

## 第3章 前端开发基础

# 目录

1. Web前端简介
2. Vue.js介绍
3. Node.js入门

# 前言

- 本章主要介绍了Web前端的基础知识，包括浏览器架构、HTML、CSS和JavaScript的基础知识；并对Vue.js和Node.js进行了介绍。

# 目标

---

- 学完本课程后，你可以
  - 了解浏览器的基本原理
  - 了解HTML、CSS和JavaScript的基本用法和原理
  - 了解Vue.js的原理并创建第一个Vue.js项目
  - 了解Node.js的基本使用

# 目录

---

## 1. Web前端

- 浏览器架构
- HTML
- CSS
- JavaScript

## 2. Vue.js

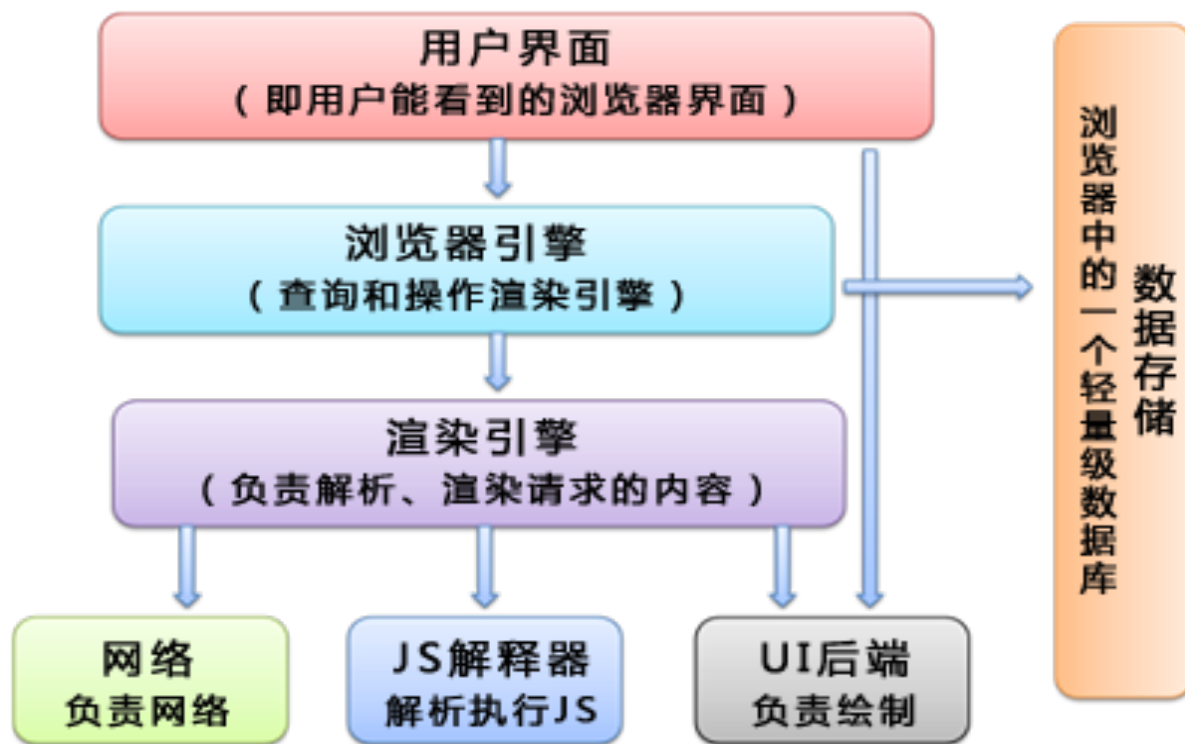
## 3. Node.js

# 本节概述和学习目标

---

- 本节介绍Web前端的基础知识。主要是浏览器架构、HTML、CSS和JavaScript的介绍。

# 浏览器架构



# 浏览器架构

- 浏览器架构各构件的作用
  - 用户界面：主要负责与用户的交互，如地址栏输入、刷新/向前/后退按钮。
  - 浏览器引擎：协调上层的User Interface和下层的渲染引擎（Rendering Engine）。Browser Engine主要是实现浏览器的动作（初始化加载，刷新、向前、后退等）。
  - 渲染引擎：为给定的URL提供可视化展示，它解析HTML、XML、JavaScript。
  - 底层组件库
    - Networking负责基于HTTP和FTP协议处理网络请求，还提供文档的缓存功能以减少网络传输。
    - JavaScript Interpreter解释器负责解释执行页面的js代码，得到的结果传输给Rendering Engine。
    - UI Backend是UI的基础控件，隔离平台的实现，提供平台无关的接口给上层。
    - Data Storage则是负责对用户数据、书签、Cookie和偏好设置等数据的持久化工作。



# HTML

- 超文本标记语言（HyperText Markup Language, HTML）是一种用于创建网页的标准标记语言，用来描述网页的内容。
  - HTML 标签通常是成对出现的，比如 `<b>` 和 `</b>`
  - 标签内可以嵌套标签

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>Title</title>
</head>

<body>
<h1>Heading 1</h1>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
</body>

