1. vue 2.x 的安装

```
# 安装 vue 脚手架工具
npm install -g vue-cli

# 安装 webpack 模板, my-project 为工程名称
vue init webpack "my-project"

# 启动项目
cd "my-project"
npm install
npm run dev
```

2. 项目结构

```
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----           2017/2/4     15:01            build
d----           2017/2/4     15:01            config
d----           2017/2/4     15:05            node_modules
d----           2017/2/4     15:01            src
d----           2017/2/4     15:01            static
-a---           2017/2/4     15:01        200 .babelrc
-a---           2017/2/4     15:01        147 .editorconfig
-a---           2017/2/4     15:01         23 .eslintignore
-a---           2017/2/4     15:01        642 .eslintrc.js
-a---           2017/2/4     15:01         44 .gitignore
-a---           2017/2/4     15:01        195 index.html
-a---           2017/2/4     15:01       1772 package.json
-a---           2017/2/4     15:01        448 README.md

# build: webpack相关配置文件
# config: webpack相关配置文件
# node_modules: npm 依赖代码库
# src: 项目源码
# static: 第三方静态资源目录
# .babelrc: babel 配置
# .editorconfig: 编辑器配置
# .eslintignore:  忽略语法检查的目录文件
# .eslintignore: eslint 配置文件
# .gitignore: 忽略提交 git 的目录文件
# index.html: 项目入口文件
```

```
# package.json: 项目配置文件
```

3. 基本用法

3.1 Hello World 起步

```
# Hello World 例子

// 引入 Vue
<script src="https://unpkg.com/vue/dist/vue.js"></script>

---------------------------------

// 模板:
<div id="app">
  // 声明式渲染
  {{ message }}
</div>

/* 创建一个 vue 对象, 同时将这个对象挂载到 #app 的元素上 */
var app = new Vue({
  // 挂载点
  el: '#app',
  // Vue 对象中管理的数据 VM ( ViewModel ), 可以直接在模板上通过声明来进行数据访问
  data: {
    message: 'Hello Vue!'
  }
})
```

3.2 模板语法

```
{{}}

# 模板:

<div>Message: {{ msg }} </div>
```

Mustache 标签将会被替代为对应数据对象上 msg 属性的值。无论何时，绑定的数据对象上 msg 属性发生了改变，插值处的内容都会更新。

```
# v-html 指令

// 模板:

<div v-html="rawHtml"></div>

new Vue({
    el: "#app",
    data() {
        return {
            rawHtml: "<h1>hello world</h1>"
        }
    }
})
```

双大括号会将数据解释为纯文本，而非 HTML 。为了输出真正的 HTML ，你需要使用 v-html 指令
被插入的内容都会被当做 HTML —— 数据绑定会被忽略。注意，你不能使用 v-html 来复合局部模板，因为 Vue 不是基于字符串的模板引擎。组件更适合担任 UI 重用与复合的基本单元。

```
# 属性
# Mustache 不能在 HTML 属性中使用，应使用 v-bind 指令
<span v-bind:title="message">
    Hover your mouse over me for a few seconds to see my dynamically
bound title!
</span>

new Vue({
    el: '#app',
    data() {
        return {
            message: 'You loaded this page on ' + new Date()
        }
    }
})
```

v-bind 属性被称为指令。指令带有前缀 v-，以表示它们是 Vue.js 提供的特殊属性。这个指令的简单含义是说：将这个元素节点的 title 属性和 Vue 实例的 message 属性绑定到一起。

```
# JavaScript 表达式
```

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div v-bind:id="'list-' + id"></div>
```

这些表达式会在所属 Vue 实例的数据作用域下作为 JavaScript 被解析。有个限制就是，每个绑定都只能包含单个表达式，所以下面的例子都不会生效。

```
# 这是语句，不是表达式
{{ var a = 1 }}
# 流控制也不会生效，请使用三元表达式
{{ if (ok) { return message } }}
```

```
v-bind 缩写:
# 完整语法
<a v-bind:href="url"></a>
# 缩写
<a :href="url"></a>

v-on 缩写:
# 完整语法
<a v-on:click="doSomething"></a>
# 缩写
<a @click="doSomething"></a>
```

Class 与 Style 绑定

