Vue.js 生命周期

主讲人： 和凌志

主要内容

- 计算属性 （computed）
- 全局组件
- 局部组件

计算属性 computed

Vue 计算属性 computed

- 模板内的表达式非常便利，设计它们的初衷是用于简单运算。在模板中放入太多的逻辑会让模板过重且难以维护。 比如:

```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

- 在这个地方，模板不再是简单的声明式逻辑。必须看一段时间才能意识到，这里是想要显示变量 message 的翻转字符串。当想要在模板中多次引用此处的翻转字符串时，就会更加难以处理。
- 所以，对于任何复杂逻辑，都应当使用计算属性 （computed） 。

计算属性 computed

```
<div id="example">
    <p>Original message: "{{ message }}"</p>
    <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>

<script>
    var vm = new Vue({
        el: '#example',
        data: {
            message: 'Hello'
        },
        computed: {
            // 计算属性的 getter
            reversedMessage: function () {
                // `this` 指向 vm 实例
                return this.message.split('').reverse().join('')
            }
        }
    })
</script>
```

Original message: "Hello"

Computed reversed message: "olleH"

计算属性 computed

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>

<script>
  var vm = new Vue({
    el: '#example',
    data: {
      message: 'Hello'
    },
    computed: {
      // 计算属性的 getter
      reversedMessage: function () {
        // `this` 指向 vm 实例
        return this.message.split('').reverse().join('')
      }
    }
  })
</script>
```

前提条件： 引入 Vue.js 库

代码： vue-case-01

计算属性 vs 方法

可以通过在表达式中调用方法来达到同样的效果：

```
<p>Reversed message: "{{ reversedMessage() }}"</p>
```

```
// 在组件中
methods: {
  reversedMessage: function () {
    return this.message.split('').reverse().join('')
  }
}
```

计算属性 vs 方法

- 可以将同一函数定义为一个方法而不是一个计算属性。两种方式的最终结果确实是完全相同的。然而，不同的是**计算属性是基于它们的响应式依赖进行缓存的**。只在相关响应式依赖发生改变时它们才会重新求值。
- 这就意味着只要 message 还没有发生改变，多次访问 reversedMessage 计算属性会立即返回之前的计算结果，而不必再次执行函数。
- 这也同样意味着下面的计算属性将不再更新，因为 Date.now() 不是响应式依赖：

```
computed: {
  now: function () {
    return Date.now()
  }
}
```

计算属性 vs 方法

- 相比计算属性，每当触发重新渲染时，调用方法将总会再次执行函数。
- 为什么需要缓存？假设有一个性能开销比较大的计算属性 A，它需要遍历一个巨大的数组并做大量的计算。然后可能有其他的计算属性依赖于 A 。如果没有缓存，将不可避免的多次执行 A 的 getter！
- 如果不希望有缓存，请用方法来替代。

computed 计算属性 应用场景

- 计算属性是基于他们的依赖进行缓存的。计算属性只有在它的相关依赖发生改变时才会重新求值。也就是说只要message值没有发生变化，多次访问 reversedMessage 计算属性会立即返回之前的结果，不必再执行函数。

  而method方法只要发生重新渲染，总是会执行该函数。

Vue 实例生命周期钩子

Vue.js 生命周期钩子—— created 用法

- Vue实例在创建时有一系列初始化步骤----例如，它需要建立数据观察，编译模板，创建必要的数据绑定。在此过程中，它也将调用一些生命周期钩子，给自定义逻辑提供运行机会。
- 例如 created 钩子在实例创建之后调用：

```
<script>
    var vm = new Vue({
        data: { a: 1 },
        created: function () {    // `this` 指向 vm 实例
            alert('a is: ' + this.a)
        }
    });
</script>
```

