

COURSE TITLE

# Thinking in Vue



奇舞团 前端基础课



讲师 | 刘观宇



# 刘观宇

- 360PC小程序前端负责人
- 技术经理
- 360前端高级技术专家
- W3C CSS工作组成员



# The Progressive JavaScript Framework

# 前导课程

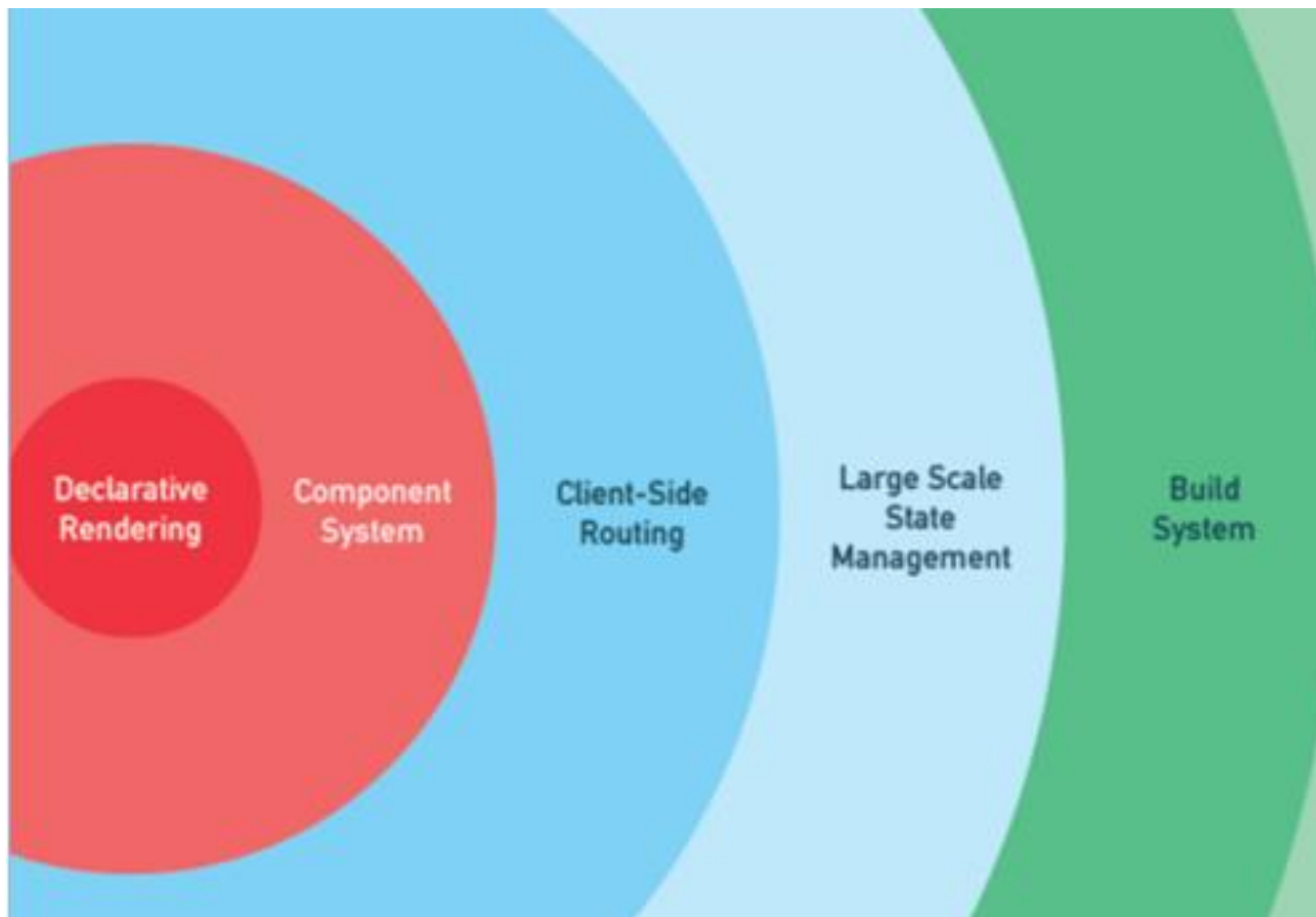
- HTML基础
- CSS基础
- JavaScript基础
- NodeJS
- 前端工程化



# 课程安排

- 概 览
- 基 础
- 生 态
- 进 阶
- 参 考

# 01 概览





# 概览

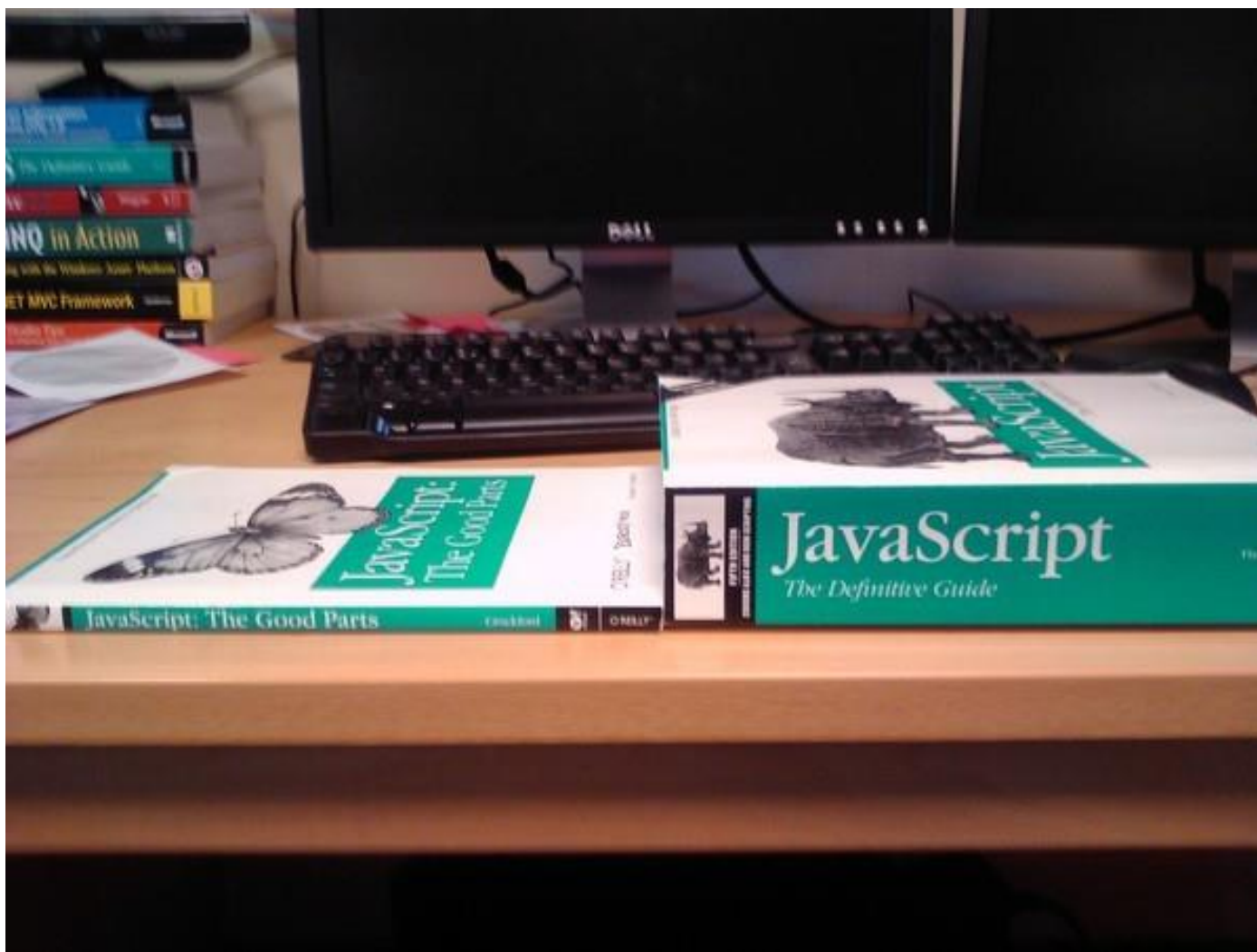
- 前端的演进
- WebComponents
- MVC、MVP、MVVM、Flux
- Vue的特点