```
# 绑定 HTML Class
# 表示 class active 的更新将取决于数据属性 isActive 是否为真值
<div v-bind:class="{ active: isActive }"></div>
```

我们也可以在对象中传入更多属性用来动态切换多个 class 。此外， v-bind:class 指令可以与普通的 class 属性共存。如下模板:

```
# 1. 绑定某一个 Class:

<div class="static"
     v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
```

```
data: {
  isActive: true,
  hasError: false
}

渲染为:
<div class="static active"></div>

---------------------------------

# 2. 直接绑定数据里的一个对象:

<div v-bind:class="classObject"></div>

data: {
  classObject: {
    active: true,
    'text-danger': false
  }
}

# 3. 数组语法:

<div v-bind:class="[activeClass, errorClass]">

data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}

渲染为:
<div class="active text-danger"></div>
```

```
# 绑定内联样式 style

---------------------------------

# 1. 绑定一个 style
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"
></div>

data: {
  activeColor: 'red',
  fontSize: 30
}
```

```
# 2. 绑定一个对象

<div v-bind:style="styleObject"></div>

data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}

# 3. 数组

<div v-bind:style="[baseStyles, overridingStyles]">
```

3.3 条件与循环

```
# 1. 条件 v-if

---------------------------------
模板:
<div id="app-3">
  # 通过 if-else 指令来控制元素的显示
  <p v-if="seen">Now you see me</p>
  <p v-else>look at me</p>
</div>

new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})

或

<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
```

```
<div v-else>
  Not A/B/C
</div>

# 使用 key 控制元素是否重用

<div v-if="state">
    <span>用户名</span><input type="text" key="username-input" />
</div>
<div v-else>
    <span>邮箱</span><input type="text" key="email-input" />
</div>

var app = new Vue({
    el: '#app',
    data() {
        return {
            state: true
        }
    },
    methods: {
        trans() {
            this.state = !this.state;
        }
    }
})


# 2. 循环 v-for

---------------------------------

# 数组迭代

# 模板:
<div id="app-4">
  <ol>
    <li v-for="(item, index) in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>

var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
```

```
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
  }
})

# 对象迭代

<ul>
    <li v-for="(value, key, index) in obj">{{index}}{{" "}}{{key}}:{{
value}}</li>
</ul>

data: {
    obj: {
        username: "zhangsan",
        password: "111111",
        gender: "male"
    }
}

# 整数迭代

<span v-for="n in 10">{{ n }}</span>

# 结果:

12345678910
```

3.4 处理用户输入

```
# 1. v-on 指令

# 在监听事件中来触发对 this.data 的修改
---------------------------------

# 模板:
<div id="app-5">
  <p>{{ message }}</p>
  # v-on 指令绑定一个监听事件用于调用我们 Vue 实例中定义的方法
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>

var app5 = new Vue({
```

```
    el: '#app-5',
    data: {
        message: 'Hello Vue.js!'
    },
    methods: {
        reverseMessage: function() {
           // this.message 是指的是 data 中的 message 属性
           // 当 this.data 中的属性值发生改变后，视图也会重新渲染
            this.message = this.message.split('').reverse().join('')
        }
    }
})

# 2. v-model 指令

# 在表单输入和应用状态中做双向数据绑定
---------------------------------

# 模板:
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>

var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

4. 组件

4.1 什么是组件

组件可以扩展 HTML 元素，封装可重用的代码。在较高层面上，组件是自定义元素， Vue.js 的编译器为它添加特殊功能。在有些情况下，组件也可以是原生 HTML 元素的形式，以 is 特性扩展。

4.2 组件的注册或创建

```
# 全局组件
Vue.component(tagName, options)
```

```
# 例如:

<div id="app">
    <hello-world></hello-world>
</div>

// 注册
Vue.component("helloWorld", {
    template: "<div>helloWorld</div>"
})

// 创建根实例
new Vue({
    el: "#app"
})

---------------------------------

# 局部注册：通过使用组件实例选项注册，可以使组件仅在另一个实例/组件的作用域中可用:

var helloWorld = {
    template: "<div>hello world!!!</div>"
}

new Vue({
    el: "#app",
    components: {
        "hello-world": helloWorld
    }
})
```