</html>
```

← → ↻ ⓘ localhost:63344/HelloWorld/hello.html

## Heading 1

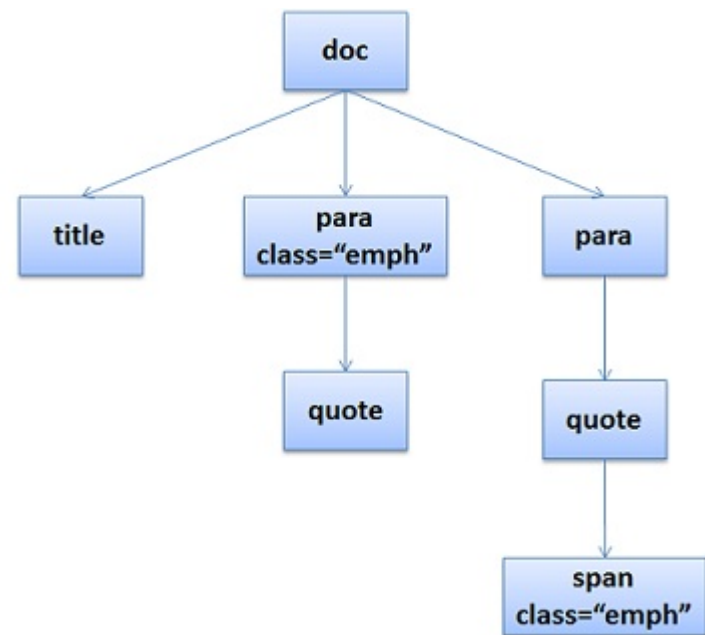
This is a paragraph.

This is a paragraph.

# HTML解析树

- HTML 运行在浏览器上，由浏览器来解析，解析生成DOM(Document Object Model) Tree。
  - DOM Tree是由HTML文件解析（parse）生成代表内容的树状结构。

```
<doc>
<title>A few quotes</title>
<para>
  Franklin said that <quote>"A penny saved is a penny earned."</quote>
</para>
<para>
  FDR said <quote>"We have nothing to fear but <span>fear itself.</span>"</
quote>
</para>
</doc>
```

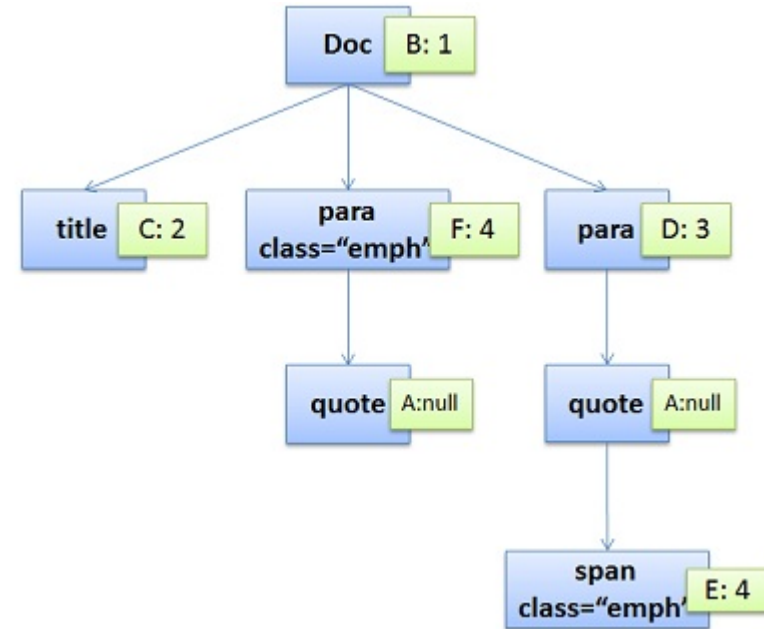
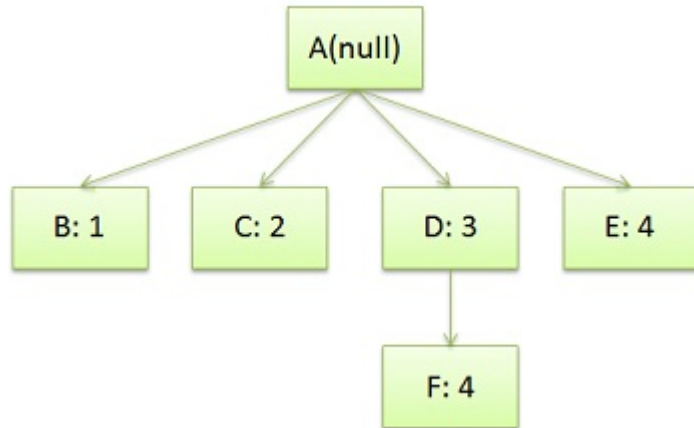


# CSS

- 层叠样式表（Cascading Style Sheets, CSS）用于控制网页的样式和布局。
  - CSS不能单独使用，必须与HTML或XML一起协同工作，为HTML或XML起装饰作用。
  - HTML负责确定网页中有哪些内容，CSS确定以何种外观（大小、粗细、颜色、对齐和位置）展现这些元素。
  - CSS可以用于设定页面布局、设定页面元素样式、设定适用于所有网页的全局样式。
  - CSS可以零散地直接添加在要应用样式的网页元素上，也可以集中化内置于网页、链接式引入网页以及导入式引入网页。

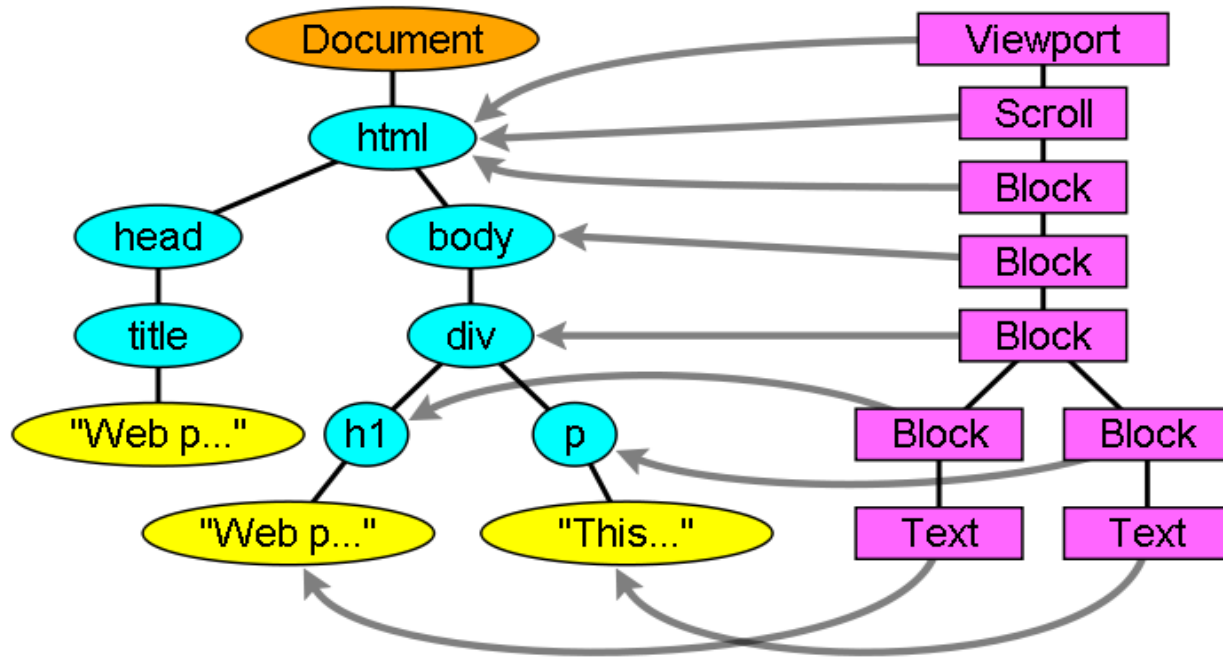
# CSS