# 01 概览

1/4 前端的演进





- 组件化与封装
- 数据与表现解耦
- 性能优化
- Debug

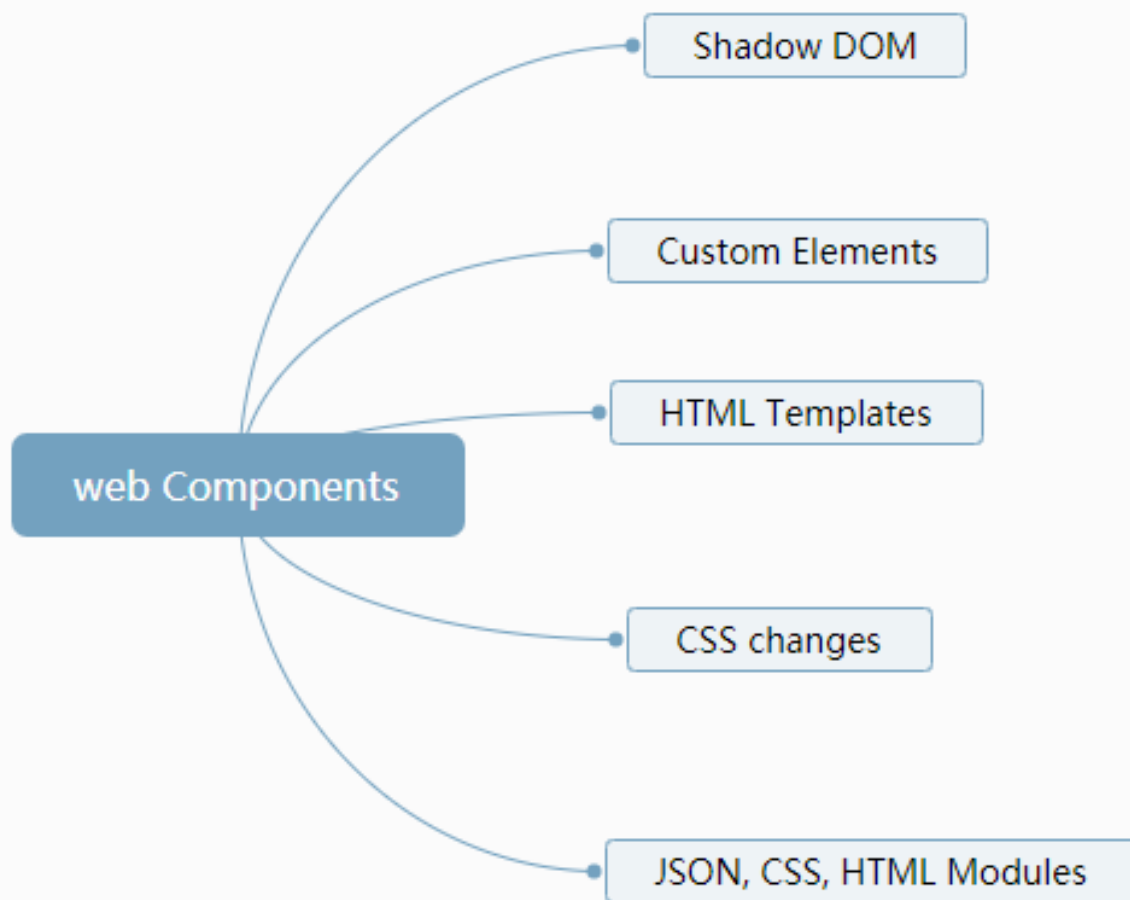


# 01 概览

2/4 Web Components

# 一些关于组件化的技术

- HTML Components （IE5~IE9 IE10 废弃）
- XML Binding Language （FireFox 2012年废弃）
- Web Components