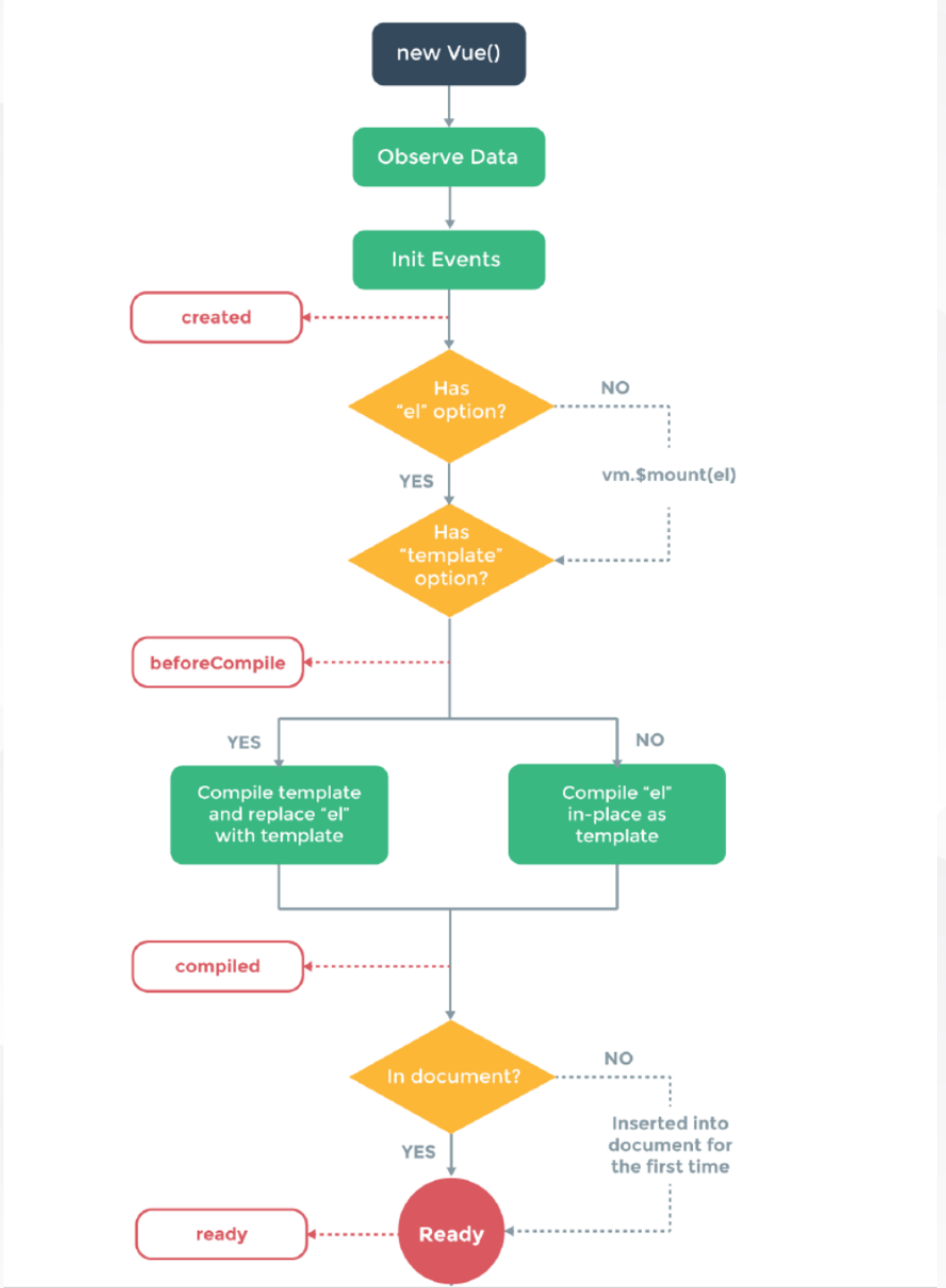
Vue.js 生命周期钩子—— created 用法

```
<script>
    var vm = new Vue({
      data: { a: 1 },
      created: function () {    // `this` 指向 vm 实例
        alert('a is: ' + this.a)
      }
    });
</script>
```

Vue.js 生命周期其他钩子

- Vue的其他钩子，在实例生命周期的不同阶段调用，
  - compiled
  - ready
  - destroyed
- 钩子的 this 指向调用它的 Vue 实例。

Vue 实例生命周期钩子



动手实验