```
$ hexo n vue-skill two-way-biding
```

前言

双向绑定 是 vue 的一大特点之一，使用也非常方便。

在需要双向绑定的**表单控件元素**使用指令 v-model即可。

官方文档教程 —— 表单输入绑定（传送门）

官方文档api —— v-model 指令（传送门）

注意了：到目前为止，**双向绑定只是针对表单控件元素，并没有说明其他DOM元素，或者是自定义组件。**

那我们要如何实现在组件（或者说其他DOM元素）的双向绑定呢？

```
<base-components v-model="fieldValue"></base-components>
```

v-model 浅析

> v-model 本质上不过是语法糖。它负责监听用户的输入事件以更新数据，并对一些极端场景进行一些特殊处理。
>
> v-model 会忽略所有表单元素的 value、checked、selected attribute 的初始值而总是将 Vue 实例的数据作为数据来源。你应该通过 JavaScript 在组件的 data 选项中声明初始值。
>
> v-model 在内部为不同的输入元素使用不同的属性并抛出不同的事件：
>
> - text 和 textarea 元素使用 value 属性和 input 事件；
> - checkbox 和 radio 使用 checked 属性和 change 事件；
> - select 字段将 value 作为 prop 并将 change 作为事件。

节选几段官方文档的资料，不难察觉v-model 会占用一个 prop 属性和一个 $emit 事件。如果有冲突，也是可以更改的。

如果还是好奇 v-model的实现原理，我可以告诉你大概是这样。

> **双向绑定基于getter/setter 结合观察者模式实现**
>
> getter/setter指Object.defineProperty遍历vue实例的data选项
>
> 观察者模式是js的一种设计模型，可以实现发布订阅功能

组件使用双向绑定（基本类型版）

<!-- 组件使用双向绑定并不罕见，很多vue的ui框架包装的组件都支持直接使用v-model指令。 -->

有了相应的资料，我们可以开工了。准备一个组件 base-input.vue

```
export default {
    name: 'base-input',
    props: {
        value: {
            type: String,
            default: ''
        }
    },
    data () {
        return {
            fieldValue: this.value // 初始化赋值
        }
    },
    watch: {
        value (newVal) {
            this.fieldValue = newVal // 监听props（外部）更新，赋值给data（内部）
        },
        fieldValue () {
            this.$emit('input', this.fieldValue) // data（内部）更新，抛出到外部
        }
    }
}
<input type="text" v-model="fieldValue">
```

有几个注意点要说明下：

1. fieldValue 必须初始化赋值为 props 的 value
2. 初始化页面时 watch 并不执行
3. 内部更新，触发 fieldValue 监听函数
4. 外部更新，触发 value监听函数

- base-input 源码 传送门
- 示例 传送门

:value 与v-model

因为之前被这两个搞混过，不明白之前的联系和区别，这里单独拿出来讲一下。

- :value 是绑定一个prop value 给组件，实现外部数据传入内部， **单向绑定**
- v-mdoel **是双向绑定**，默认占用prop value 属性和一个 $emit input 事件，在:value 的基础上$emit input实现内部数据抛出外部，从而外部、内部数据达成同步且外内均可更改。

把 base-input v-model="inputValue"></base-input> 换成 :value="inputValue"试试？

我们也将在下一节更改指令默认使用的porp 属性 和$emit 事件

可其实这种方式只支持js的基本类型，像对象或者数组的引用类型，这样处理是不够的，所以我们需要对现有的双向绑定方式进行升级。

组件使用双向绑定（引用类型版）

准备另一个组件：base-div.vue

把base-input.vue 的代码复制过来，组件名，类名什么的这些标识信息该改的就改一改。

然后可能有的朋友到这里会把 prop value 的 type 改成 Object，这只是第一步而已。

因为引用类型的特性，我们手动给内部变量赋值，会更改其引用地址，故相当于重新初始化了一个变量。

这里会触发另一个watch函数，而另一个watch也具备给内部变量赋值的能力；

如果不特殊处理，这两个watch 函数触发其中一个，就会不断调用另一条，造曾死循环。

所以我们需要判断当内外部变量相等的时候，就不赋值了。嗯，**判断两个对象相等。**

判断两个对象相等可能有点尴尬， 那我们把两个对象序列化为JSON 字符串就可以了。

```
JSON.stringify(obj1) === JSON.stringify(obj2);
```

