Vue.js 入门— 轻量级的JavaScript MVVM库

@坤坤2012

调查

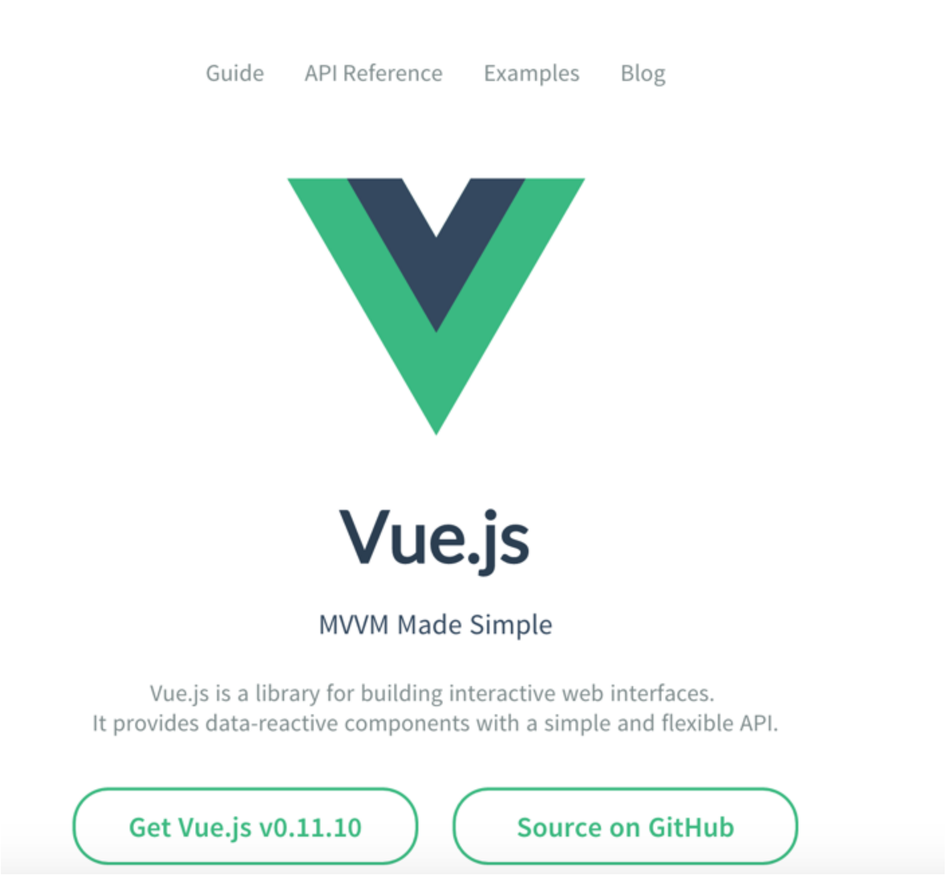
大家都用哪些JS框架？ dojo / angular / react / backbone

Vue.js 概况

作者 尤小右（尤雨溪）meteor

- 2013 年底作为个人实验项目开始开发
- 2014 年 2 月公开发布
- 2015年 5 月发布发布的 0.11.10
- 截止 2015 年 5 月15 日 :4379+ Stars on GitHub 12contributors

vuejs.org



cn.vuejs.org

获取

Standalone

Simply download and include with a script tag. `Vue` will be registered as a global variable.

Development Version 194.12kb, plenty of comments and debug/warning messages.

Production Version 59.72kb minified / 19.53kb gzipped

Also available on cdnjs (takes some time to sync so the latest version might not be available yet).

NPM

Shell

```
$ npm install vue
# for edge version:
$ npm install yyx990803/vue#dev
```

