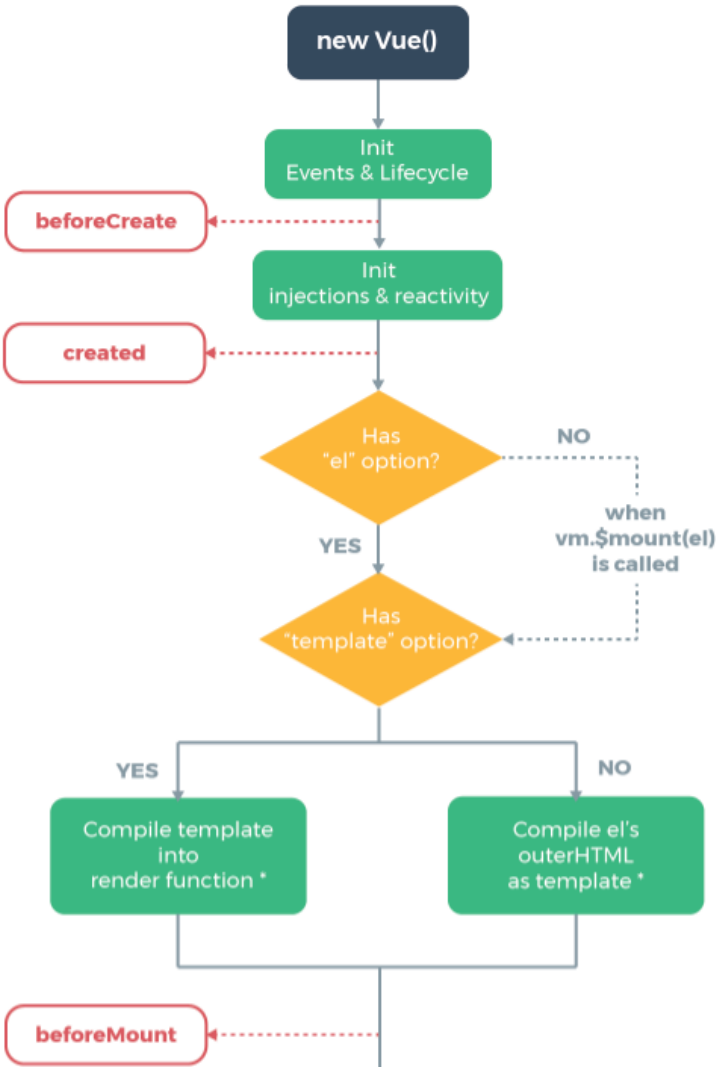
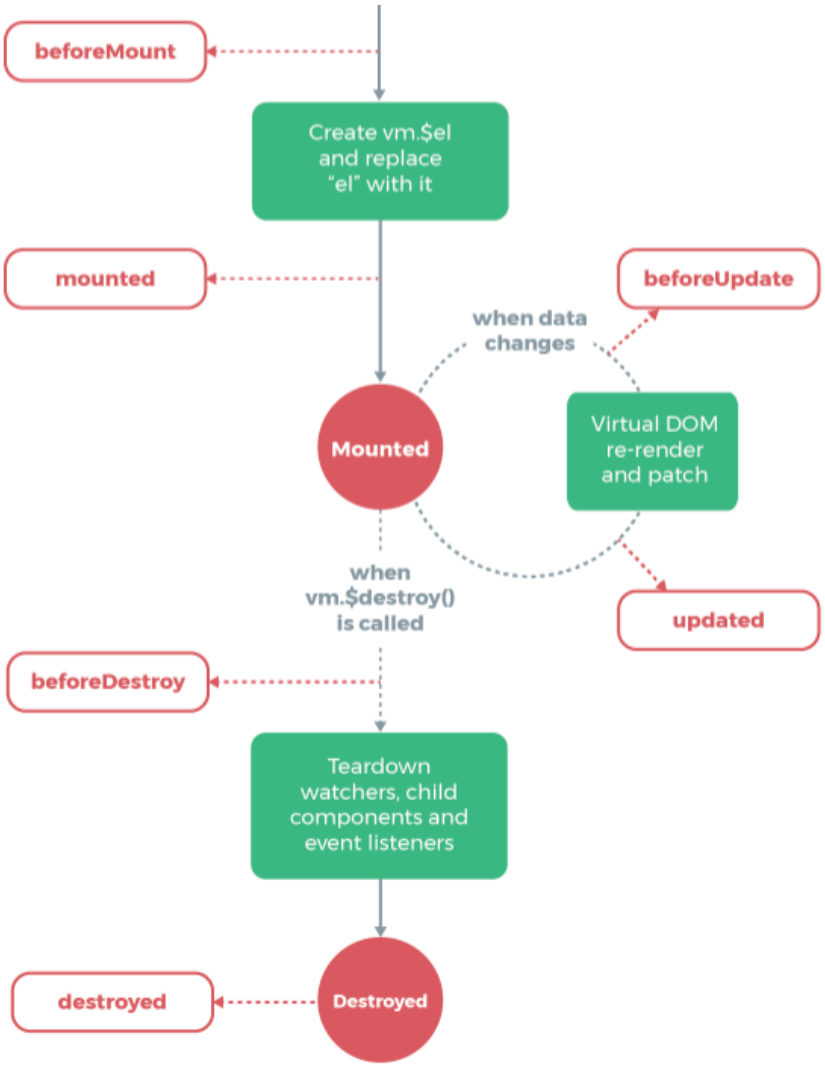
1、`Object.freeze()`，这会阻止修改现有的属性，也意味着响应系统无法再*追踪*变化。

