Vue.js

什么是Vue.js

- Vue.js 是目前最火的一个前端框架，React是最流行的一个前端框架（React除了开发网站，还可以开发手机App， Vue语法也是可以用于进行手机App开发的）
- Vue.js 是前端的**主流框架之一**，和Angular.js、React.js 一起，并成为前端三大主流框架！
- Vue.js 是一套构建用户界面的框架，**只关注视图层**，它不仅易于上手，还便于与第三方库或既有项目整合。（Vue有配套的第三方类库，可以整合起来做大型项目的开发）
- 前端的主要工作？主要负责MVC中的V这一层；主要工作就是和界面打交道，来制作前端页面效果；

为什么要学习流行框架

- 企业为了提高开发效率：在企业中，时间就是效率，效率就是金钱；
- 企业中，使用框架，能够提高开发的效率；
- 提高开发效率的发展历程：原生JS -> Jquery之类的类库 -> 前端模板引擎 -> Angular.js / Vue.js（能够帮助我们减少不必要的DOM操作；提高渲染效率；双向数据绑定的概念【通过框架提供的指令，我们前端程序员只需要关心数据的业务逻辑，不再关心DOM是如何渲染的了】）
- 在Vue中，一个核心的概念，就是让用户不再操作DOM元素，解放了程序员的双手，让程序员可以更多的时间去关注业务逻辑；

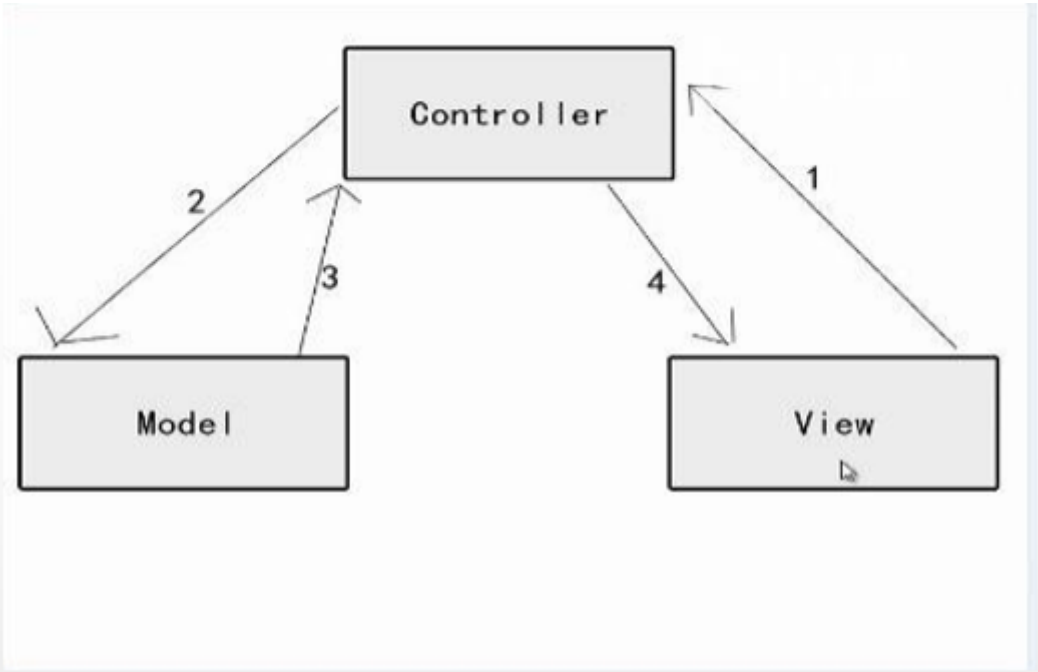
对我们的好处：

- 增强自身的能力
- 人无我有，人有我优
- 平时不上课的课余时间，都在干吗？

框架和库的区别

- 框架：是一套完整的解决方案；对项目的侵入性较大，项目如果需要更换框架，则需要重新架构整个项目。
- 库（插件）：提供某一个小功能，对项目的侵入性较小，如果某个库无法完成某些需求，可以很容易切换到其它库实现需求。