```
/* rule 1 */ doc { display: block; text-indent: 1em; }  
/* rule 2 */ title { display: block; font-size: 3em; }  
/* rule 3 */ para { display: block; }  
/* rule 4 */ [class="emph"] { font-style: italic; }
```

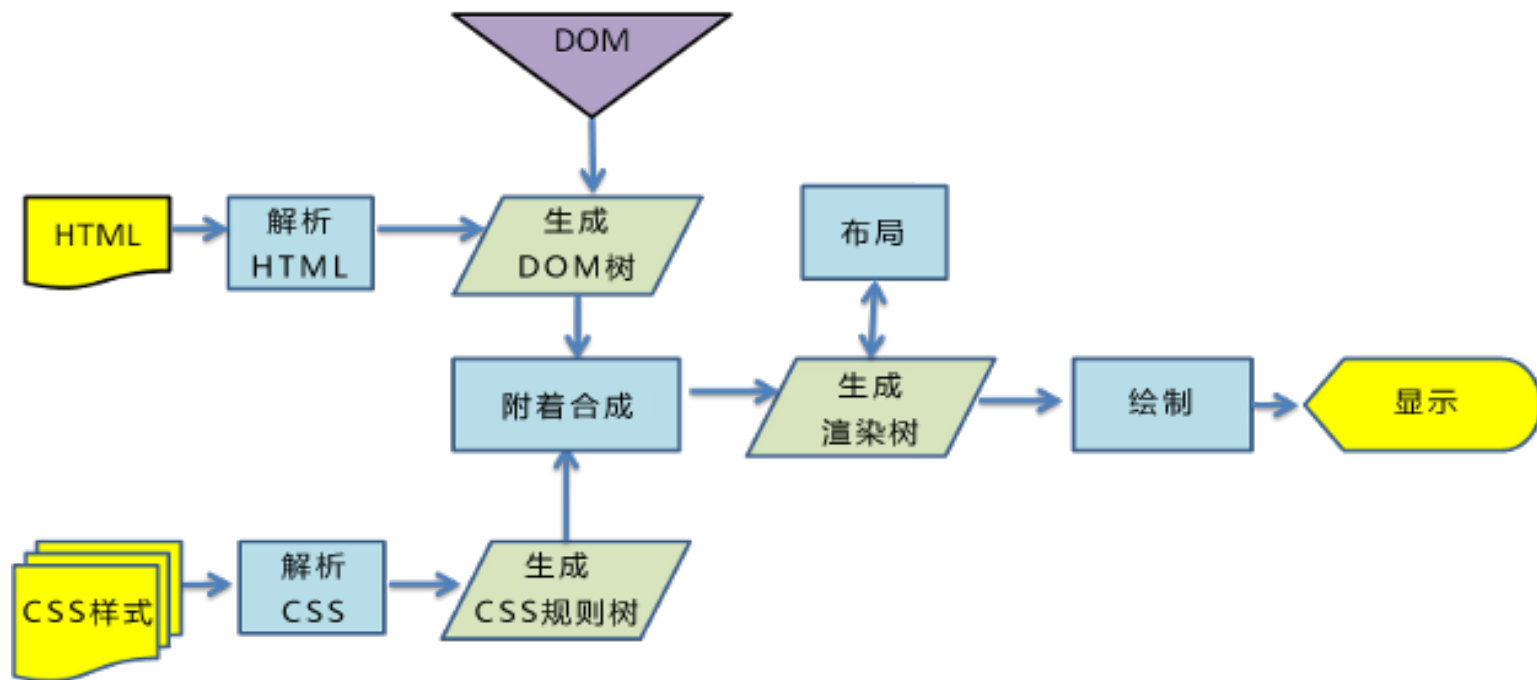


# 渲染树

- 渲染树（Render Tree）是渲染流程的输出目标。
  - 它包含具有显示属性（颜色和大小）的长方形组成的树状结构。
  - 渲染的目标就是将最初写的页面内容HTML、样式CSS渲染为最后的Render Tree，然后调用系统图形API来显示。



# 渲染流程



# 渲染流程

- 渲染流程如下：
  - 解析（parse）： parse HTML/SVG/XHTML文件，产生DOM Tree； parse CSS文件生成CSS Rule Tree。
  - 附着合成（construct）： DOM树和CSS规则树连接在一起construct形成Render Tree（渲染树）。
  - 布局（reflow/layout）： 计算出Render Tree每个节点的具体位置。
  - 绘制（paint）： 调用系统图形API，通过显卡，将Layout后的节点内容分别呈现到屏幕上。

# JavaScript

- 用HTML和CSS已经可以显示页面的静态内容，但是网页还需要和用户和服务端进行交互。所以，可以用JavaScript语言完成与用户的交互。
  - 可以直接在HTML文件中加入JavaScript脚本，也可以使用单独的文件，再导入。
  - JavaScript语言可以通过API完成对DOM Tree和CSS Tree的操作，可以完成和用户的交互，完成和远端服务器的交互，也可以执行简单的前端和业务的逻辑。



# JavaScript

- script in HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path: Script</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
    <script>
      var span = document.getElementsByTagName('span')[0];
      span.textContent = 'interactive'; // change DOM text content
      span.style.display = 'inline'; // change CSSOM property
      // create a new element, style it, and append it to the DOM
      var loadTime = document.createElement('div');
      loadTime.textContent = 'You loaded this page on: ' + new Date();
      loadTime.style.color = 'blue';
      document.body.appendChild(loadTime);
    </script>
  </body>
</html>
```