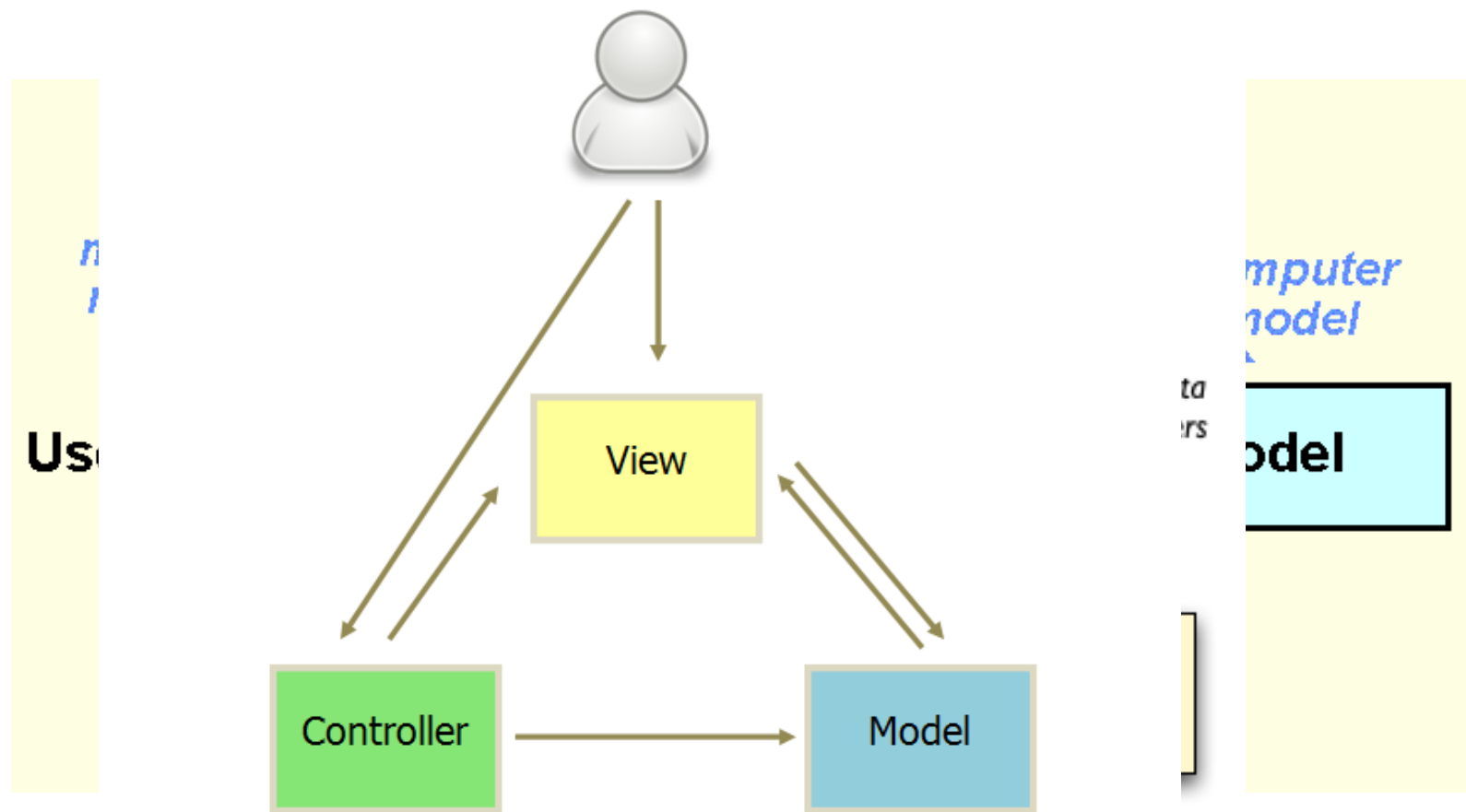
<https://www.caniuse.com/#search=web%20components>