MVC 与MVVM 之间的区别



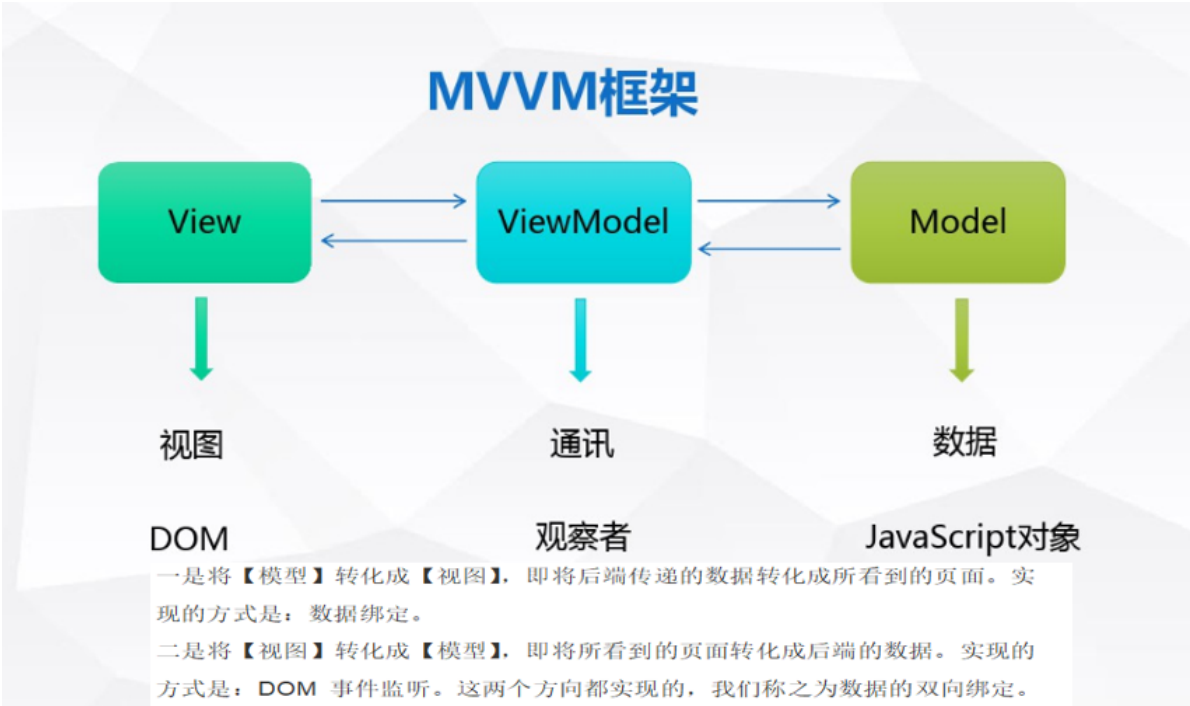
第一步 浏览者 -> 调用控制器,对他发出指令

第二步 控制器 -> 按指令选取一个合适的模型

第三步 模型 -> 按控制器指令取相应数据

第四步 控制器 -> 按指令选取相应视图

第五步 视图 -> 把第三步取到的数据按用户想要的样子显示出来



- MVC 是后端的分层开发概念;
- MVVM是前端视图层的概念，主要关注于 视图层分离，也就是说：MVVM把前端的视图层，分为了三部分 Model, View , VM ViewModel
- 为什么有了MVC还要有MVVM

Vue.js 基本代码 和 MVVM 之间的对应关系

Vue之 - 基本的代码结构

Vue指令之v-text和v-html

Vue指令之v-bind的三种用法

1. 直接使用指令 `v-bind`
2. 使用简化指令 `:`
3. 在绑定的时候，拼接绑定内容：`:title="btnTitle + '，这是追加的内容'"`

Vue指令之v-on和跑马灯效果

跑马灯效果

1. HTML结构：

```
<div id="app">

    <p>{{info}}</p>

    <input type="button" value="开启" v-on:click="go">

    <input type="button" value="停止" v-on:click="stop">

  </div>
```

2. Vue实例：

```
    // 创建 Vue 实例，得到 ViewModel

    var vm = new Vue({

      el: '#app',

      data: {

        info: '猥琐发育，别浪~! ',

        intervalId: null

      },

      methods: {

        go() {

          // 如果当前有定时器在运行，则直接return
```

```
        if (this.intervalId != null) {
          return;
        }
        // 开始定时器
        this.intervalId = setInterval(() => {
          this.info = this.info.substring(1) + this.info.substring(0, 1);
        }, 500);
      },
      stop() {
        clearInterval(this.intervalId);
      }
    }
  });
```