技术点已经准备够了，我们足够实现用v-model绑定一个对象了。

```
    props:  {
        value:  {
            type:  [ Object, Array]
        }
    },
    data  ()  {
        return  {
            fieldValue:  this.value,
            fieldValueStringifty:  JSON.stringify(this.value)  //  保存外部传入的变量快照，将在监听器中做比较使用
        }
    },
    watch:  {
```

```
        value: {
            deep: true, // 深度监听
            handler (newVal) {
                const currentValue = JSON.stringify(newVal)
                // 外部传入的变量与内部变量比较
                // 不相等，内部变量方可赋值为this.value
                // * 内部变量赋值，将会触发fieldValue监听函数
                // 且此时应更新fieldValue 对象字符串快照
                if (currentValue !== this.fieldValue) {
                    this.fieldValue = JSON.parse(currentValue)
                    this.fieldValueStringifty = currentValue
                }
            }
        },
        fieldValue: {
            deep: true, // 为什么要使用深度监听呢？
            handler (newVal) {
                // 自己改变内部变量，或者因外部变量更新手动赋值内部变量
                // 这个函数都会被触发
                // 要实现的是内部与外部变量同步，所以快照才是保存外部变量
                // 1.当外部变量改动，会先调用 watch value，此时内外部变量同步
                //     这时不需要抛出事件，否则出现死循环
                // 2.当内部变量改动，会先调用 watch fileldValue
                //     抛出事件，外部的v-model又会自动更新内部的value
                //     从而触发 watcu value
                //   （这里触发watch value了，可以回去看看1）
                //     如果你不会兜兜转，那我想你应该feel到了整个更新流程
                // **如果被兜住了，先理解外部变量的改动，再理解内部变量的改动
                const currentValue = JSON.stringify(newVal)
                if (currentValue !== this.fieldValueStringifty) {
                    this.$emit('input', JSON.parse(currentValue))
                }
            }
        }
    }
```

- base-div 源码 传送门
- 示例 传送门

接下来我们不使用 prop value $emit input了，我们使用 prop insert $emit output。

直接在base-div 改动吧，其实也不复杂。

```
export default {
    model: {
        prop: 'insert',
        event: 'output'
    },
  props: {
        insert: {
            type: [Object, Array]
        }
    },
    data () {
        return {
            fieldValue: this.insert,
            fieldValueStringifty: JSON.stringify(this.insert) // 保存外部传入的变量快照，将在监听器中做比较使用
        }
    },
  watch: {
    insert: {
      // ...
    },
    fieldValue: {
      // ...
      this.$emit('output', JSON.parse(currentValue))
    }
  }
}
```

然后在外部使用 @output 的时候，会发现外部更新不触发 output 事件。

刚开始以为组件实现还是有问题，后面想想不对，外部更新没必要抛出事件。

外部更新，可以在外部使用 watch 函数，这里确实不管内部组件的事情了。

```
<base-div v-model="formValue" @output="handleOutput"></base-div>
```

最后

今天我们重新了解了 v-model 只针对于表单控件数据的双向绑定，并且会默认占用一个属性和一个事事件。

也通过v-model，我们得以在自己的组件实现双向绑定，基本类型和引用类型的处理机制也不太一样。

工作中也经常用到这种需要自定义双向绑定的场景，有时候写得次数太多，就想着能不能抽象出来复用。

想想 mixis ， extend 都不太适合，因为会固定占用prop data 事件， 容易起冲突。

目前觉得指令可以一试，不过还需要好好研究一下，希望下次的更新能定义一条全局指令——base-model。