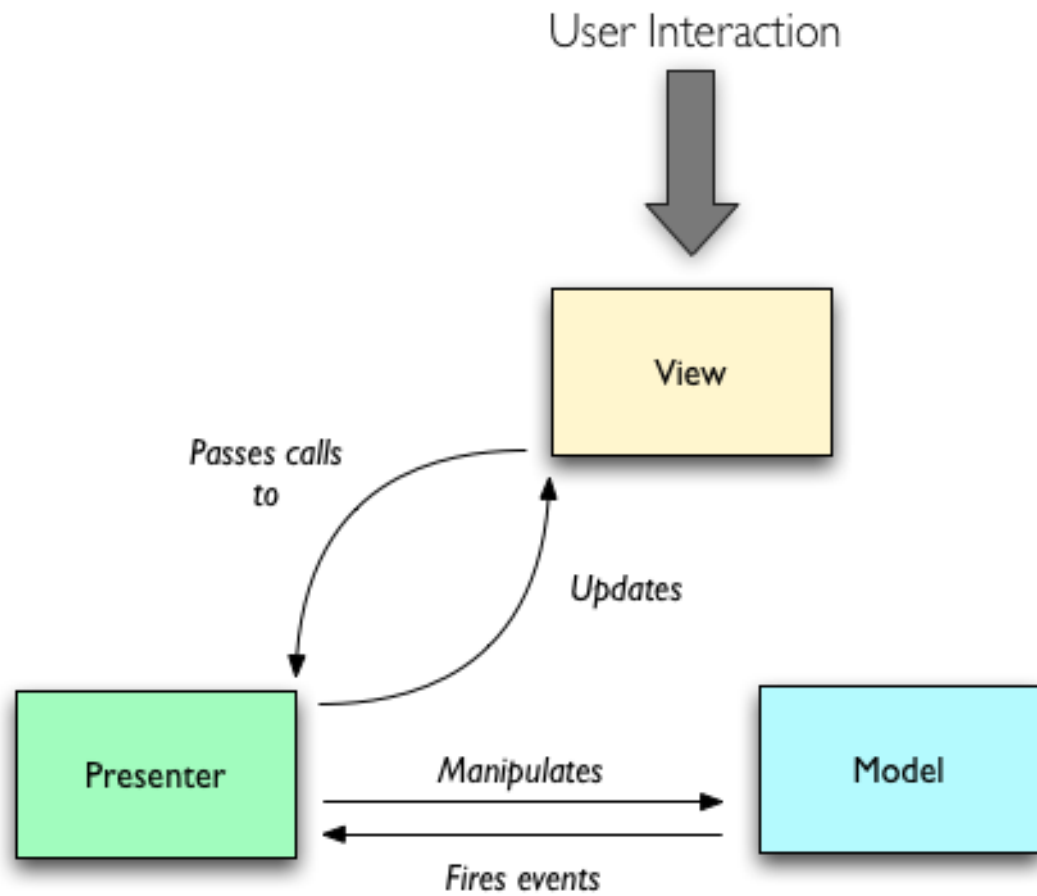
# 01 概览

3/4 MVC、MVP、MVVM、Flux

# M - V - C

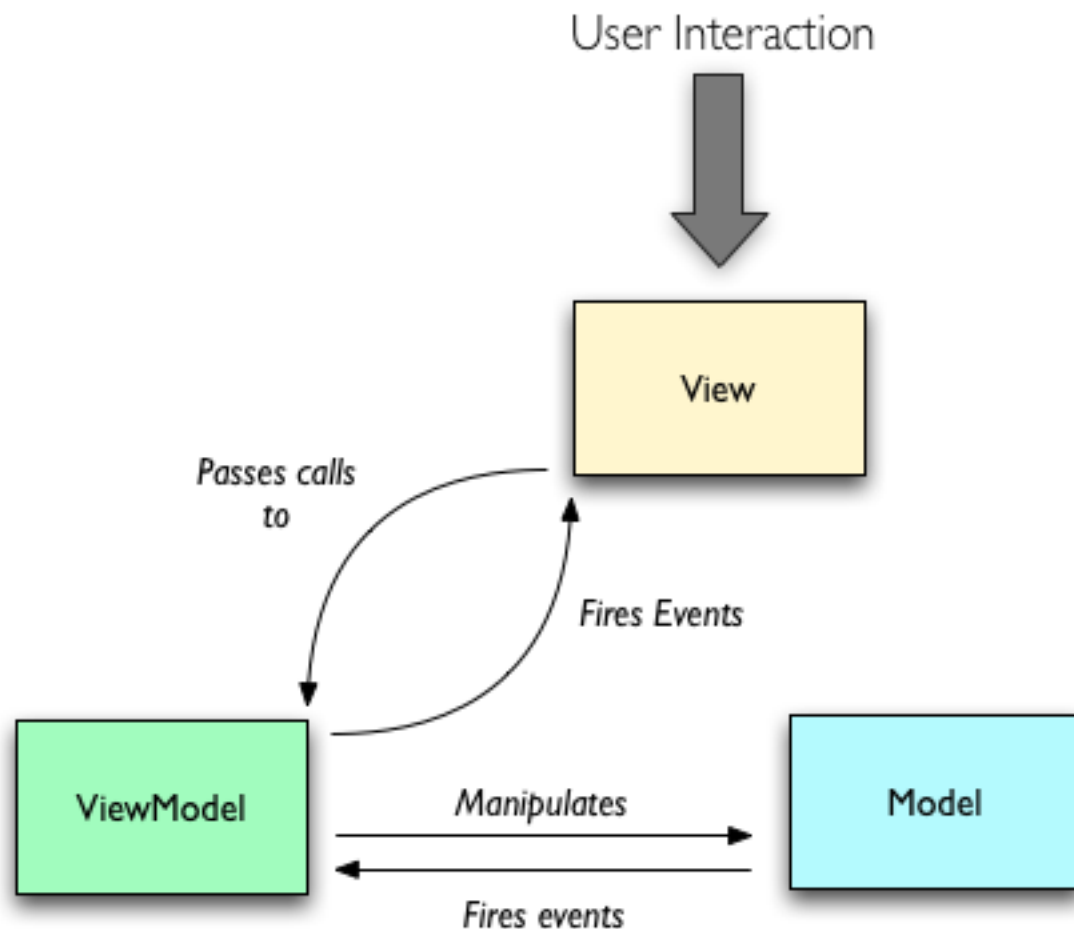


# M - V - P



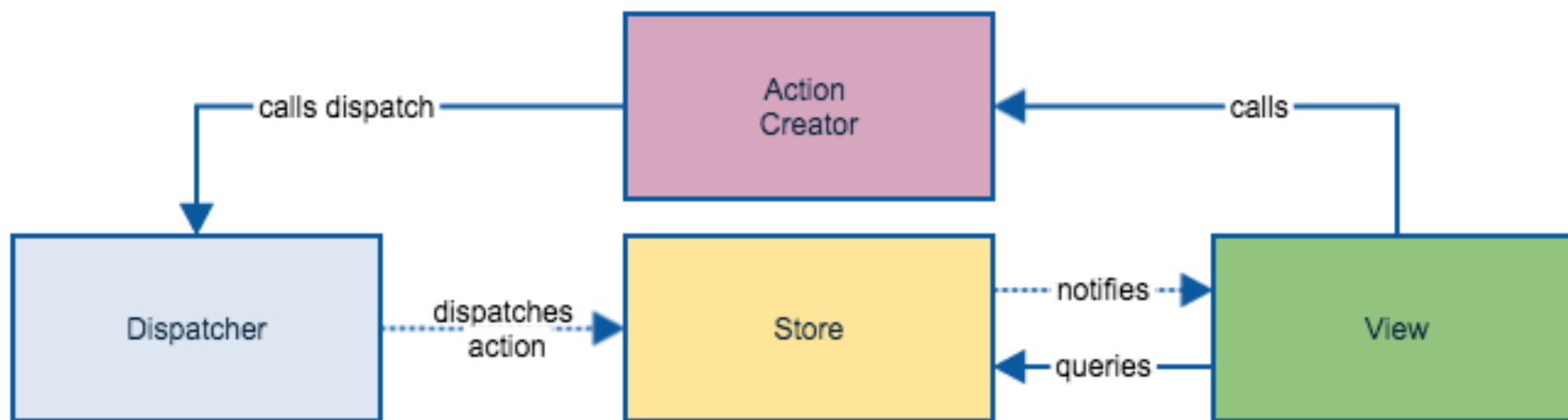


# M - V - V M



- ✓ 开发：Ken Cooper和Ted Peters
- ✓ 发表：John Gossman
- ✓ 又称model-view-binder

# Flux

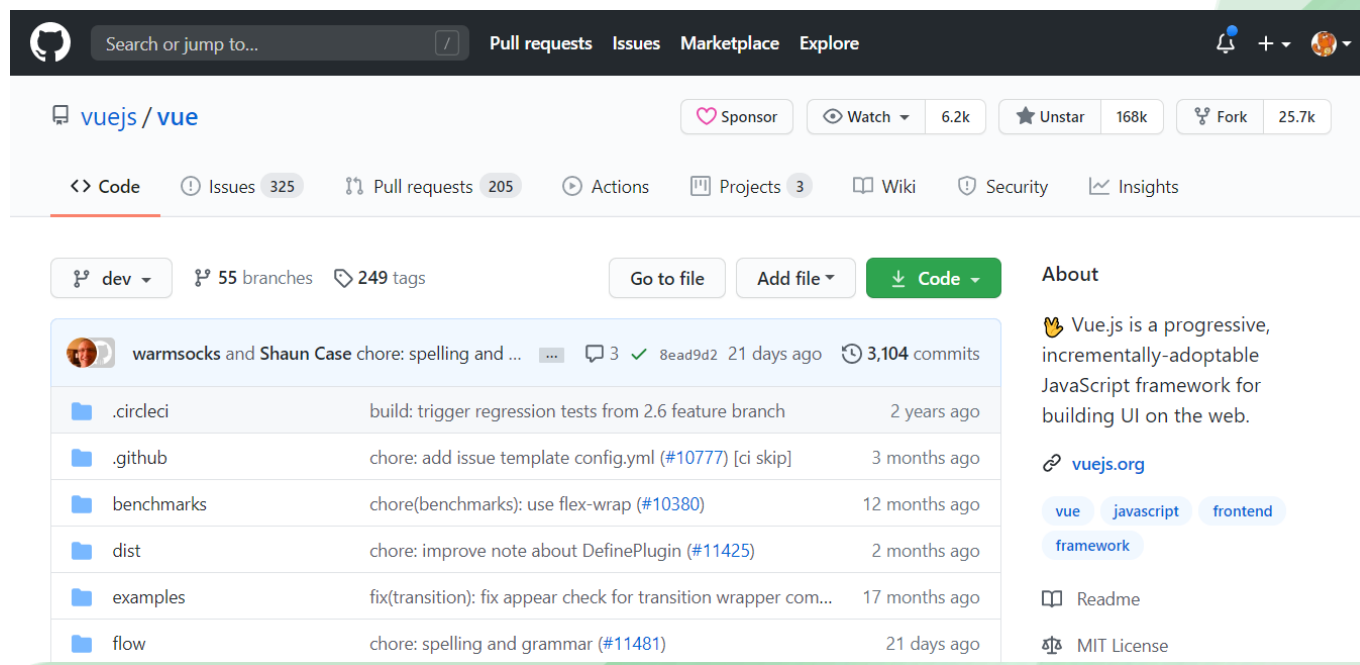


# 01 概览

4/4 Vue的特点



- 渐进式
- 上手容易
- 组件化、靠近标准
- 默认整合最佳实践
- 生态丰富、社区活跃



## 02 基础

# 基础

- 起步
- Vue实例
- 组件



## 02 基础

1/3 起步

	UMD	CommonJS	ES Module (for bundlers)	ES Module (for browsers)
<b>Full</b>	vue.js	vue.common.js	vue.esm.js	vue.esm.browser.js
<b>Runtime-only</b>	vue.runtime.js	vue.runtime.common.js	vue.runtime.esm.js	-
<b>Full (production)</b>	vue.min.js	-	-	vue.esm.browser.min.js
<b>Runtime-only (production)</b>	vue.runtime.min.js	-	-	-



## 02 基础

2/3 Vue实例



# todos

练习做一个Demo

☐ 学习Vue

☐ 学习前端

2 items left

根实例

- └─ TodoList
  - ├─ TodoItem
    - ├─ DeleteTodoButton
    - └─ EditTodoButton
  - └─ TodoListFooter
    - ├─ ClearTodosButton
    - └─ TodoListStatistics

<https://6mr2n.csb.app/>

# 配置

```
let vm = new Vue({  
  // 选项  
})
```

# 配置

```
let vm = new Vue({  
  el: '#app',  
  data: {  
    message: 'Hello 清华前端!'  
  }  
})
```

根元素挂载点

响应数据




# 配置

data: Object | Function

```
data: {  
  message: 'Hello 清华前端!'  
}
```

```
data: function () {  
  return {  
    message: 'Hello 清华前端!'  
  }  
}
```

 不要使用箭头函数

# 配置

props: Array | Object

```
props: ["message", "age"]
```

```
props: {  
  message: {  
    "type" : String,  
    "required": true,  
    "default": "Hello world"  
  },  
  age: {  
    "type" : String,  
    "validator" : val => val>=0  
  }  
}
```

# 配置

computed: { [key: string]: Function | { get: Function, set: Function } }

```
computed: {  
  // 仅读取  
  aDouble: function () {  
    return this.a * 2  
  }  
}
```

```
computed: {  
  // 读取和设置  
  aPlus: {  
    get: function () {  
      return this.a + 1  
    },  
    set: function (v) {  
      this.a = v - 1  
    }  
  }  
}
```



# 配置

methods: { [key: string]: Function }

```
methods: {  
  myFunction: function () {  
    // 自定义方法内容  
  }  
}
```