# JavaScript

- script in js file

```
<!DOCTYPE html><html>
<head>
  <meta name="viewport" content="width=device-width,initial-scale=1">
  <link href="style.css" rel="stylesheet">
  <title>Critical Path: Script External</title>
</head>
<body>
  <p>Hello <span>web performance</span> students!</p>
  <div></div>
  <script src="app.js"></script>
</body>
</html>
```

html file

```
var span = document.getElementsByTagName('span')[0];
span.textContent = 'interactive'; // change DOM text content
span.style.display = 'inline'; // change CSSOM property
// create a new element, style it, and append it to the DOM
var loadTime = document.createElement('div');
loadTime.textContent = 'You loaded this page on: ' + new Date();
loadTime.style.color = 'blue';
document.body.appendChild(loadTime);
```

js file

# JavaScript

- Async关键字
  - 告诉浏览器在等待脚本可用时不要阻止DOM构造，将脚本的执行和DOM的构造异步化，这可以显著提高性能。

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link href="style.css" rel="stylesheet">
    <title>Critical Path: Script Async</title>
  </head>
  <body>
    <p>Hello <span>web performance</span> students!</p>
    <div></div>
    <script src="app.js" async></script>
  </body>
</html>
```

# 目录

## 1. Web前端

## 2. Vue.js

- Vue.js介绍
- Vue.js安装配置
- Vue.js基本使用

## 3. Node.js

# 本节概述和学习目标

---

- 本节介绍Vue的特性、安装步骤和简单的入门案例。
- 学会Vue.js的安装和基本使用。

# Vue.js介绍

- Vue.js是一个用于创建用户界面的开源JavaScript框架，也是一个创建单页应用的Web应用框架。



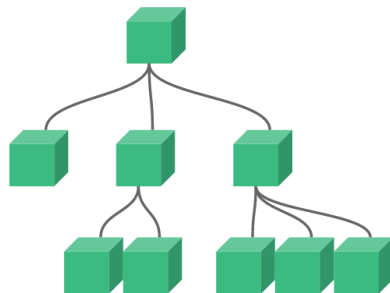
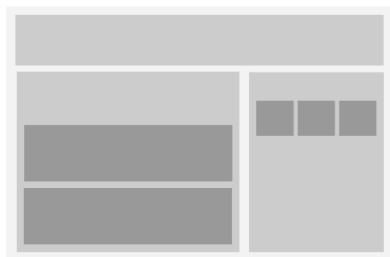
# Vue.js特性 – 声明式渲染

- Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统。
  - 下列代码中界面<div>标签内的内容和数据message实现了双向绑定，当message改变的时候，界面也会自动随之而改变。

```
<div id="app">
  {{ message }}
</div>
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

# Vue.js特性 – 组件化应用构建

- 组件系统是 Vue 的另一个重要概念，因为它是一种抽象，允许我们使用小型、独立和通常可复用的组件构建大型应用。
  - 在 Vue 里，一个组件本质上是一个拥有预定义选项的一个 Vue 实例。
  - 几乎任意类型的应用界面都可以抽象为一个组件树。



