Vue

- Vue简介
- Vue特点
- Vue数据绑定
- Vue Virtual DOM
- Vue组件
- Vue Hello World
- Vue模板语法
- Vue事件处理
- 学习网址

Vue简介

vue是一套构建用户界面的渐进式框架。

只关注视图层，采用自底向上增量开发的设计。通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。它使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。所有 Vue.js 的模板都是合法的 HTML ，所以能被遵循规范的浏览器和 HTML 解析器解析。在底层的实现上，Vue 将模板编译成虚拟 DOM 渲染函数。结合响应系统，Vue 能够智能地计算出最少需要重新渲染多少组件，并把 DOM 操作次数减到最少。

Vue特点

1）轻量级的框架

Vue.js 能够自动追踪依赖的模板表达式和计算属性，提供 MVVM 数据绑定和一个可组合的组件系统，具有简单、灵活的 API，使读者更加容易理解，能够更快上手。

2）双向数据绑定

声明式渲染是数据双向绑定的主要体现，同样也是 Vue.js 的核心，它允许采用简洁的模板语法将数据声明式渲染整合进 DOM。

3）指令

Vue.js 与页面进行交互，主要就是通过内置指令来完成的，指令的作用是当其表达式的值改变时相应地将某些行为应用到 DOM 上。

Vue特点

4）组件化

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。

在 Vue 中，父子组件通过 props 传递通信，从父向子单向传递。子组件与父组件通信，通过触发事件通知父组件改变数据。这样就形成了一个基本的父子通信模式。

在开发中组件和 HTML、JavaScript 等有非常紧密的关系时，可以根据实际的需要自定义组件，使开发变得更加便利，可大量减少代码编写量。

组件还支持热重载（hotreload）。当我们做了修改时，不会刷新页面，只是对组件本身进行立刻重载，不会影响整个应用当前的状态。CSS 也支持热重载。

Vue特点

5）客户端路由

Vue-router 是 Vue.js 官方的路由插件，与 Vue.js 深度集成，用于构建单页面应用。Vue 单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来，传统的页面是通过超链接实现页面的切换和跳转的。

6）状态管理

状态管理实际就是一个单向的数据流，State 驱动 View 的渲染，而用户对 View 进行操作产生 Action，使 State 产生变化，从而使 View 重新渲染，形成一个单独的组件。

Vue特点

Vue优势：

1. 轻量化
2. 低复杂性
3. Virtual DOM
4. 低学习曲线

缺点：

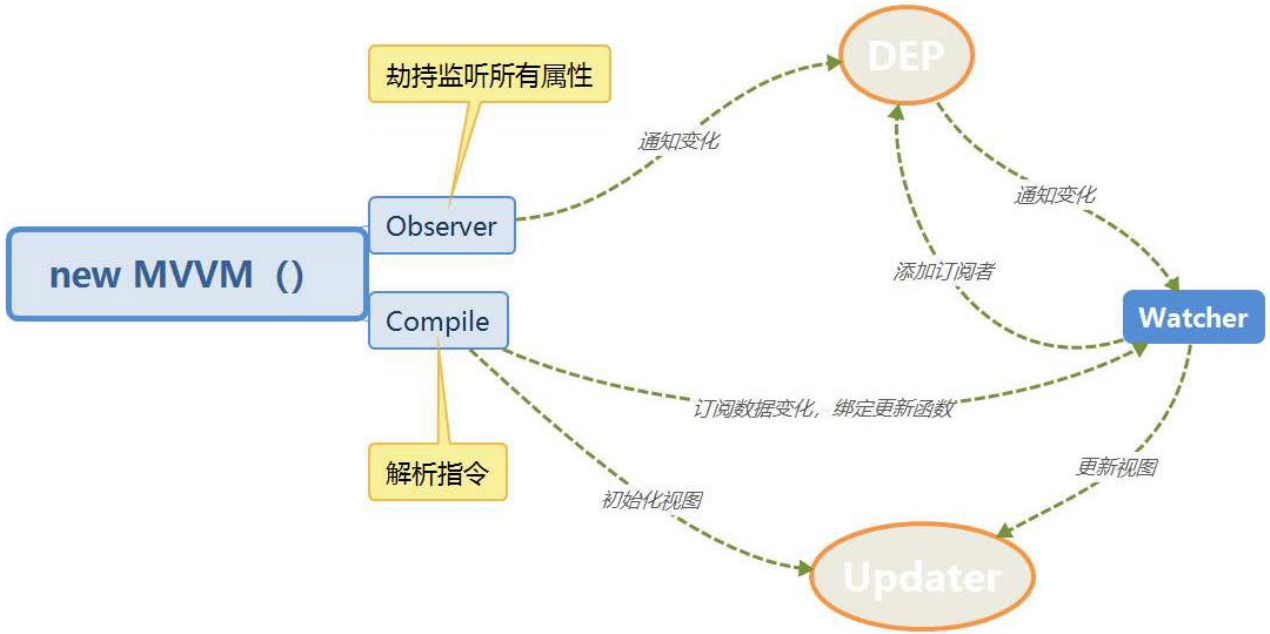
1. 相比React、Angular发展时间短，社区规模小
2. 目前使用Vue公司的不多