Vue指令之v-on的缩写和事件修饰符

事件修饰符：

- .stop　　阻止冒泡
- .prevent　阻止默认事件
- .capture　添加事件侦听器时使用事件捕获模式
- .self　　只当事件在该元素本身（比如不是子元素）触发时触发回调
- .once　　事件只触发一次

Vue指令之v-model和双向数据绑定

简易计算器案例

1. HTML 代码结构

```
<div id="app">
  <input type="text" v-model="n1">
  <select v-model="opt">
    <option value="0">+</option>
    <option value="1">-</option>
```

```
    <option value="2">*</option>
    <option value="3">÷</option>
  </select>
  <input type="text" v-model="n2">
  <input type="button" value="=" v-on:click="getResult">
  <input type="text" v-model="result">
</div>
```

2. Vue实例代码：

```
// 创建 Vue 实例，得到 ViewModel
var vm = new Vue({
  el: '#app',
  data: {
    n1: 0,
    n2: 0,
    result: 0,
    opt: '0'
  },
  methods: {
    getResult() {
      switch (this.opt) {
        case '0':
          this.result = parseInt(this.n1) + parseInt(this.n2);
          break;
        case '1':
          this.result = parseInt(this.n1) - parseInt(this.n2);
          break;
        case '2':
          this.result = parseInt(this.n1) * parseInt(this.n2);
```

```
            break;
          case '3':
            this.result = parseInt(this.n1) / parseInt(this.n2);
            break;
        }
      }
    }
  });
```

Vue指令之`v-for`和`key`属性

1. 迭代数组

```
<ul>
  <li v-for="(item, i) in list">索引：{{i}} --- 姓名：{{item.name}} --- 年龄：
{{item.age}}</li>
</ul>
```

2. 迭代对象中的属性

```
    <!-- 循环遍历对象身上的属性 -->
    <div v-for="(val, key, i) in userInfo">{{val}} --- {{key}} --- {{i}}</div>
```

3. 迭代数字

```
<p v-for="i in 10">这是第 {{i}} 个P标签</p>
```

> 2.2.0+ 的版本里，**当在组件中使用** v-for 时，key 现在是必须的。

当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用 “**就地复用**” 策略。如果数据项的顺序被改变，Vue将**不是移动 DOM 元素来匹配数据项的顺序**， 而是**简单复用此处每个元素**，并且确保它在特定索引下显示已被渲染过的每个元素。

为了给 Vue 一个提示，**以便它能跟踪每个节点的身份，从而重用和重新排序现有元素**，你需要为每项提供一个唯一 key 属性。

Vue指令之`v-if`和`v-show`

> 一般来说，v-if 有更高的切换消耗而 v-show 有更高的初始渲染消耗。因此，如果需要频繁切换 v-show 较好，如果在运行时条件不大可能改变 v-if 较好。

过滤器

概念：Vue.js 允许你自定义过滤器，**可被用作一些常见的文本格式化。**过滤器可以用在两个地方：**mustache 插值和 v-bind 表达式。**过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符指示；

私有过滤器

1. HTML元素：

```
<td>{{item.ctime | dataFormat('yyyy-mm-dd')}}</td>
```

2. 私有 `filters` 定义方式：

```
filters: { // 私有局部过滤器，只能在 当前 VM 对象所控制的 View 区域进行使用

    dataFormat(input, pattern = "") { // 在参数列表中 通过 pattern="" 来指定形参默认值，防止报错

      var dt = new Date(input);

      // 获取年月日

      var y = dt.getFullYear();

      var m = (dt.getMonth() + 1).toString().padStart(2, '0');

      var d = dt.getDate().toString().padStart(2, '0');


      // 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日

      // 否则，就返回  年-月-日 时：分：秒

      if (pattern.toLowerCase() === 'yyyy-mm-dd') {

        return `${y}-${m}-${d}`;

      } else {

        // 获取时分秒

        var hh = dt.getHours().toString().padStart(2, '0');

        var mm = dt.getMinutes().toString().padStart(2, '0');

        var ss = dt.getSeconds().toString().padStart(2, '0');
```

```
        return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;
      }
    }
  }
```

> 使用ES6中的字符串新方法 String.prototype.padStart(maxLength, fillString='') 或 String.prototype.padEnd(maxLength, fillString='')来填充字符串；