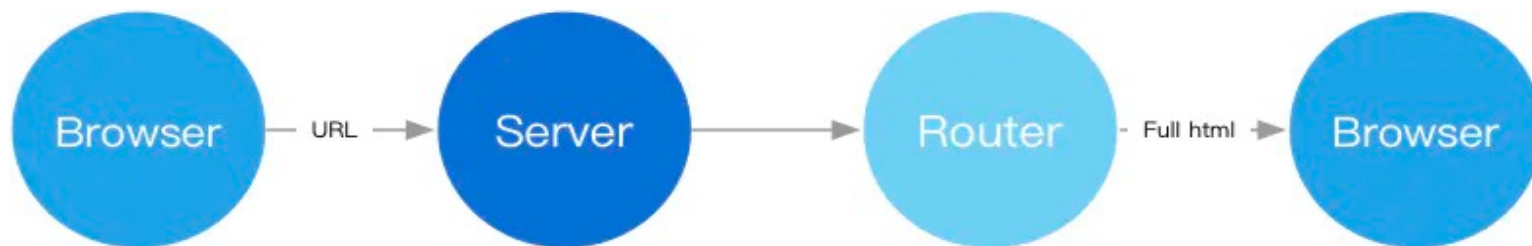
```
// 定义名为 todo-item 的新组件
Vue.component('todo-item', {
  template: '<li>这是个待办项</li>'
})

var app = new Vue(...)
```



# Vue.js特性 – 前端路由

- 路由是将URL请求映射到代码的过程。当我们在Web应用中点击链接。URL的改变将会给用户新的数据或者跳转新的网页。
  - 传统路由是服务器端路由，或者说后端路由。
    - 浏览器向当前URL路径下的子路径/hello发送GET请求（ContentType一般为text/html）
    - 服务端解析请求并进行路由，向浏览器发送HTML
    - 浏览器解析DOM、CSS、JS并进行渲染



# Vue.js特性 – 前端路由

- 点击链接之后，不需要后端发送完整的页面，甚至不需要向后端请求。这就是单页面应用。（Single Page Application, SPA）。单页面的应用一般采用的就是前端路由。
  - 在首屏加载时，需要对应用需要的资源进行完整请求；
  - 点击链接时检测请求事件，阻止浏览器发送GET请求；
  - 前端路由代码改变地址栏URL（利用Hash、HTML5 History API等）；
  - 如果需要从服务端接收数据，发送Ajax请求获取需要的数据即可(JSON等)；
  - 路由代码对页面需要改变的DOM结构进行加载和渲染。



# Vue.js特性 – 前端路由

- vue-router的功能
  - 前端路由：让页面中的部分内容可以无刷新的跳转，就像原生APP一样。
  - 懒加载：结合异步组件以及在组件的created钩子上触发获取数据的Ajax请求可以最大化的降低加载时间，减少流量消耗。
  - 重定向：可以实现某些需要根据特定逻辑改变页面原本路由的需求，比如说未登录状态下访问“个人信息”时应该重定向到登录页面。
  - 美化URL：通过HTML5 History 模式优化URL。

# Vue.js特性 – 大规模状态管理

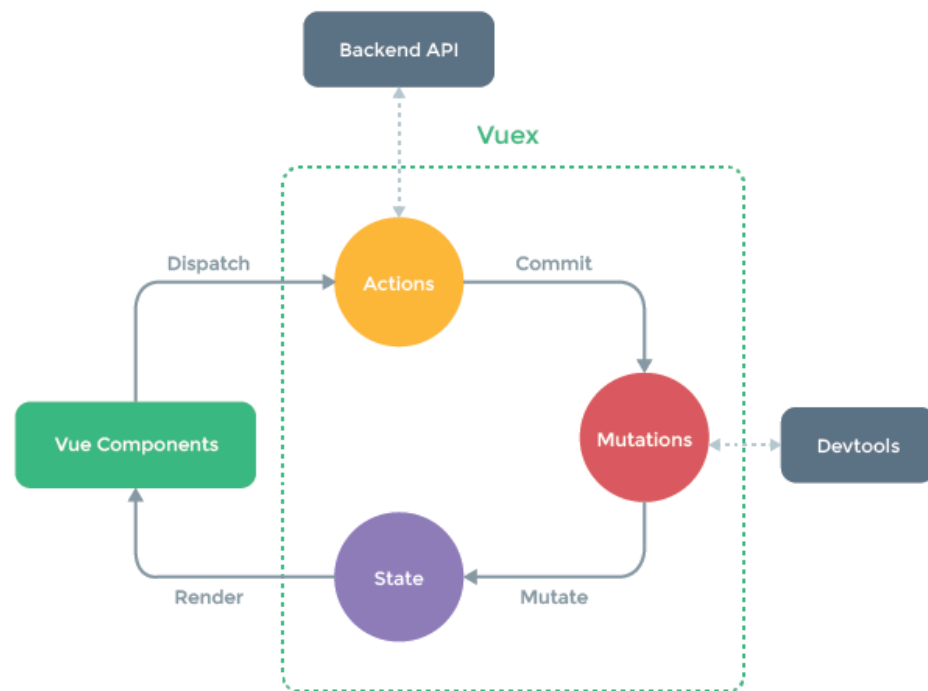
- Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。
  - 它采用集中式存储管理应用的所有组件的状态；
  - 它包含以下几部分：
    - state, 驱动应用的数据源
    - view, 以声明方式将 state 映射到视图
    - actions, 响应在 view 上的用户输入导致的状态变化
  - Vuex强调数据流的单向流动。

