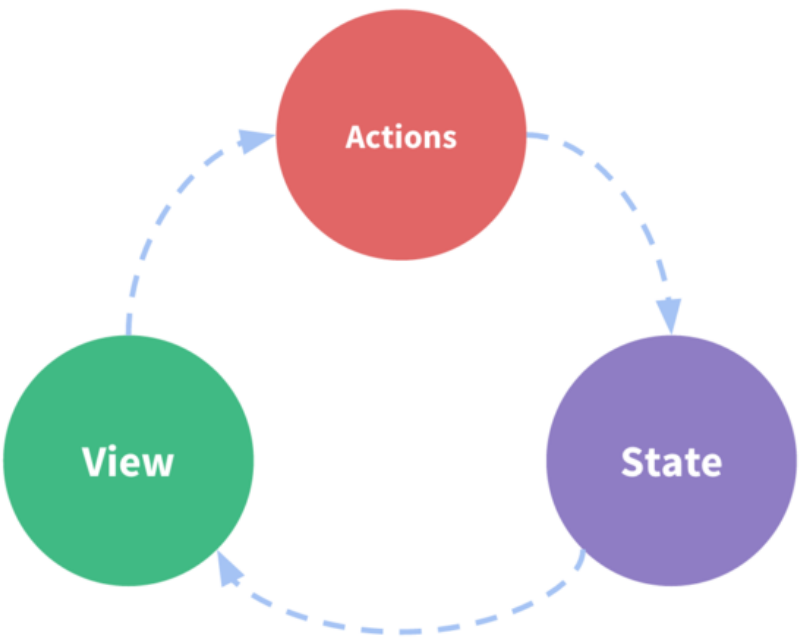
单向数据流与双向数据绑定的区别

在学习和接触 Vue 的过程中，接触到 Vue 的两个特性，单向数据流与双向数据绑定，他们是什么，特点如何，以及彼此之间的关系和联系又有什么，接下来，我们深入的探究一下它们。

单向数据流



单向数据流的极简示意

数据流，表明的是数据流向，用大白话说就是数据传递。那么单项数据流是我们的数据单一方向传输。对于 Vue 来说，组件之间的数据传递具有单向数据流这样的特性。

首先对于父子组件来说，父组件总是通过 Props 向子组件传递数据。
例如：

```
<div id="app">
```

```
        <show-title></show-title>
    </div>

     <script>
        Vue.component('showTitle', {
            template: '<div>{{ title }}</div>',
            props: ['title']
        })
        new Vue({
          el: '#app',
          data: {
            title: 'Hello Vue'
          }
        })
  </script>
```

所有的 prop 都使得其父子 prop 之间形成了一个单向下行绑定：父级 prop 的更新会向下流动到子组件中，但是反过来则不行。这样会防止从子组件意外改变父级组件的状态，从而导致你的应用的数据流向难以理解。

额外的，每次父级组件发生更新时，子组件中所有的 prop 都将会刷新为最新的值。这意味着你不应该在一个子组件内部改变 prop。如果你这样做了，Vue 会在浏览器的控制台中发出警告。

对于改变 Props 中的值存在两种情况

作为子组件的数据，并且在子组件中需要修改，在这种情况下，最好定义一个本地的 data 属性并将这个 prop 用作其初始值。

```
Vue.component('showTitle', {
        template: '<div><input v-model="myTitle"/></div>',
        props: ['title'],
        data() {
          return {
            myTitle: this.title
          }
        }
    })
```

这个 prop 以一种原始的值传入且需要进行转换。在这种情况下，最好使用这个 prop 的值来定义一个计算属性：

```
props: ['size'],
computed: {
  normalizedSize: function () {
    return this.size.trim().toLowerCase()
  }
}
```

注意在 JavaScript 中对象和数组是通过引用传入的，所以对于一个数组或对象类型的 prop 来说，在子组件中改变这个对象或数组本身将会影响到父组件的状态。
接下来在说一说双向数据绑定

双向数据绑定

当我们在前端开发中采用 MV*的模式时，M - model，指的是模型，也就是数据，V - view，指的是视图，也就是页面展现的部分。通常，我们需要编写代码，将从服务器获取的数据进行“渲染”，展现到视图上。每当数据有变更时，我们会再次进行渲染，从而更新视图，使得视图与数据保持一致。也就是：

