定义。

（b）当计算属性所依赖的变量被更新，Vue 框架会调用计算属性的 set 函数。

（c）计算属性在 watch 选项中定义。

（d）当计算属性被更新，Vue 框架会调用计算属性的 set 函数。

2．组件的哪个选项会通过 normal-watcher 来监听数据？（单选）

（a）data 选项　　　（b）watch 选项

（c）methods 选项　　（d）computed 选项

3．以下是 test.html 的主要代码：

```
<div id="app">
  <p>{{getB()}}</p>
</div>

 <script>
  const vue=Vue.createApp({
    data(){
      return {a:10}
    },

    computed:{
      b(){
        return this.a*10
      }
    },

    methods:{
      getB(){
```

```
          this.b=this.a*100
          return this.b
        }
      }
    }).mount('#app')
  </script>
```

通过浏览器访问 test.html，会出现哪些情况？（多选）

（a）网页上显示{{getB()}}的值为 100

（b）网页上显示{{getB()}}的值为 1000

（c）网页上显示{{getB()}}的值为 10

（d）浏览器的控制台显示执行“this.b=this.a*100"语句发生错误，错误原因为：Write operation failed: computed property "b" is readonly。

4．以下是 example.html 的主要代码：

```
<div id="app">
  <p>{{b}}</p>
</div>

 <script>
  const vue=Vue.createApp({
    data(){
      return {a:10,b:0}
    },

    watch:{
      a(){
        this.b=this.a*10
      }
    }
  }).mount('#app')
</script>
```

通过浏览器访问 example.html，网页上显示{{b}}的值是什么？（单选）

（a）{{b}}　　　　　　　　（b）100

（c）1000　　　　　　　　（d）　0

5．对第 4 题的 example.html 中的 watch 选项做如下修改：

```
watch:{
  a:{
    handler(){
      this.b=this.a*10
    },
    immediate:true
  }
}
```

通过浏览器访问 example.html，网页上显示{{b}}的值是什么？（单选）

（a）{{b}}　　　　（b）100

（c）1000　　　　（d） 0

6．当组件的变量被更新时，组件的哪个选项适合设定用于立即响应变量更新的异步操作？（单选）

（a）data 选项　　　　（b）methods 选项

（c）watch 选项　　　　（d）computed 选项