全局过滤器

```
// 定义一个全局过滤器
Vue.filter('dataFormat', function (input, pattern = '') {
  var dt = new Date(input);
  // 获取年月日
  var y = dt.getFullYear();
  var m = (dt.getMonth() + 1).toString().padStart(2, '0');
  var d = dt.getDate().toString().padStart(2, '0');

  // 如果 传递进来的字符串类型，转为小写之后，等于 yyyy-mm-dd，那么就返回 年-月-日
  // 否则，就返回  年-月-日 时：分：秒
  if (pattern.toLowerCase() === 'yyyy-mm-dd') {
    return `${y}-${m}-${d}`;
  } else {
    // 获取时分秒
    var hh = dt.getHours().toString().padStart(2, '0');
    var mm = dt.getMinutes().toString().padStart(2, '0');
    var ss = dt.getSeconds().toString().padStart(2, '0');

    return `${y}-${m}-${d} ${hh}:${mm}:${ss}`;
  }
```

```
});
```

> 注意：当有局部和全局两个名称相同的过滤器时候，会以就近原则进行调用，即：局部过滤器优先于全局过滤器被调用！

watch属性的使用

考虑一个问题：想要实现 `名` 和 `姓` 两个文本框的内容改变，则全名的文本框中的值也跟着改变；（用以前的知识如何实现？？？）

1. 监听`data`中属性的改变：

```
<div id="app">
    <input type="text" v-model="firstName"> +
    <input type="text" v-model="lastName"> =
    <span>{{fullName}}</span>
  </div>

  <script>
    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {
        firstName: 'jack',
        lastName: 'chen',
        fullName: 'jack - chen'
      },
      methods: {},
      watch: {
        'firstName': function (newVal, oldVal) { // 第一个参数是新数据，第二个参数是旧数据
          this.fullName = newVal + ' - ' + this.lastName;
        },
        'lastName': function (newVal, oldVal) {
          this.fullName = this.firstName + ' - ' + newVal;
        }
      }
    });
  </script>
```

2. 监听路由对象的改变：

```
<div id="app">
    <router-link to="/login">登录</router-link>
    <router-link to="/register">注册</router-link>

    <router-view></router-view>
  </div>

  <script>
    var login = Vue.extend({
      template: '<h1>登录组件</h1>'
```

```
    });

    var register = Vue.extend({
      template: '<h1>注册组件</h1>'
    });

    var router = new VueRouter({
      routes: [
        { path: "/login", component: login },
        { path: "/register", component: register }
      ]
    });

    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {},
      methods: {},
      router: router,
      watch: {
        '$route': function (newVal, oldVal) {
          if (newVal.path === '/login') {
            console.log('这是登录组件');
          }
        }
      }
    });
  </script>
```

computed计算属性的使用

1. 默认只有getter的计算属性：

```
<div id="app">
    <input type="text" v-model="firstName"> +
    <input type="text" v-model="lastName"> =
    <span>{{fullName}}</span>
  </div>

  <script>
    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {
        firstName: 'jack',
        lastName: 'chen'
      },
      methods: {},
      computed: { // 计算属性； 特点：当计算属性中所以来的任何一个 data 属性改变之后，都会重新触发 本计算属性 的重新计算，从而更新 fullName 的值
        fullName() {
          return this.firstName + ' - ' + this.lastName;
        }
      }
    });
  </script>
```

2. 定义有`getter`和`setter`的计算属性：

```
<div id="app">
    <input type="text" v-model="firstName">
    <input type="text" v-model="lastName">
    <!-- 点击按钮重新为 计算属性 fullName 赋值 -->
    <input type="button" value="修改fullName" @click="changeName">

    <span>{{fullName}}</span>
  </div>

  <script>
    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {
        firstName: 'jack',
        lastName: 'chen'
      },
      methods: {
        changeName() {
          this.fullName = 'TOM - chen2';
        }
      },
      computed: {
        fullName: {
          get: function () {
            return this.firstName + ' - ' + this.lastName;
          },
          set: function (newVal) {
            var parts = newVal.split(' - ');
            this.firstName = parts[0];
            this.lastName = parts[1];
          }
        }
      }
    });
  </script>
```

`watch`、`computed`和`methods`之间的对比

1. `computed`属性的结果会被缓存，除非依赖的响应式属性变化才会重新计算。主要当作属性来使用；
2. `methods`方法表示一个具体的操作，主要书写业务逻辑；
3. `watch`一个对象，键是需要观察的表达式，值是对应回调函数。主要用来监听某些特定数据的变化，从而进行某些具体的业务逻辑操作；可以看作是`computed`和`methods`的结合体；