Name	Size
Ember 2.2.0	435K
Ember 1.13.8	486K
Angular 2	566K
Angular 2 + Rx	**766K**
Angular 1.4.5	143K
Vue 2.4.2	58.8K
Inferno 1.2.2	48K
Preact 7.2.0	**16K**
React 0.14.5 + React DOM	**133K**
React 0.14.5 + React DOM + Redux	139K
React 16.2.0 + React DOM	97.5K

不同前端框架/库的下载空间

Vue数据绑定

目前主流的数据绑定方式有三种，分别是发布者-订阅者模式、脏值检查、数据劫持。vue.js 则是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。



Vue数据绑定

Vue数据绑定

vue实现双向数据绑定，它会自动响应数据的变化情况，并且根据用户在代码中预先写好的绑定关系，对所有绑定在一起的数据和视图内容都进行修改。这需要实现三个模块：Observer、Compile与Watcher。

Observer：能够对数据对象的所有属性进行监听，如有变动可拿到最新值并通知订阅者。

Compile：对每个元素节点的指令进行扫描和解析，根据指令模板替换数据，以及绑定相应的更新函数。

Watcher：作为连接Observer和Compile的桥梁，能够订阅并收到每个属性变动的通知，执行指令绑定的相应回调函数，从而更新视图。

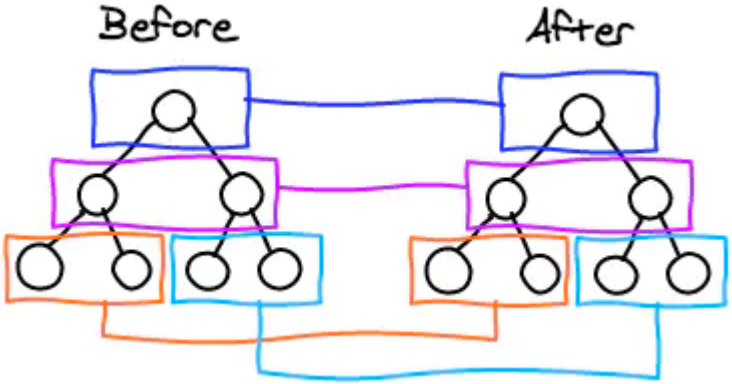
Vue数据绑定

具体来说vue会遍历此data中对象所有的属性，并使用Object.defineProperty把这些属性全部转为getter/setter，而每个组件实例都有watcher对象，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 setter被调用时，会通知watcher重新计算，从而致使它关联的组件得以更新。

Vue Virtual DOM

Vue和React类似，也是使用了Virtual DOM

为了实现高效的DOM操作，一套高效的虚拟DOM diff算法显得很有必要。我们通过patch 的核心----diff 算法，找出本次DOM需要更新的节点来更新，其他的不更新。



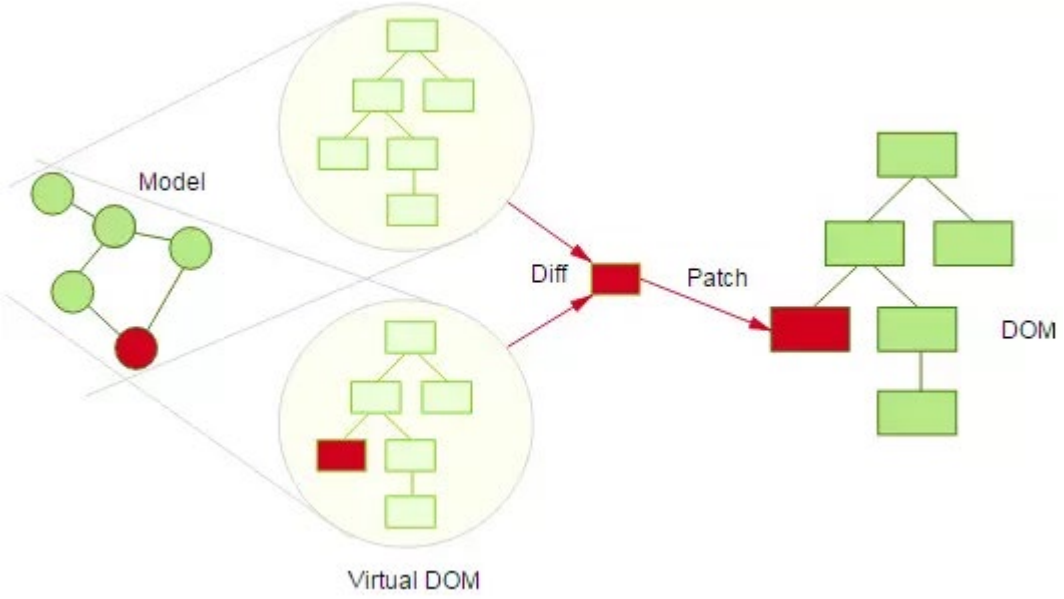
diff算法

仅在同级的vnode间做diff，递归地进行同级vnode的diff，最终实现整个DOM树的更新。因为跨层级的操作是非常少的，忽略不计，这样时间复杂度就是O(n)。