不要使用箭头函数

✓ 同一个组件或Vue实例: this.myFunction()

## 配置

watch: { [key: string]: string | Function |  
Object | Array }

```
watch: {  
  a: function (val, oldVal) {  
    console.log('new: %s, old: %s', val, oldVal)  
  },  
  // 方法名  
  b: 'someMethod',
```



不要使用箭头函数

## 配置

watch: { [key: string]: string | Function |  
Object | Array }

// 该回调会在任何被侦听的对象的 property 改变时被调用, 不论其被嵌套多深

```
c: {  
  handler: function (val, oldVal) { /* ... */ },  
  deep: true  
},
```

// 该回调将会在侦听开始之后被立即调用

```
d: {  
  handler: 'someMethod',  
  immediate: true  
},
```



不要使用箭头函数



## 配置

`watch: { [key: string]: String | Function | Object | Array }`

```
// 你可以传入回调数组，它们会被逐一调用
e: [
  'handle1',
  function handle2 (val, oldVal) { /* ... */ },
  {
    handler: function handle3 (val, oldVal) { /* ... */ },
    /* ... */
  }
],
// watch vm.e.f's value: {g: 5}
'e.f': function (val, oldVal) { /* ... */ }
```



不要使用箭头函数

# 配置

## el: String | Element

- 提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标
- 可以是 CSS 选择器，也可以是一个 HTMLElement 实例。
- 在实例挂载之后，元素可以用 `vm.$el` 访问。
- 如果在实例化时存在这个选项，实例将立即进入编译过程，否则，需要显式调用 `vm.$mount()` 手动开启编译。

# 配置

template: String | Element

- 模板内容
- # 开头的, 目标Dom的innerHTML



# 生命周期

- 生命周期是程序执行过程中的固定的回调钩子
- 根据约定在约定时机、用约定参数访问同名钩子、有时候还要求特定的返回
- 类似于设计模式的模板方法模式

[Vue的生命周期](#)

[一个例子](#)

# 模版语法

- 基于HTML语法
- 通过虚拟 DOM 渲染函数响应系统减少Dom  
操作次数
- 可以直接写render函数

# 模版语法:插值

- 文本插值
- html插值
- 属性插值
- 使用JavaScript表达式

[演示例子](#)



# 模版语法：指令

- 带有 v- 前缀的特殊 attribute
- 当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM

## 模版语法：指令

- v-show
- v-if
- v-else

[演示例子](#)

# 模版语法：指令

## v-for：循环

```
<ul id="example-1">  
  <li v-for="item in  
items" :key="item.messag  
e">  
    {{ item.message }}  
  </li>  
</ul>
```

```
var example1 = new Vue({  
  el: '#example-1',  
  data: {  
    items: [  
      { message: 'Foo' },  
      { message: 'Bar' }  
    ]  
  }  
})
```



# 模版语法：指令

## v-on：事件

- 支持主流的Dom事件
- 事件修饰符

.stop

.prevent

.capture



.self

.once

.passive

[演示例子](#)

# 模版语法：指令缩写与参数

指令名	缩写方式	例子
v-bind	:	:isBinded='false'  v-bind:isBinded='false'
v-on	@	@click='fun'  v-on:click='fun'