# Vue.js特性 – 大规模状态管理

- 当我们的应用遇到多个组件共享状态时，单向数据流的简洁性很容易被破坏：
  - 多个视图依赖于同一状态
    - 传参的方法对于多层嵌套的组件将会非常繁琐，并且对于兄弟组件间的状态传递无能为力。
  - 来自不同视图的行为需要变更同一状态
    - 我们经常会采用父子组件直接引用或者通过事件来变更和同步状态的多份拷贝。

# Vue.js特性 – 大规模状态管理

- 解决方案：把组件的共享状态抽取出来，以一个全局单例模式管理。
  - 通过定义和隔离状态管理中的各种概念并通过强制规则维持视图和状态间的独立性，我们的代码将会变得更结构化且易维护。
  - Vue组件接收交互行为，调用dispatch方法触发action相关处理，若页面状态需要改变，则调用commit方法提交mutation修改state，通过getters获取到state新值，重新渲染Vue Components，界面随之更新。



# Vue.js特性 – 强大的构建工具

- Vue CLI 是一个基于 Vue.js 进行快速开发的完整系统，提供：
  - 通过 @vue/cli 实现的交互式的项目脚手架。
  - 通过 @vue/cli + @vue/cli-service-global 实现的零配置原型开发。
  - 一个运行时依赖 (@vue/cli-service)，该依赖：
    - 可升级；
    - 基于 webpack 构建，并带有合理的默认配置；
    - 可以通过项目内的配置文件进行配置；
    - 可以通过插件进行扩展。
  - 一个丰富的官方插件集合，集成了前端生态中最好的工具。
  - 一套完全图形化的创建和管理 Vue.js 项目的用户界面。

# Vue.js安装配置

- 安装Node.js和npm
  - 访问<https://nodejs.org/en/>官方网址，选择对应操作系统的安装软件。
  - npm:为Nodejs下的包管理器。一般安装Node.js后，会自动安装npm。

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#).

#BlackLivesMatter

The 2021 Node.js User Survey is open now

Download for macOS (x64)

14.15.4 LTS

Recommended For Most Users

15.8.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

```
[QindeMacBook-Pro:~ qinliu$ node -v  
v10.11.0  
[QindeMacBook-Pro:~ qinliu$ npm -v  
6.14.3
```



# Vue.js安装配置

- 安装cnpm
  - cnpm是中国npm镜像的客户端，用以访问npm相关的镜像文件在国内的备份站点，以提高国内用户镜像文件的下载速度。
  - 运行：`npm install -g cnpm --registry=http://registry.npm.taobao.org` 安装cnpm。

# Vue.js安装配置

- 安装vue-cli
  - vue-cli是一种全局脚手架用于帮助搭建所需的模板框架。
  - 运行：npm install -g vue-cli 安装vue-cli。
  - 运行：vue -v查看vue版本。

# Vue.js安装配置

- 创建第一个Vue.js项目
  - 运行vue init webpack helloworld。设置你的项目名称，其它可以默认回车。
  - 具体项目目录如下：

```
[QindeMacBook-Pro:~ qinliu$ cd helloworld ]
[QindeMacBook-Pro:helloworld qinliu$ ls ]
README.md      index.html      package.json    test
build           node_modules    src
config          package-lock.json  static
[QindeMacBook-Pro:helloworld qinliu$ npm run dev ]

> helloworld@1.0.0 dev /Users/qinliu/helloworld
> webpack-dev-server --inline --progress --config build/webpack.dev.conf.js

12% building modules 23/31 modules 8 active ...!/Users/qinliu/helloworld/src/App.vue{ parser: "babylon"
} is deprecated; we now treat it as { parser: "babel" }.
95% emitting

DONE Compiled successfully in 3442ms

i Your application is running here: http://localhost:8080
```

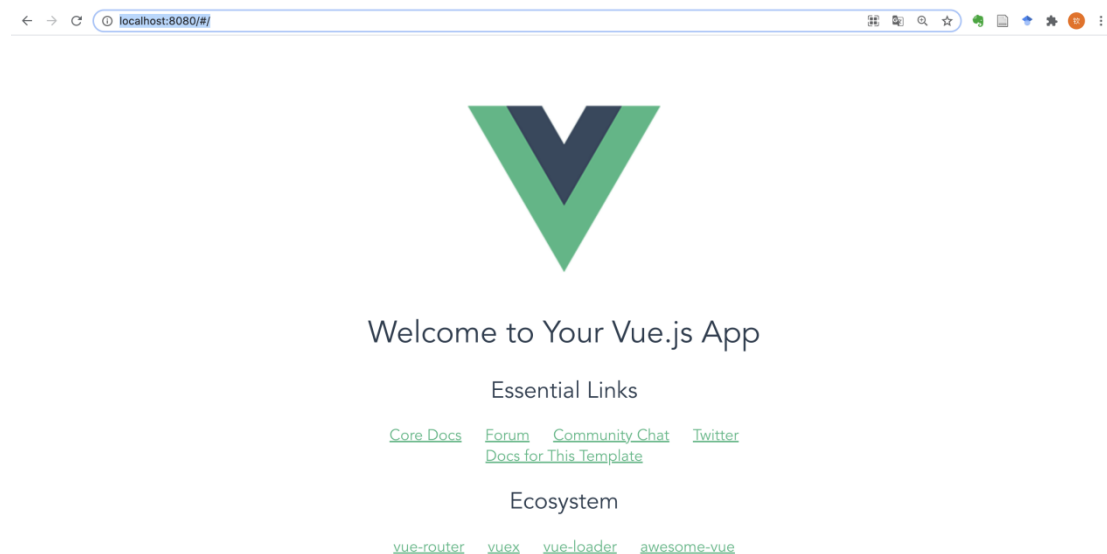
下午 4:20:34

# Vue.js安装配置

- 安装依赖
  - 运行`npm install` 或者 `cnpm install`。安装完成之后，会在我们的项目目录文件夹中多出一个 `node_modules` 文件夹，这里边就是我们项目需要的依赖包资源。

