**Vue 导论 - 什么是 Vue.js**

**# 前言**

**作为还在漫漫前端**

在这个系列里我希望能从零开始讲解所有关于 Vue.js 的知识,

并结合实战项目和源码分析, 来让读者可以在最后做到全面的掌握 Vue.js 框架.

要先讲解一下到底什么是 Vue, 我们要学的到底是个什么东西.

在讲解之前, 首先结论先行: **Vue 是一个采用 MVVM 架构, 通过数据驱动视图的形式来构建用户界面的渐进式框架.**

**# "渐进式框架" & "自底向上逐层应用"**

Vue 是一套用于构建用户界面的**渐进式框架**。Vue 被设计为可以**自底向上逐层应用**。

"渐进式" 的意思就是从少到多, 从弱到强, Vue 框架并不强制开发者一下子就使用它的全部.

（小程序相等于VUE）

图示

描述已自动生成

Vue 从设计角度来讲，可以分成上图的这几大部分，但是开发者并不需要把所有的东西都学会, 或者都用上. "声明式渲染" 和 "组件系统" 是 Vue 核心库所包含内容，而 "客户端路由"、"状态管理"、"构建工具" 都有其他各自专门的解决方案。这些解决方案相互独立，**Vue 将框架做了分层设计, 你可以在 Vue 核心库的基础上灵活地在不同层上使用其他方案, 而不是说一定要用 Vue 提供的**.

**# 数据驱动视图**

**Vue 是数据驱动视图的框架, 采用了 MVVM 架构. 其核心库只关注视图层, 视图与数据状态保持同步**.

**## 什么是 MVC 架构?**

一个清晰合理的架构是一个复杂程序所必备的. MVVM 是一种软件的架构模式. 但在讲它之前, 要先了解一下 MVC 架构.

在 MVC 架构下, 软件可以分成三个部分:

* • 视图 (View): 用户界面
* • 控制器 (Controller): 业务逻辑
* • 模型 (Model): 数据保存

图示

描述已自动生成

它们各个部分之间的通信方式如上图:

* • 用户在 View 上进行操作, 触发相应事件;
* • Controller 接到事件, 执行相应业务逻辑, 调用 Model 的接口修改数据状态;
* • Model 中新的数据发送到 View, 视图更新, 用户得到反馈;

当然上述所讲为一般而言的 MVC 模式. 在不同的项目中, MVC 的实现方法可能都不尽相同. 比如, 网页开发中, 某些项目除了允许用户在页面上触发 DOM 事件, 从而让相关事件处理函数得到执行外. 用户还可以通过在 URL 后加上不同的 # 锚点来触发 hashChange 事件, 从而让相关业务逻辑得到执行.

**## 什么是 MVVM 架构?**

在 MVVM 架构中, Controller 被替换成了 ViewModel. 它作为 View 与 Model 沟通的桥梁, 负责监听 Model 或 View 的修改. 由此实现了 View 和 Model 的 "双向绑定". 意思就是说, 当 Model 层数据进行修改时, ViewModel 层检测到了变化, 然后通知 View 层进行相应的视图修改. 反之, 当 View 层做了修改时, Model 层的数据也会得到相应的修改.

图示

描述已自动生成

**## 响应式的双向数据绑定**

在页面中, **DOM 状态其实就是数据状态的一种映射**. 我们的业务逻辑应该关注于对于数据状态的操作. 当数据发生改变的时候, 框架应当帮助我们自动更新视图, 而不需要我们再手动获取 DOM 元素, 然后再对它进行修改了.

**Vue 的核心是一个 "响应式的数据绑定系统"**. 数据和视图建立绑定之后, DOM 将与数据保持同步. 当数据被修改了, DOM 便相应的更新; 同样, 用户在视图上的操作也会自动更新相关的数据. 所有关于 DOM 的操作都被 Vue 在背后帮我们执行了, 我们只需关注于数据就好了. 这种思想也被称为 "数据驱动"

简化地表示 "数据" 与 "视图" 的关系就如下图:

图示

描述已自动生成

**## 声明式渲染 && 虚拟 DOM**

**Vue 提供一套基于 HTML 的模板语法, 允许开发者声明式地将真实 DOM 与 Vue 实例的数据绑定在一起**。

首先, "声明式" 的意思就是: 开发者告诉 Vue 框架想要什么结果, 而如何达成由 Vue 自己去做, 开发者不用管. 例如下面这段模板代码声明式地让 Vue 将 name 的数据和 <p> 标签绑定在一起:

<p>Hello {{ name }}</p>

Vue 会把这些模板编译成一个渲染函数。该函数被调用后会渲染并且返回一个虚拟的 DOM 树. 这个 "树" 的职责就是描述当前视图应处的状态。之后再通过一个 Patch 函数，将这些虚拟 DOM 创建成真实的 DOM 元素。在这个过程中, Vue 有一个 "依赖追踪系统" 会侦测在渲染过程中所依赖到的数据来源. 当数据源发生变动时, 就可以根据需要重新渲染虚拟 DOM 树。当一个新的虚拟树被渲染出来之后, Vue 会将其与老树进行对比，并最终得出应施加到真实 DOM 上的改动, 然后通过 Patch 函数施加改动。

图示

描述已自动生成

同时, Vue 也允许用户直接跳过写模板这一层去手写渲染函数. 模板和渲染函数都各有利弊, 首先模板更贴近 HTML，有着良好的可读性的同时, 可以让我们更直观地思考语义结构，更好地结合 CSS 的书写。而直接写渲染函数，因为写的是真正的 JavaScript 代码，函数内可以进行更复杂的逻辑判断，可以选择性地返回最终要返回的 DOM 结构，从而实现一些在模板的语法限制下，很难做到的一些事情。但虽然两个都是可以选择的。在绝大部分情况下还是推荐使用模板，只有在需要复杂逻辑的情况下，才使用渲染函数。