# 模版语法：指令

指令名	作用
v-model	表单双向绑定语法糖
v-pre	原始字符串、不编译
v-clack	模版编译期置标
v-once	只渲染一次



# 模版语法：自定义指令

```
// 注册一个全局自定义指令 `v-focus`  
Vue.directive('focus', {  
  // 当被绑定的元素插入到 DOM 中时……  
  inserted: function (el) {  
    // 聚焦元素  
    el.focus()  
  }  
})
```

```
directives: {  
  focus: {  
    // 指令的定义  
    inserted: function (el) {  
      el.focus()  
    }  
  }  
}
```

➡ **<input v-focus>**

# 过滤器

```
// 注册一个全局自定义过滤器 `capitalize`  
Vue.filter('capitalize', function (value) {  
  if (!value) return ''  
  value = value.toString()  
  return value.charAt(0).toUpperCase() +  
  value.slice(1)  
})
```

```
filters: {  
  capitalize: function (value) {  
    if (!value) return ''  
    value = value.toString()  
    return  
    value.charAt(0).toUpperCase() +  
    value.slice(1)  
  }  
}
```

➡ **{{message | capitalize }}**

➡ **<div :message="message | capitalize"></div>**



## 02 基础

3/3 组件

# 组件基础

- Vue.component 方法：定义并注册到全局

```
Vue.component('my-component-name', { /* ... */ })
```



# 组件基础

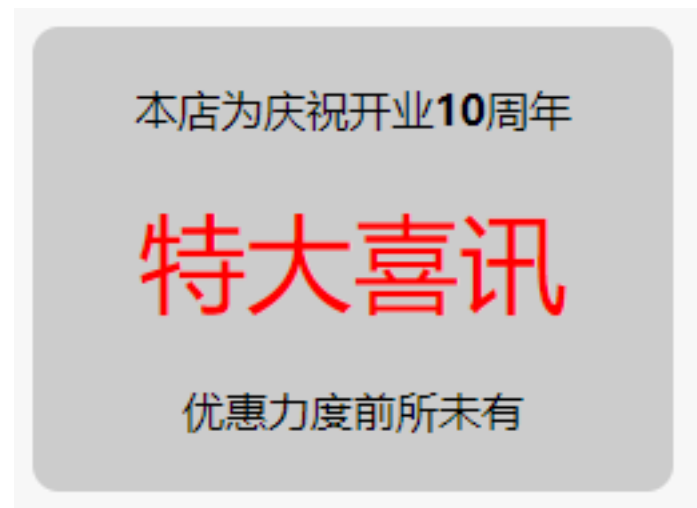
- data必须是函数，并且建议不使用箭头函数
- template使用单一的根元素

```
<h3>{{ title }}</h3>  
<div v-html="content"></div>
```

```
<div>  
  <h3>{{ title }}</h3>  
  <div v-html="content"></div>  
</div>
```

# 组件基础：插槽

```
Vue.component('alert-box', {  
  template: `  
    <div class="demo-alert-box">  
      <strong>特大喜讯</strong>  
      <slot></slot>  
    </div>  
  `,  
})
```

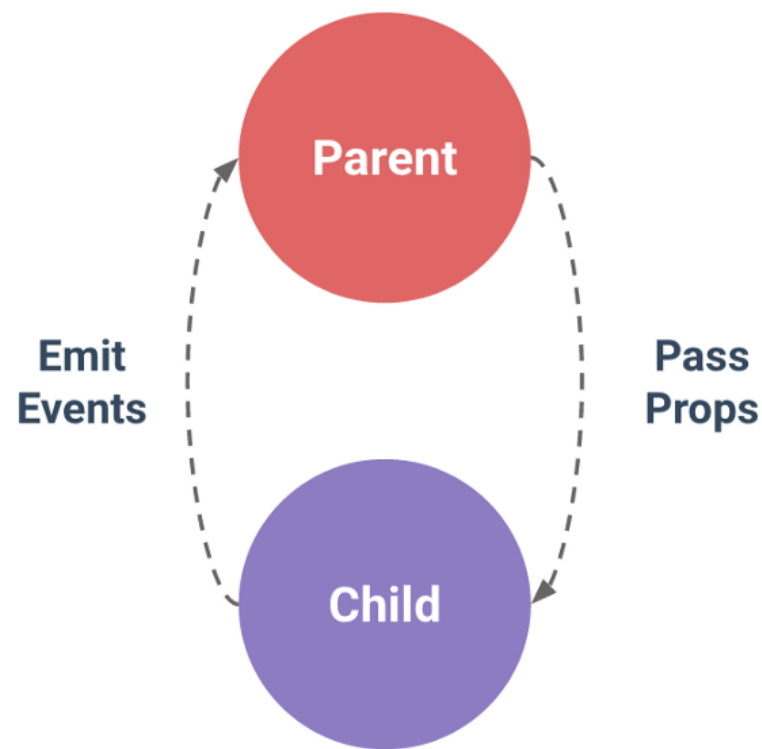


`<alert-box>推广活动</alert-box>`

[插槽例子](#)

# 父子组件

- 父组件向子组件props传值
- 子组件向父组件发送事件
- 中央事件总线



[演示例子](#)

[中央事件总线](#)

# 官方组件

组件	作用
transition	动画、过渡
keep-alive	缓存不活动的组件



# 渲染函数

- 大多数情况推荐使用模板方式
- 有些情况指定渲染函数可以简化代码
- 模板其实最终也会被编辑成渲染函数

[未使用渲染函数的例子](#)

[使用渲染函数的例子](#)