2、除了数据属性，Vue 实例还暴露了一些有用的实例属性与方法。它们都有前缀 `$`，以便与用户定义的属性区分开来。

3、生命周期




beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
when vm.$destroy() is called
updated
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed

Lifecycle hooks

生命周期钩子应该算 vue 这次升级中 broken changes 最多的一部分了，对照 1.0 的文档和 release note，作了下面这张表

vue 1.0+	vue 2.0	Description
init	beforeCreate	组件实例刚被创建，组件属性计算之前，如 data 属性等
created	created	组件实例创建完成，属性已绑定，但 DOM 还未生成，`$el` 属性还不存在
beforeCompile	beforeMount	模板编译/挂载之前
compiled	mounted	模板编译/挂载之后
ready	mounted	模板编译/挂载之后（不保证组件已在 document 中）
-	beforeUpdate	组件更新之前
-	updated	组件更新之后
-	activated	for `keep-alive`，组件被激活时调用
-	deactivated	for `keep-alive`，组件被移除时调用
attached	-	不用了还说啥哪...
detached	-	那就不说了吧...
beforeDestory	beforeDestory	组件销毁前调用
destoryed	destoryed	组件销毁后调用

4、通过使用 v-once 指令

你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。

5、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>
```

6、计算属性和方法区别

不同的是计算属性是基于它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。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。

7、v-if vs v-show

`v-if` 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

`v-if` 也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，`v-show` 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，`v-if` 有更高的切换开销，而 `v-show` 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 `v-show` 较好；如果在运行时条件很少改变，则使用 `v-if` 较好。

8、v-if 与 v-for 一起使用

当 `v-if` 与 `v-for` 一起使用时，`v-for` 具有比 `v-if` 更高的优先级。

9、数组更新检测

变异方法

Vue 包含一组观察数组的变异方法，所以它们也将会触发视图更新。这些方法如下：

- `push()`
- `pop()`
- `shift()`
- `unshift()`
- `splice()`
- `sort()`
- `reverse()`

你打开控制台，然后用前面例子的 `items` 数组调用变异方法：`example1.items.push({ message: 'Baz' })` 。

10、事件

2.3.0 新增

Vue 还对应 `addEventListener` 中的 `passive` 选项提供了 `.passive` 修饰符。

```
<!-- 滚动事件的默认行为 (即滚动行为) 将会立即触发 -->
<!-- 而不会等待 `onScroll` 完成  -->
<!-- 这其中包含 `event.preventDefault()` 的情况 -->
<div v-on:scroll.passive="onScroll">...</div>
```

这个 `.passive` 修饰符尤其能够提升移动端的性能。

11、组件运用

data 必须是函数

构造 Vue 实例时传入的各种选项大多数都可以在组件里使用。只有一个例外：`data` 必须是函数。---防止组件间数据相互影响

组件间传值

使用 Prop 传递数据---父组件传递数据到子组件

组件实例的作用域是孤立的。这意味着不能 (也不应该) 在子组件的模板内直接引用父组件的数据。父组件的数据需要通过 prop 才能下发到子组件中。

子组件要显式地用 `props` 选项声明它预期的数据：

```
Vue.component('child', {
  // 声明 props
  props: ['message'],
  // 就像 data 一样，prop 也可以在模板中使用
  // 同样也可以在 vm 实例中通过 this.message 来使用
  template: '<span>{{ message }}</span>'
})
```

Prop 验证

我们可以为组件的 prop 指定验证规则。如果传入的数据不符合要求，Vue 会发出警告。这对于开发给他人使用的组件非常有用。

要指定验证规则，需要用对象的形式来定义 prop，而不能用字符串数组：`required(必传)`

```
Vue.component('example', {
  props: {
    // 基础类型检测 (`null` 指允许任何类型)
    propA: Number,
    // 可能是多种类型
    propB: [String, Number],
    // 必传且是字符串
    propC: {
      type: String,
      required: true
    },
    // 数值且有默认值
    propD: {
      type: Number,
      default: 100
    },
```

```
  // 数组/对象的默认值应当由一个工厂函数返回
  propE: {
    type: Object,
    default: function () {
      return { message: 'hello' }
    }
  },
  // 自定义验证函数
  propF: {
    validator: function (value) {
      return value > 10
    }
  }
 }
})
```

使用 v-on 绑定自定义事件---子组件传递数据到父组件

每个 Vue 实例都实现了事件接口，即：

- 使用 `$on(eventName)` 监听事件
- 使用 `$emit(eventName, optionalPayload)` 触发事件

Vue 的事件系统与浏览器的 EventTarget API 有所不同。尽管它们的运行起来类似，但是 `$on` 和 `$emit` 并不是 `addEventListener` 和 `dispatchEvent` 的别名。

另外，父组件可以在使用子组件的地方直接用 `v-on` 来监听子组件触发的事件。

不能用 `$on` 监听子组件释放的事件，而必须在模板里直接用 `v-on` 绑定，参见下面的例子。

下面是一个例子：

```
<div id="counter-event-example">
  <p>{{ total }}</p>
  <button-counter v-on:increment="incrementTotal"></button-counter>
  <button-counter v-on:increment="incrementTotal"></button-counter>
</div>
```

```
Vue.component('button-counter', {
  template: '<button v-on:click="incrementCounter">{{ counter }}</button>',
  data: function () {
    return {
      counter: 0
    }
  },
  methods: {
    incrementCounter: function () {
      this.counter += 1
      this.$emit('increment')
    }
  },
})

new Vue({
  el: '#counter-event-example',
  data: {
    total: 0
  },
  methods: {
    incrementTotal: function () {
      this.total += 1
    }
  }
})
```

切换写法

```
<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```