Vue Virtual DOM

diff 算法包括几个步骤：

1.用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中。

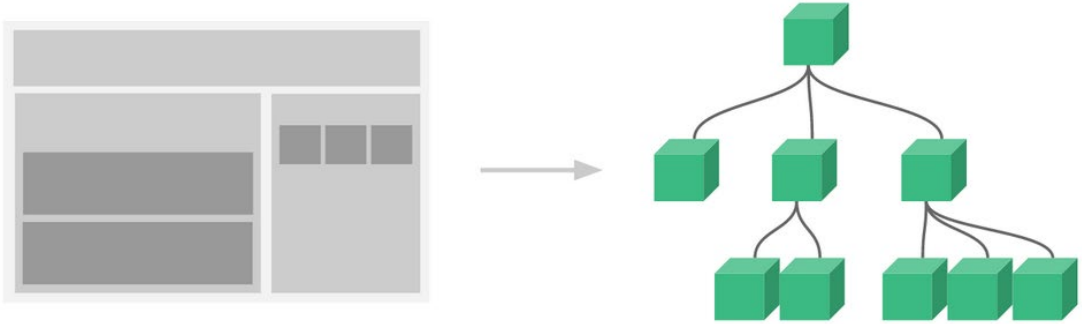


2.当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异。

3.把所记录的差异应用到所构建的真正的DOM树上，视图就更新了。

Vue组件

组件（Component）是 Vue.js 最强大的功能之一。组件可以扩展 HTML 元素，封装可重用的代码。组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树，如图所示，例如，可能会有页头、侧边栏、内容区等组件，每个组件又包含了其它的像导航链接、博文之类的组件。



组件树

Vue Hello World

一个简单的Hello World示例

```
<div id="app">
    {{ message }}
</div>
</body>
<script src="js/vue.js"></script>
<script>
    var exampleData = {
        message: 'Hello World!'
    }

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

Vue模板语法

Vue.js 使用了基于 HTML 的模版语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。

Vue.js 的核心是一个允许你采用简洁的模板语法来声明式的将数据渲染进 DOM 的系统。

结合响应系统，在应用状态改变时， Vue 能够智能地计算出重新渲染组件的最小代价并应用到 DOM 操作上。

Vue模板语法

文本：使用双大括号：

```
<div id="app">
    {{ message }}
</div>
```

Html：使用 v-html 指令用于输出 html 代码：

```
<div id="app">
    <div v-html="message"></div>
</div>

<script>
    new Vue({
        el: '#app',
        data: {
            message: '<h1>hello world</h1>'
        }
    })
</script>
```

Vue模板语法

属性：HTML 属性中的值应使用 v-bind 指令。

以下实例判断 use 的值，如果为 true 使用 class1 类的样式，否则不使用该类：

```
<div id="app">
    <label for="r1">修改颜色</label><input type="checkbox" v-model="use" id="r1">
    <br><br>
    <div v-bind:class="{'class1': use}">
        v-bind:class 指令
    </div>
</div>

<script>
    new Vue({
        el: '#app',
        data:{
            use: false
        }
    });
</script>
```

Vue模板语法

条件判断：条件判断使用 v-if 、v-else、v-else-if指令：

```
<div id="app">
    <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>
</div>
<script>
    new Vue({
        el: '#app',
        data: {
            type: 'C'
        }
    })
</script>
```

判断type变量值

Vue事件处理

可以用 v-on 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码。使用 v-on 有几个好处：

能轻松定位在 JavaScript 代码里对应的方法。因为无须在 JavaScript 里手动绑定事件，你的 ViewModel 代码可以是非常纯粹的逻辑，和 DOM 完全解耦，更易于测试。

```
<div id="app">
    <!-- `greet` 是在下面定义的方法名 -->
    <button v-on:click="greet">Greet</button>
</div>

<script>
    var app = new Vue({
        el: '#app',
        data: {
            name: 'Vue.js'
        },
        // 在 `methods` 对象中定义方法
        methods: {
            greet: function (event) {
                // `this` 在方法里指当前 Vue 实例
                alert('Hello ' + this.name + '!')
                // `event` 是原生 DOM 事件
                if (event) {
                    alert(event.target.tagName)
                }
            }
        }
    })
    // 也可以用 JavaScript 直接调用方法
    app.greet() // -> 'Hello Vue.js!'
</script>
```

学习网址

- https://www.runoob.com/vue2/vue-tutorial.html
- https://www.vue-js.com/
- https://vuejs.bootcss.com/guide/