Bower

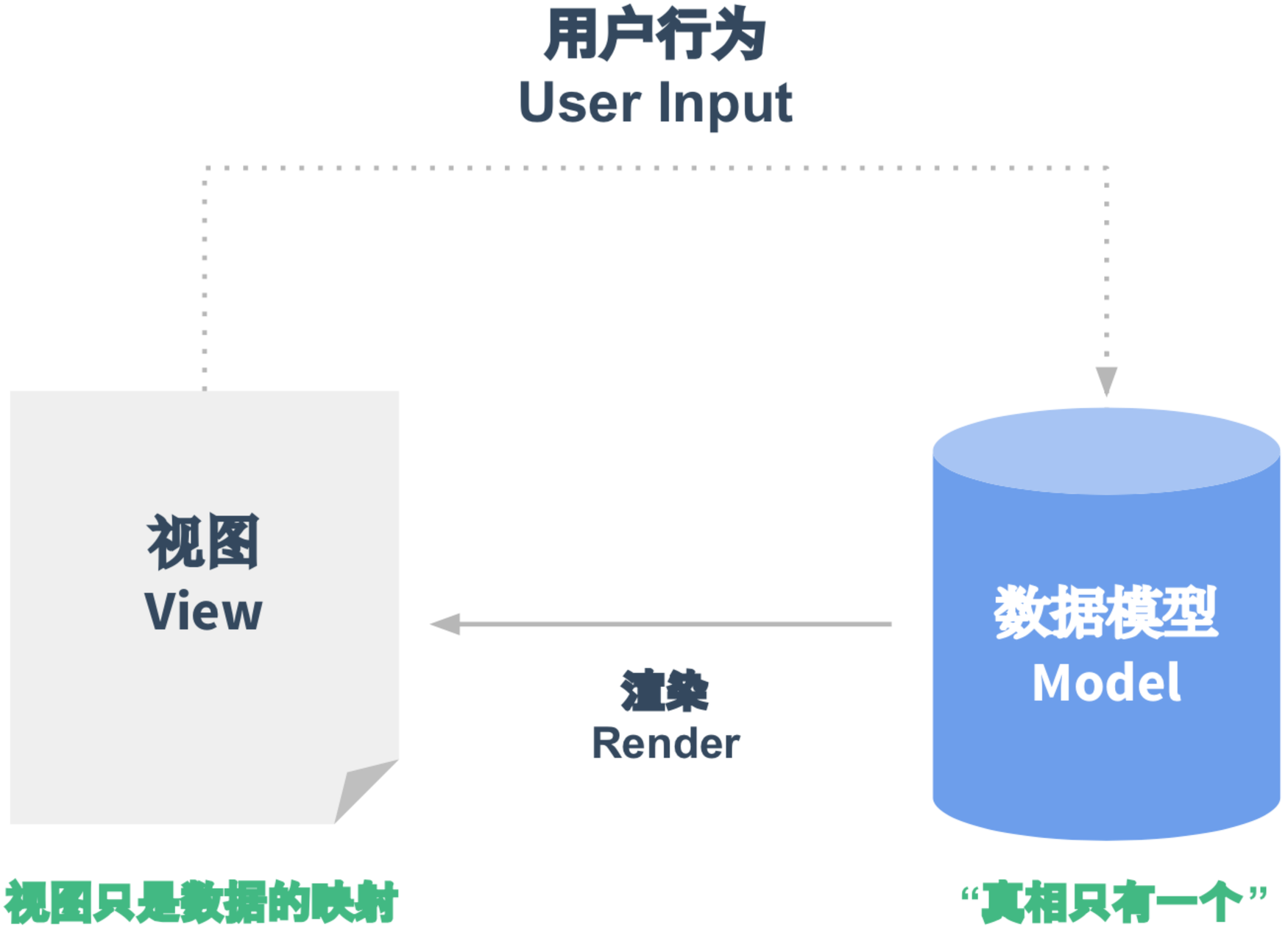
Shell

```
# only stable version is available through Bower
$ bower install vue
```