# Vue.js安装配置

- 运行项目
  - 进入你项目的目录，运行`npm run dev`，启动本地服务器。
  - 通过浏览器访问`http://localhost:8080/#/` 网址。



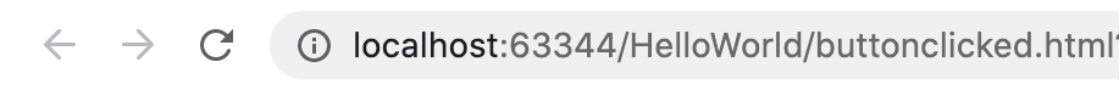
# Vue.js基本使用 – 计数器

- 简单计数器案例： 点击按钮，显示计数器次数。

```
<!DOCTYPE html>

<head>
  <meta charset="utf-8">
  <title>Vue Example</title>
  <script src="https://cdn.staticfile.org/vue/2.2.2/vue.min.js"></script>
</head>
<body>
<div id="app">
  <button v-on:click="increment">plus 1</button>
  <p>This button is clicked {{ count }} times。 </p>
</div>

<script>
const app=new Vue({
  el:'#app',
  // state
  data:{
    count: 0
  },
  // actions
  methods: {
    increment:function () {
      this.count++
    }
  }
})
</script>
</body>
</html>
```



plus 1

This button is clicked 3 times。

# Vue.js基本使用 – 网站列表

- 网站列表案例：客户端ViewModel异步请求Model数据。

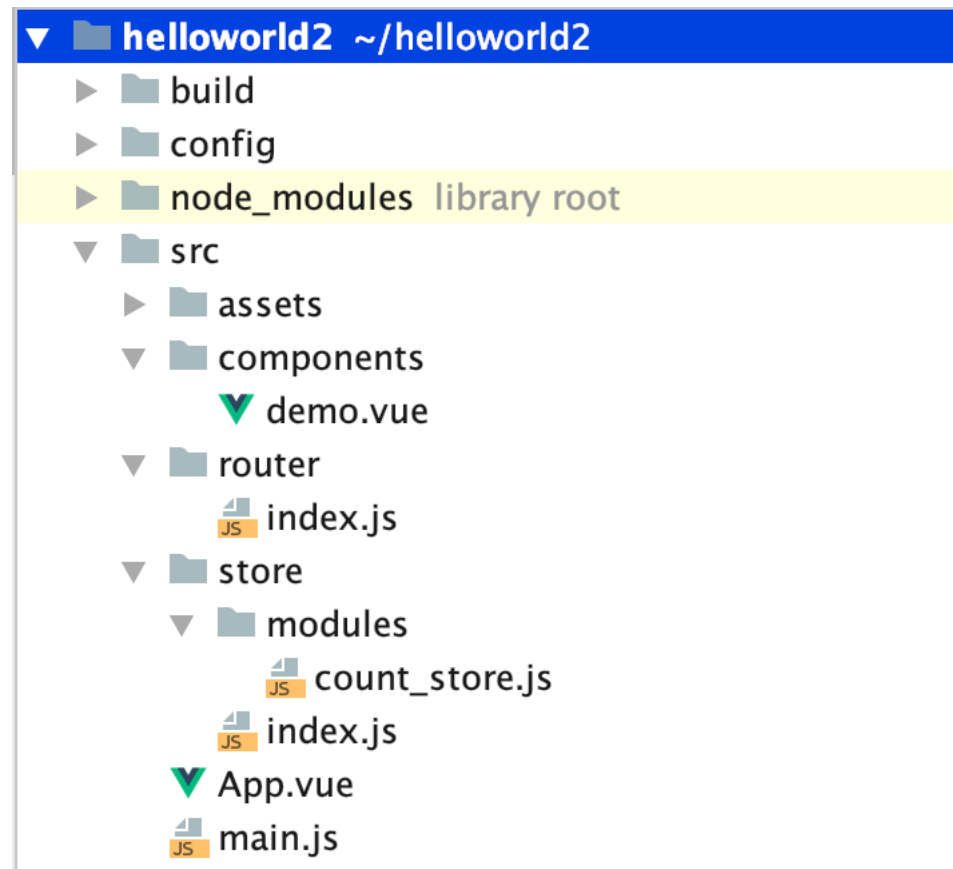
```
<div id="app">
  <h1>网站列表</h1>
  <div
    v-for="site in info"
  >
    {{ site.name }}
  </div>
</div>
<script type = "text/javascript">
new Vue({
  el: '#app',
  data () {
    return {
      info: null
    }
  },
  mounted () {
    axios
      .get('https://www.runoob.com/try/ajax/json_demo.json')
      .then(response => (this.info = response.data.sites))
      .catch(function (error) { // 请求失败处理
        console.log(error);
      });
  }
})
</script>
```

## 网站列表

Google  
Runoob  
Taobao

# Vue.js基本使用 – Vuex实现计数器

- 项目主要目录文件
  - App.vue：根页面级组件
  - main.js：入口文件
  - src/router/index.js：实现路由
  - store目录下有index.js文件和modules文件夹。把相关的状态（store）分离到modules目录下，再在index.js中引入
  - component目录：界面组件