注意: 当使用 DOM 作为模版时（例如，将 el 选项挂载到一个已存在的元素上），你会受到 HTML 的一些限制，因为 Vue 只有在浏览器解析和标准化 HTML 后才能获取模版内容。尤其像这些元素 `<ul>`， `<ol>`， `<table>`， `<select>` 限制了能被它包裹的元素， `<option>` 只能出现在其它元素内部。

```
# 例如:

<table id="app">
  # 自定义组件 <my-row> 被认为是无效的内容，因此在渲染的时候会导致错误
  <my-tr></my-tr>
</table>
```

```
# 变通的方案，使用特殊的 is 属性
<table id="app">
    <tr is="my-tr"></tr>
</table>

var trRow = {
    template: `
        <tr>
            <td>HTML</td>
            <td>CSS</td>
            <td>JavaScript</td>
        </tr>
    `
}

new Vue({
    el: "#app",
    components: {
        "my-tr": trRow
    }
})
```

4.3 data

```
<div id="example-2">
  <simple-counter></simple-counter>
  <simple-counter></simple-counter>
  <simple-counter></simple-counter>
</div>
```

```
var data = { counter: 0 }
Vue.component('simple-counter', {
  template: '<button v-on:click="counter += 1">{{ counter }}</button>
',
  // data 是一个函数，因此 Vue 不会警告，
  // 但是我们为每一个组件返回了同一个对象引用
  data: function () {
    return data
  }
})
new Vue({
  el: '#example-2'
```

```
})
```

由于这三个组件共享了同一个 data， 因此增加一个 counter 会影响所有组件！

```
data: function () {
  return {
    counter: 0
  }
}
```

4.4 Prop

组件实例的作用域是孤立的。这意味着不能并且不应该在子组件的模板内直接引用父组件的数据。可以使用 props 把数据传给子组件。

prop 是父组件用来传递数据的一个自定义属性。子组件需要显式地用 props 选项声明 “prop”：

```
# prop 的使用

<div id="app">
    <container></container>
</div>

var container = {
    template: `
        <div>
            <span>这是容器组件</span>
            # 动态 prop
            <child :msg="message" />
        </div>
    `,
    data() {
        return {
            message: "hello!!!"
        }
    }
};

var child = {
    template: `
        <div>
            <span>这是子组件</span>
```

```
            {{ msg }}
        </div>
    `,
    props: ["msg"]
};

Vue.component("container", container);
Vue.component("child", child);

new Vue({
    el: "#app"
})
```

4.5 单向数据流

prop 是单向绑定的：当父组件的属性变化时，将传导给子组件，但是不会反过来。这是为了防止子组件无意修改了父组件的状态——这会让应用的数据流难以理解。

```
var container = {
    template: `
        <div>
            <span>这是容器组件</span>
            <input type="text" v-model="message" />
            # 动态 prop
            <child :msg="message" />
        </div>
    `,
    data() {
        return {
            message: "hello!!!"
        }
    }
};

var child = {
    template: `
        <div>
            <span>这是子组件</span>
            <input type="text" v-model="message" />
            {{ message }}
        </div>
    `,
    props: ["msg"],
    data() {
```

```
        return {
            message: this.msg
        }
    }
};

Vue.component("container", container);
Vue.component("child", child);

new Vue({
    el: "#app"
})
```

4.6 自定义事件

父组件是使用 props 传递数据给子组件，但如果子组件要把数据传递回去，应该怎样做？那就是自定义事件！

```
# 使用 $on(eventName) 监听事件
# 使用 $emit(eventName) 触发事件

---------------------------------

var container = {
    template: `
        <div>
            <span>这是容器组件</span>
            <input type="text" v-model="message" />
            // :msg 动态 prop
            // v-on:click 监听子组件 $emit 触发的事件
            <child :msg="message" v-on:click="setMessage" />
        </div>
    `,
    data() {
        return {
            message: "hello!!!"
        }
    },
    methods: {
        setMessage(msg) {
            this.message = msg;
        }
    }
};
```

```
var child = {
    template: `
        <div>
            <span>这是子组件</span>
            <input type="text" v-model="message" v-on:input="setMessa
ge" />
            {{ message }}
        </div>
    `,
    props: ["msg"],
    data() {
        return {
            message: this.msg
        }
    },
    methods: {
        setMessage() {
          // 子组件触发父级组件监听的 click 事件
            this.$emit("click", this.message)
        }
    }
};

Vue.component("container", container);
Vue.component("child", child);

new Vue({
    el: "#app"
})
```