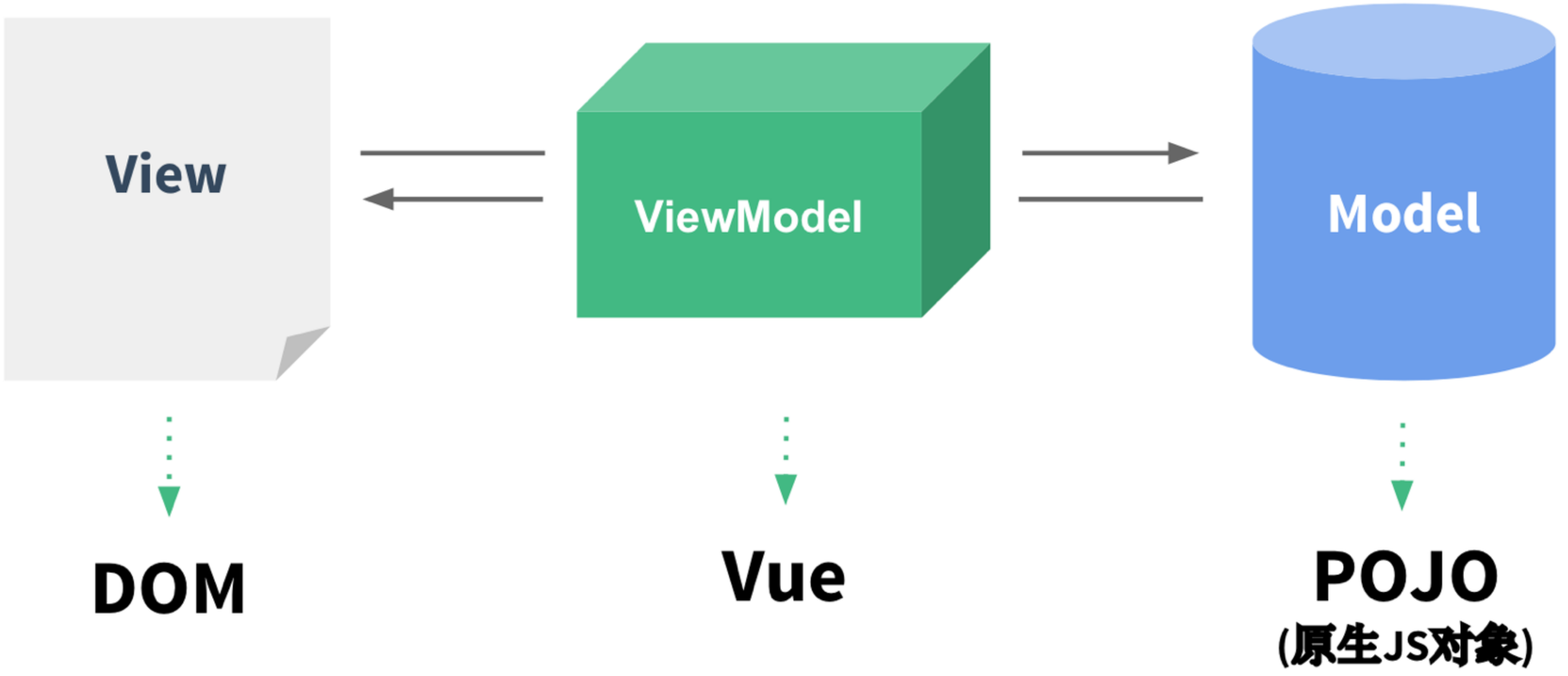
helloword

核心思想

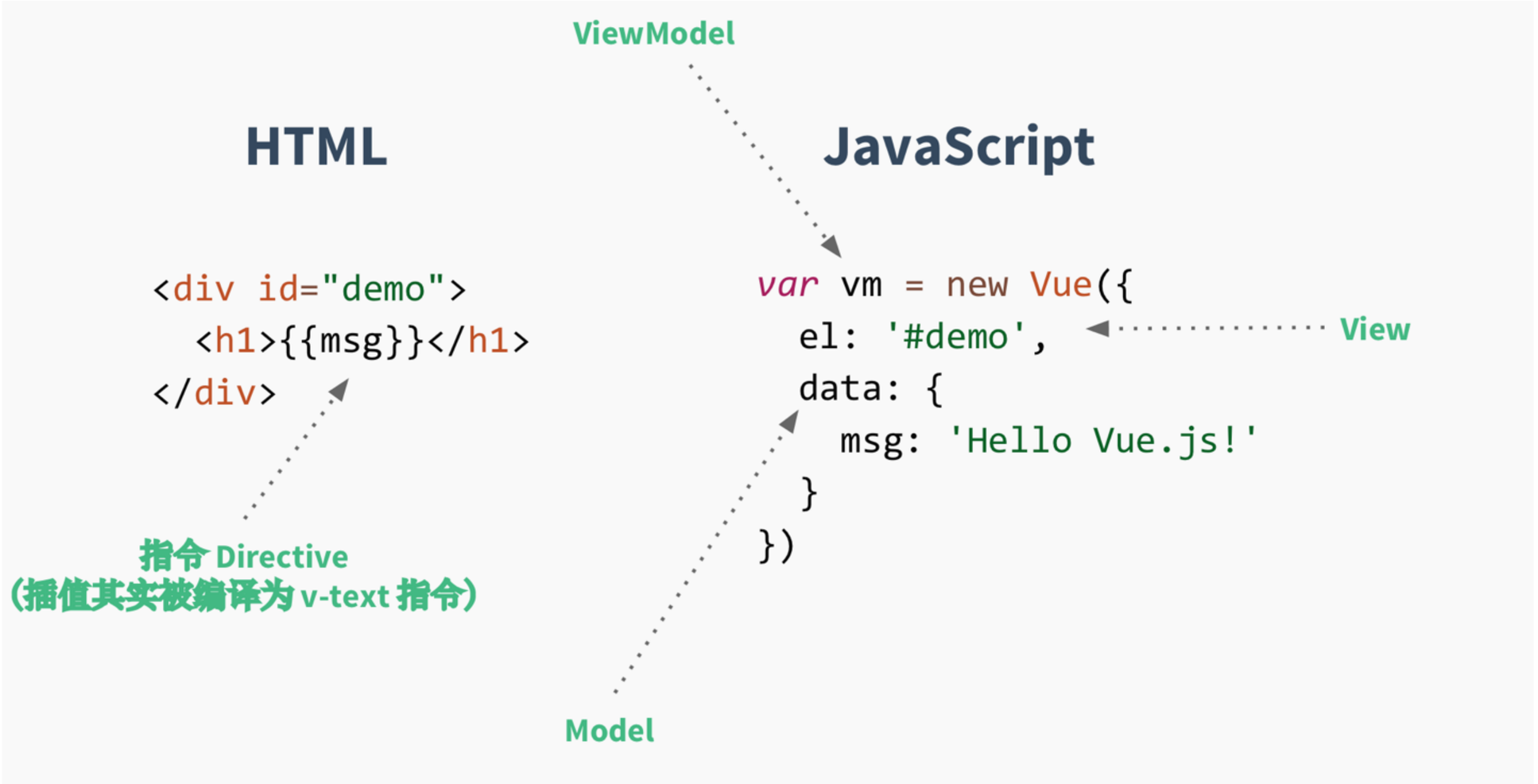
1. 数据驱动UI
2. 组件化


用户行为
User Input
视图
View
渲染
Render
数据模型
Model
视图只是数据的映射
“真相只有一个”

通过 MVVM 的数据绑定实现自动同步



Vue是一个创建VM（ViewModel）的构造函数，用于连接View和model



view：vm.$el

model: vm.$data

可以使用vm.$data.msg也可以直接vm.msg来获取数据

1.指令
2.过滤器
3.组件系统

指令

一个特殊的 HTML 属性 (attribute)

```
8       <body>
9         <div id="demo">
10          <p>{{message}}</p>
11          <input v-model="message">
12        </div>
13
14
15        <script type="text/javascript">
16            var data = {
17                message: 'Hello Vue.js!'
18              }
19            var demo = new Vue({