## 03 生态

# 生态

- devtools
- 工具链
- 路由
- 状态管理
- 组件库

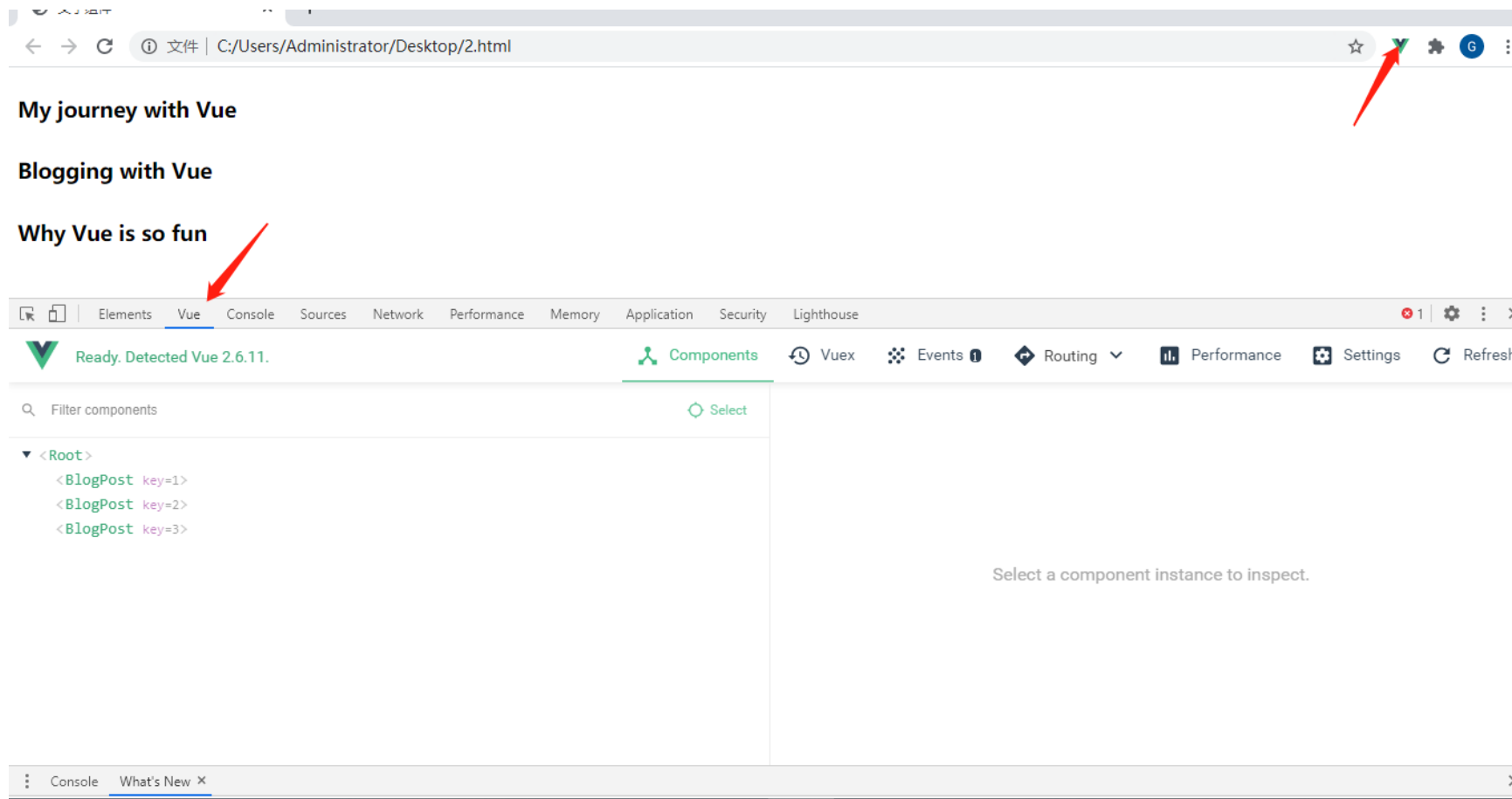


## 03 生态

1/5 Devtools

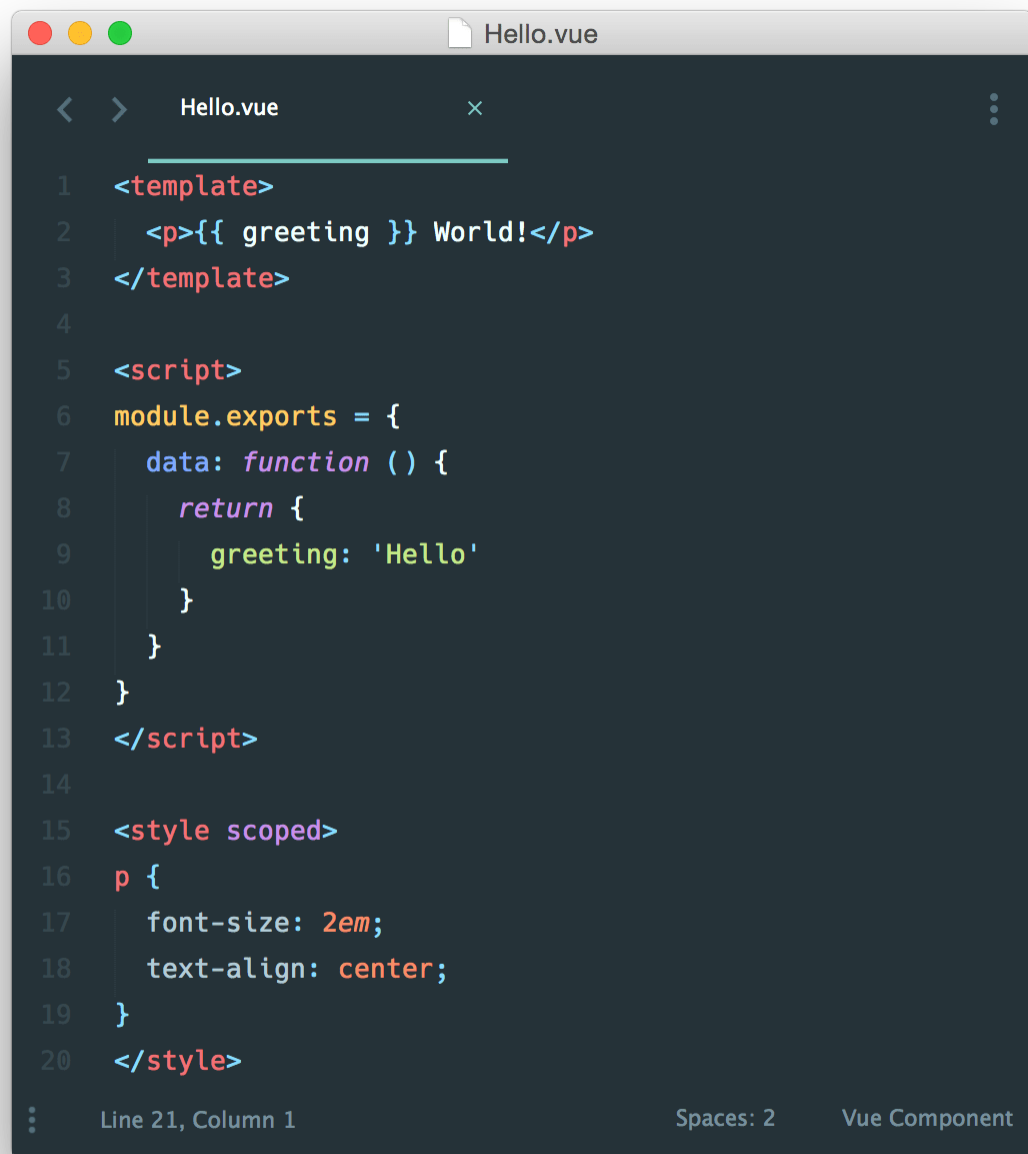


# Devtools



## 03 生态

2/5 工具链



```
1 <template>
2   <p>{{ greeting }} World!</p>
3 </template>
4
5 <script>
6   module.exports = {
7     data: function () {
8       return {
9         greeting: 'Hello'
10      }
11    }
12  }
13 </script>
14
15 <style scoped>
16   p {
17     font-size: 2em;
18     text-align: center;
19   }
20 </style>
```