而另一方面，页面也会通过用户的交互，产生状态、数据的变化，这个时候，我们则编写代码，将视图对数据的更新同步到数据，以致于同步到后台服务器。也就是：

能提现出来这种双向数据绑定的特点是 Vue 中的 v-model 指令

代码如下：

```
<div id="app">
    <input type="text" v-model="title">
</div>

<script>
    new Vue({
        el: '#app',
        data: {
            title: 'Hello Vue'
        }
    })
</script>
```

代码正确执行后，页面上 input 元素对应的位置会显示上面代码中给出的初始值：Hello Vue。

由于双向数据绑定已经建立，因此：

执行 vm.title = 'vue' 后，页面上 input 也会更新为显示： vue
在页面文本框中修改内容为：Javascript，则通过 vm.title 获取的值为："JavaScript"
双向数据绑定指的是单纯的数据与视图层之间的绑定。

单向数据流与双向数据绑定的联系与区别

对于非 UI 控件来说，不存在双向，只有单向。只有 UI 控件才有双向的问题。单向绑定使得数据流也是单向的，对于复杂应用来说这是实施统一的状态管理（如 Vuex）的前提。双向绑定在一些需要实时反应用户输入的场合会非常方便（比如表单提交）。但通常认为复杂应用中这种便利比不上引入状态管理带来的优势。

对于 Vue 中的 v-model 来说，他是一个语法糖，利用 bind 语法与事件来实现数据与视图之间的绑定。
v-model 可以理解下面这样的代码（不完全一致）

```
<input type="text" v-bind:value="msg" v-on:input="msg = $event.target.value">
```

实质上也是单向数据流在加上用户的操作事件来实现双向数据绑定。
搞清楚双向绑定的实现原理之后，可以看到双绑跟单向绑定之间的差异只在于，双向绑定把数据变更的操作隐藏在框架内部，调用者并不会直接感知。

单向绑定相应地使得数据流也是单向的，而在践行单向数据流的 Vuex 中，其实不过是在全局搞了一个单例的事件分发器 (dispatcher)，开发者必须显式地通过这个统一的事件机制做数据变更通知。其实这种方式跟框架对 UI 控件 上实现双向绑定的方式是一样的。底层都是事件机制。试想一下，假设在双向绑定的应用中，我们有办法修改框架对 UI 控件 自动绑定的事件 listener 或 数据 watcher，然后加上类似 dispatcher 的逻辑，双向绑定背后的状态变化我们一样可以管理起来，一样可以享用单向数据流才有的收益。

单向数据流和双向数据绑定有什么优缺点？
单向数据流优缺点
优点：
1. 所有状态的改变可记录、可跟踪，源头易追溯;
2. 所有数据只有一份，组件数据只有唯一的入口和出口，使得程序更直观更容易理解，有利于应用的可维护性;
3. 一旦数据变化，就去更新页面(data-页面)，但是没有(页面-data);
4. 如果用户在页面上做了变动，那么就手动收集起来(双向是自动)，合并到原有的数据中。
缺点：
1. HTML 代码渲染完成，无法改变，有新数据，就须把旧 HTML 代码去掉，整合新数据和模板重新渲染;
2. 代码量上升，数据流转过程变长，出现很多类似的样板代码;
3. 同时由于对应用状态独立管理的严格要求(单一的全局 store)，在处理局部状态较多的场景时(如用户输入交互较多的“富表单型”应用)，会显得啰嗦及繁琐。

双向数据绑定的优缺点：
优点：
1. 用户在视图上的修改会自动同步到数据模型中去，数据模型中值的变化也会立刻同步到视图中去；
2. 无需进行和单向数据绑定的那些相关操作；
3. 在表单交互较多的场景下，会简化大量业务无关的代码。

缺点：

1. 无法追踪局部状态的变化；
2. “暗箱操作”，增加了出错时 debug 的难度；
3. 由于组件数据变化来源入口变得可能不止一个，数据流转方向易紊乱，若再缺乏“管制”手段，血崩。

这样来看，单向绑定跟双向绑定在功能上基本上是互补的，所以我们可以在合适的场景下使用合适的手段。比如在 UI 控件 中(通常是类表单操作)，我会使用双向的方式绑定数据；而其他场景则统一采用
单向 + inline event
(<component :msg="msg" @update="updateMsg(msg)"></component>) 的方式构建应用。