20                el: '#demo',
21                data: data
22            })
23        </script>
```

Vue内部自带指令

v-text 更新元素的textContent
v-html 更新元素的innerHTML
v-show 根据绑定的真实值将元素的 display 属性设置为none或它的原始值
v-class 如果没有提供参数名，将会直接将绑定值加入到元素的 classList 中，
当绑定值改变时更新元素的 class.
v-attr 更新元素的指定属性（更新的属性由参数指定）
v-style 将CSS属性以内联的形式应用到元素上
v-on 为元素添加一个事件监听器
v-model 创建双向绑定
v-if 以绑定值为基础条件插入或删除元素
v-pre 跳过编译此元素和此元素所有的子元素
v-cloak 直到关联的 ViewModel 结束编译之前本属性都会留在元素上
v-el 在一个DOM元素上注册一个更容易被自身 Vue 实例访问的引用。
如，<div v-el=“hi">可以使用vm.$.hi访问到。

指令Demo

v-repeat

对于数组中的每个对象，该指令将创建一个以该对象作为其 $data 对象的子Vue实例。
这些子实例继承父实例的所有数据，因此在重复的模板元素中你既可以访问子实例的属性，
也可以访问父实例的属性。
此外,你还可以访问 $index 属性，其表示所呈现的实例在对象数组中对应的索引。