Line 21, Column 1      Spaces: 2      Vue Component



# 工具链

- devtools
- 初始化：vue-cli
- babel-preset：@vue/app
- webpack: vue-loader
- 脚手架：vue-cli-service
- vscode高亮：Vetur



## 03 生态

3/5 路由

# 路由：vue-router

- 以组件形式提供：router-link router-view
- abstract模式、hash模式、history模式

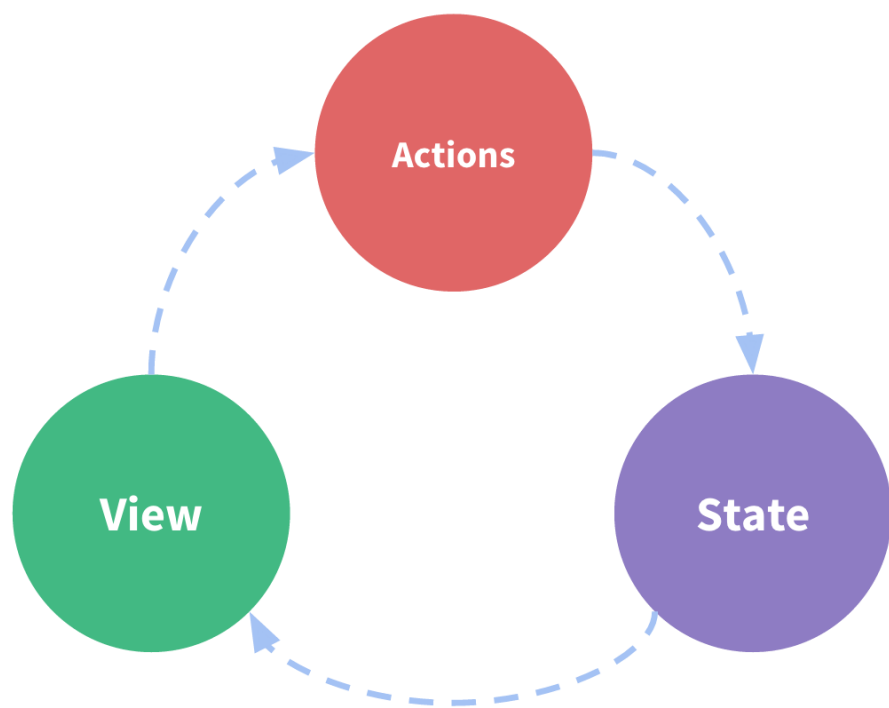
[演示例子](#)



## 03 生态

4/5 状态管理

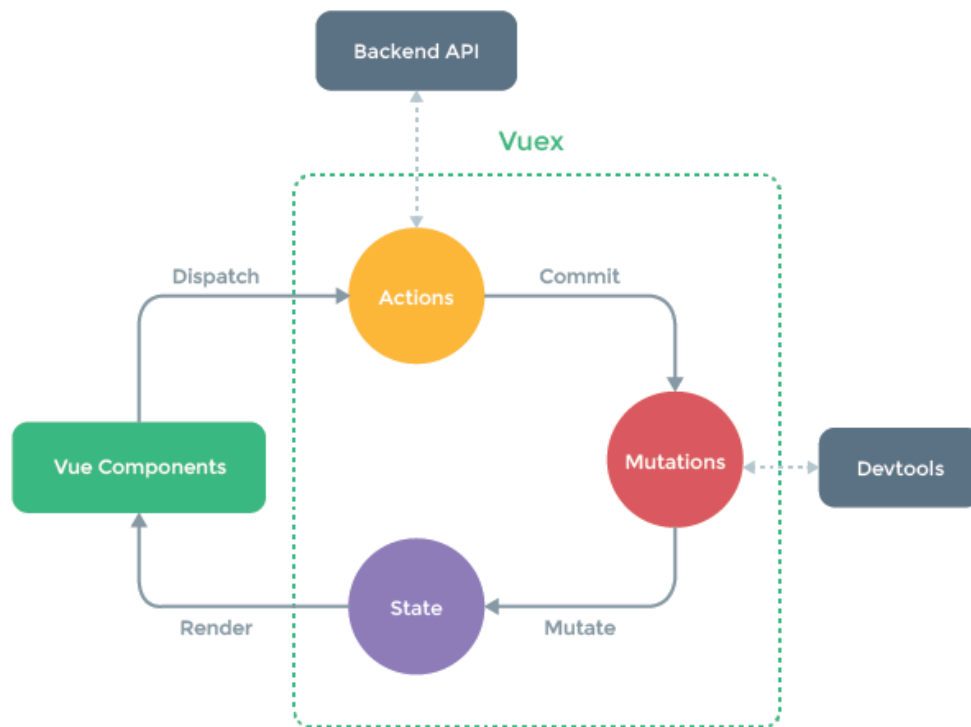
# 状态管理



- 多个视图依赖于同一状态。
- 来自不同视图的行为需要变更同一状态。



# 状态管理



## 03 生态

5/5 组件库

# 组件库

- [Element UI](#)
- [AntDesign of vue](#)
- [clair](#)



## 04 进阶



# 进阶

- Vue3.0最新进展
- Vue 与多端

## 04 进阶

1/2 Vue3.0最新进展

# Vue 3.0 最新进展

- Composition API
- Fragments
- Suspense组件
- 更好的TypeScript支持
- Vite

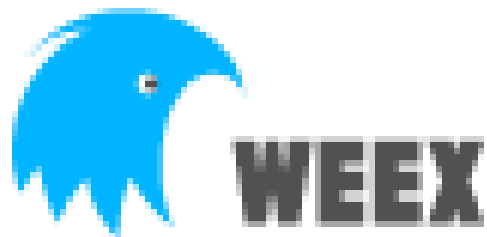


## 04 进阶

2/2 Vue与多端



# Vue 与 多端



## 05 参考

## 参考

- [Vue官网](#)
- [Vue3官网](#)
- [Vue-element-admin](#)
- [Awesome Vue](#)





谢谢观看！