HTML

```html
<ul id="demo">
  <li v-repeat="items" class="item-{{$index}}">
    {{$index}} - {{parentMsg}} {{childMsg}}
  </li>
</ul>
```

JS

```js
var demo = new Vue({
  el: '#demo',
  data: {
    parentMsg: 'Hello',
    items: [
      { childMsg: 'Foo' },
      { childMsg: 'Bar' }
    ]
  }
})
```

demo1

demo2

Vue内部自定义指令

Vue.js允许你注册自定义指令，实质上是让你教Vue一些新技巧：怎样将数据的变化映射成DOM的行为。
你可以使用Vue.directive(id, definition)的方法传入指令名称和定义对象来注册一个全局自定义指令。
定义对象要能提供一些钩子函数（全都可选）：

JS

```
Vue.directive('my-directive', {
  bind: function () {
    // do preparation work
    // e.g. add event listeners or expensive stuff
    // that needs to be run only once
  },
  update: function (newValue, oldValue) {
    // do something based on the updated value
    // this will also be called for the initial value
  },
  unbind: function () {
    // do clean up work
    // e.g. remove event listeners added in bind()
  }
})
```

指令对象有一些有用的公开属性

el: 指令绑定的元素
vm: 拥有该指令的上下文视图模型
expression: 绑定的表达式，不包括参数和过滤器
arg: 当前参数
raw: 未被解析的原始表达式
name: 不带前缀的指令名

DEMO

动画

在节点插入/移除（比如使用v-if),
或者隐藏/显示（比如v-show)
DOM的过程中实现过渡效果，使用v-transition 指令。

1.CSS 过渡

2.Javascript过渡

过渡效果详情>>

CSS实现过渡效果

对于使用了v-transition指令的元素（比如v-transition=“test”），Vue将会自动完成

1.在元素显示之前时候添加test-enter的class样式类，接着添加 / 显示DOM到浏览器，然后移除该类，开始动画过渡

2.在元素隐藏之前时候添加test-leave样式类，动画结束，从浏览器中移除/隐藏M

对于开发者来说除了在元素上使用v-transition=“test”

只需要在定义test-enter、test-leave这两个css就行了。

Demo

Javascript 实现过渡效果

```
<p v-transition="fade"></p>
```

```
Vue.transition('fade', {
      enter: function (el, done) {
        //借助jquery实现过渡效果 当显示的时候调用
        $(el)
          .css('opacity', 0)
          .animate({ opacity: 1 }, 1000, done)
        return function () {
          $(el).stop()
        }
    },
    leave: function (el, done) {
      //隐藏的时候调用
      $(el).animate({ opacity: 0 }, 1000, done)
      return function () {
        $(el).stop()
      }
    }
});
```

过滤器

简单来说，就是一个函数，获得一个值，进行一些处理，
然后返回处理后的值。当一个值从一个个过滤器中传递过去以后，
最后得到我们想要的值，就像光线穿过相机镜头的一层层滤镜一样。

过滤器

capitalize ‘abc’ => ‘Abc’
uppercase ‘abc’ => ‘ABC’
lowercase ‘ABC’ => ‘abc’
currency “12345 => $12,345.00”
json
filterBy
orderBy

过滤器

key

1.enter
2.tab
3.delete
4.esc
5.up
6.down
7.left
8.right

包裹 handler 使得其仅在 keyCode 是指定的参数时才被调用。
你也可以使用几个常用键值的别名：

```
<input v-on="keyup:doSomething | key enter">
</div>


<script type="text/javascript">
      vm = new Vue({
        el: '#demo',
        data:{
          message:"Hello World"
        },
        methods:{
          doSomething:function(){
            alert('doSomething');
          }
        }
    })
</script>
```

Demo

自定义过滤器Vue.filter

```
Vue.filter('reverse', function (value) {
    return value.split('').reverse().join('')
})
```

Demo

侦听事件（v-on）

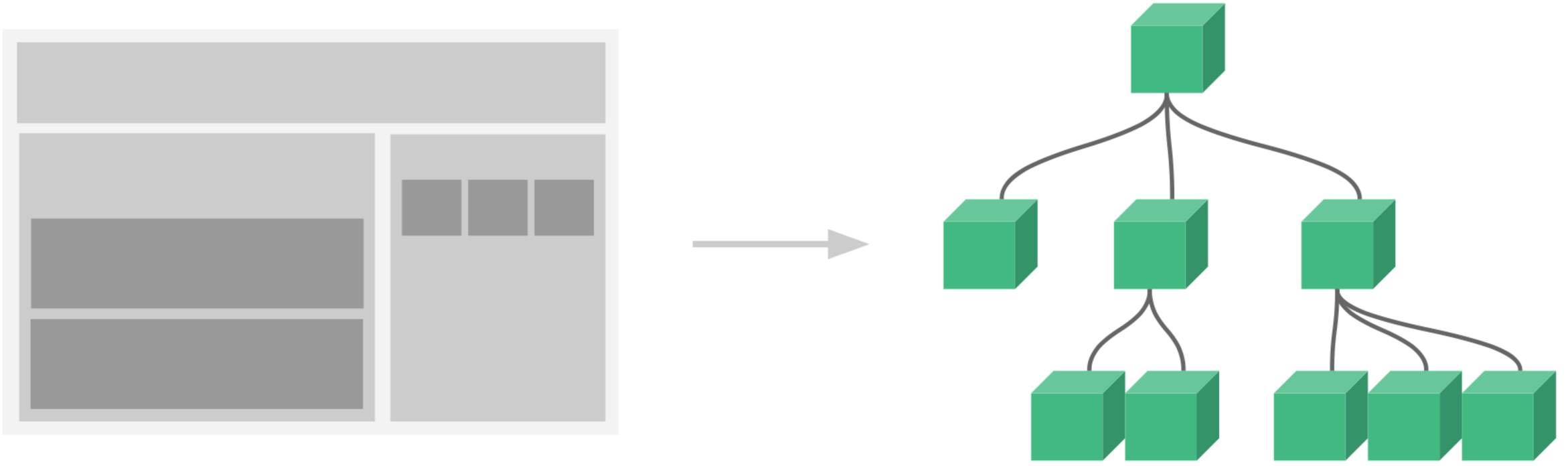
```
    <div id="demo">
      <button v-on="click: onClick">click me</button>
    </div>
new Vue({
      el: '#demo',
      data: {
          n: 1
      },
      methods: {
          onClick: function (e) {
              // e是原生的DOM事件对象
              // this 指向该ViewModel实例
              this.n++
          }
      }
    })
```

```
<button v-on="click: n++"></button>
```

组件化

每一个应用界面都可以 看作是组件构成的

可以把界面抽象为 ViewModel Tree



在vueJS中注册组件

```
// 扩展 Vue 来自定义一个可复用的组件类
var MyComponent = Vue.extend({
  template: '<p>{{msg}}</p>',
  paramAttributes: ['msg']
})
// 全局注册该组件
 Vue.component('my-component', MyComponent)
```

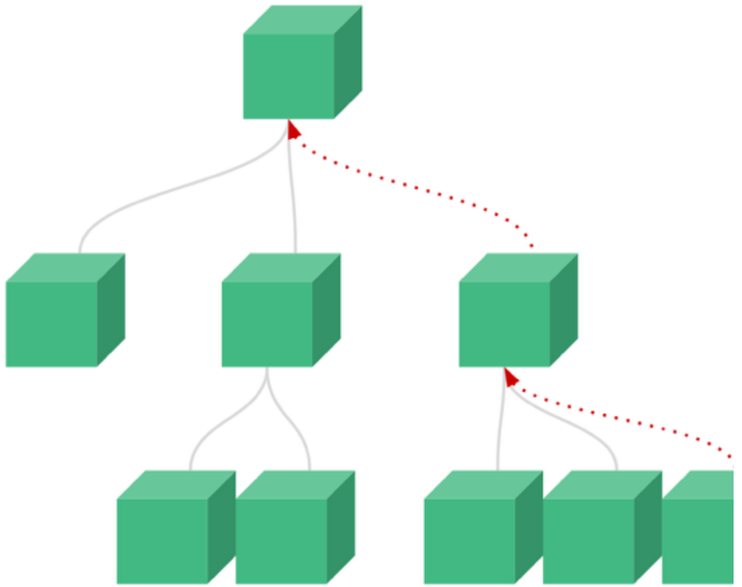
```
<my-component msg="Hello!"></my-component>
```

my-component 组件的模板将会被填充到该元素中，而 msg 则会被作为数据传入该组件实例。渲染结果如下。

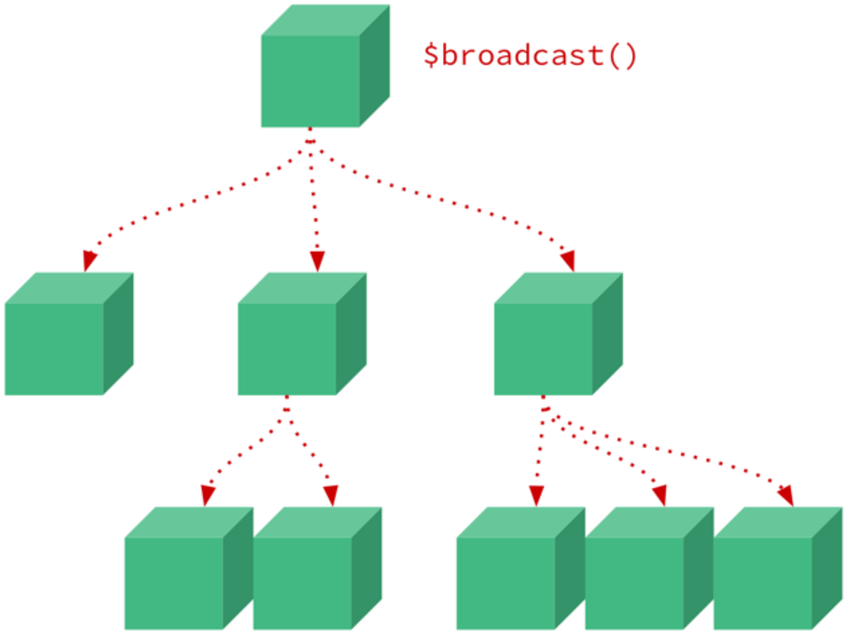
```
<my-component>
  <p>Hello!</p>
</my-component>
```

组件间的事件通信

vm.$dispatch



vm.$broadcast



组件生命周期

created

- 类型： `Function`

在实例被创建的时候同步调用。在这个阶段，实例完成了包含以下内容的预处理：数据健康，数据监控，计算属性，方法，监控事件回调。但DOM编译还没开始， `$el` 还不可用。

beforeCompile

- 类型： `Function`

在编译之前调用。

compiled

- 类型： `Function`

编译完成后调用，在这个阶段，所有的指令都绑定，数据变化会触发DOM更新。但不能保证 `$el` 已经被插入到DOM中。

ready

- 类型： `Function`

当完成编译**而且** `$el` 也第一次的插入到DOM中了之后调用。注意这个插入必须要通过Vue完成的(例如 `vm.$appendTo()` 的方法或者是一个指令更新的结果)来触发的 `ready` 事件。

组件生命周期

attached

- 类型: `Function`

当 `vm.$el` 被一个指令或是VM实例方法（例如 `$appendTo()` ）添加到DOM里的时候调用。直接操作 `vm.$el` **不会**触发这个事件。

detached

- 类型: `Function`

当 `vm.$el` 被一个指令或是VM实例方法从DOM里删除的时候调用。直接操作 `vm.$el` **不会**触发这个事件。

beforeDestroy

- 类型: `Function`

在一个Vue实例被销毁之前调用。这个时候，实例的绑定和指令仍工作正常。

destroyed

- 类型: `Function`

在一个Vue实例被销毁之后调用。如果被执行，所有的Vue实例的绑定和指令都会被解除绑定，所有子组件也会被销毁.

DEMO

优势和使用场景

1. **以原生的JS对象作为 Model， 使 Vue 对于 数据持久层的接口非常灵活**
2. **侵入性低，方便与其他框架结合使用**
3. **组件化**
4. **生态：插件，Vue + Browserify Vue + Webpack**
5. **轻量+高性能**

简单

写点 HTML，拿来 JSON，创建一个 Vue 示例，就这么简单。

快速

以精确有效的异步批处理方式更新 DOM。

组合

用解耦的、可复用的组件组合你的应用程序。

紧凑

~18kb min+gzip，且无依赖。

强大

表达式 & 无需声明依赖的可推导属性 (computed properties)。

对模块友好

可以通过 NPM、Bower 或 Duo 安装——无缝融入你的生命！

学习资料

Vue.js 中文入门

官方文档

谢谢！