# Vue.js基本使用 – Vuex实现计数器

- App.vue

```
<template>
  <div id="app">
    <Demo></Demo>
  </div>
</template>

<script>
import Demo from './components/demo.vue'

export default {
  components: {
    Demo
  }
}
</script>
```

# Vue.js基本使用 – Vuex实现计数器

- main.js引入router和store

```
import Vue from 'vue'  
import App from './App'  
import router from './router'  
import store from './store'
```

```
Vue.config.productionTip = false
```

```
new Vue({  
  el: '#app',  
  router,  
  store,  
  components: { App },  
  template: '<App/>'  
})
```

# Vue.js基本使用 – Vuex实现计数器

- store目录index.js： 这里定义了一个的count模块， 来自modules目录的count\_store.js。

```
import Vue from 'vue'  
import Vuex from 'vuex'  
import countStore from './modules/count_store.js'
```

```
Vue.use(Vuex)
```

```
export default new Vuex.Store({  
  modules: {  
    count: countStore  
  }  
})
```

# Vue.js基本使用 – Vuex实现计数器

- count\_store.js

```
export default{  
  state: {  
    count: 1  
  },  
  mutations: {  
    increment (state) {  
      state.count++  
    },  
    decrement (state) {  
      state.count--  
    }  
  }  
}
```

# Vue.js基本使用 – Vuex实现计数器

- demo.vue

- 定义一个<p>标签，MVVM双向绑定了{{ \$store.state.count.count }}值

- 两个按钮，点击之后的回调分别执行"\$store.commit('increment')"和

- "\$store.commit('decrement')"提交变更

```
<template>
  <div>
    <p>{{ $store.state.count.count }}</p>
    <div>
      <button @click="$store.commit('increment')">+</button>
      <button @click="$store.commit('decrement')">-</button>
    </div>
  </div>
</template>

<script>
export default {
  name: 'demo'
}
</script>

<style scoped>

</style>
```

# Vue.js基本使用 – Vuex实现计数器

- 使用npm run dev命令运行项目



3



# 目录

---

1. Web前端

2. Vue.js

3. Node.js

- 介绍
- 安装配置
- 基本使用

# 本节概述和学习目标

---

- 本节介绍Node.js的特性、Hello World项目和简单的基于Node.js的服务器案例。
- 学会Node.js的安装和基本使用。



# Node.js介绍

- Node.js 就是能够在服务端运行的JavaScript开放源代码、跨平台JavaScript环境。
  - 采用Google开发的V8运行代码。
  - 使用事件驱动、非阻塞和异步输入输出模型等技术来提高性能，可优化应用程序的传输量和规模。
  - 大部分基本模块都用JavaScript语言编写。
  - Node.js的出现之后，使得JavaScript应用于服务端编程成为可能。
    - Node.js的一系列内置模块，使得程序可以脱离Apache HTTP Server或IIS，作为独立服务器运行。
  - Npm：Node.js附带的包管理器。

# Node.js安装配置

- 安装Node.js和npm
  - 访问<https://nodejs.org/en/>官方网址，选择对应操作系统的安装软件。
  - npm:为Nodejs下的包管理器。一般安装Node.js后，会自动安装npm。

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#).

#BlackLivesMatter

The 2021 Node.js User Survey is open now

Download for macOS (x64)

14.15.4 LTS

Recommended For Most Users

15.8.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

```
[QindeMacBook-Pro:~ qinliu$ node -v  
v10.11.0  
[QindeMacBook-Pro:~ qinliu$ npm -v  
6.14.3
```

# Node.js基本使用 – Hello World

- Hello World
  - 在任何文本编辑器上输入下列代码，然后保存为hello.js。
  - 第一行总是写上‘use strict’;是因为以严格模式运行JavaScript代码，避免各种潜在陷阱。
    - ‘use strict’;
    - console.log(‘Hello, World.’);
  - 进入hello.js所在目录，执行node hello.js，控制台就会得到如下输出。
    - Hello, world.

# Node.js基本使用 – Hello World Server

- Hello World Server

- 服务器端使用require指令来载入CommonJS 模块。
- Node.js 中自带 http 的模块，在代码中请求它并返回它的实例化对象赋给一个本地变量，通过这个本地变量可以调用http 模块所提供的公共方法。

```
var http = require('http');

http.createServer(function (request, response) {

  // 发送 HTTP 头部
  // HTTP 状态值: 200 : OK
  // 内容类型: text/plain
  response.writeHead(200, {'Content-Type': 'text/plain'});

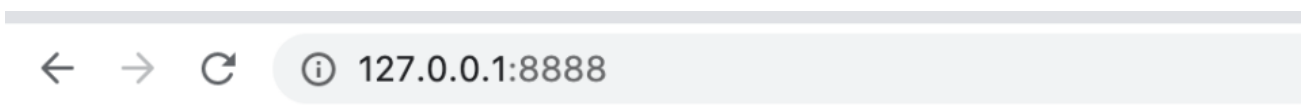
  // 发送响应数据 "Hello World"
  response.end('Hello World\n');
}).listen(8888);

// 终端打印如下信息
```

# Node.js基本使用 – Hello World Server

- Hello World Server
  - 运行：node server.js 启动服务器。通过浏览器访问<http://127.0.0.1:8888/> 网址。

```
QindeMacBook-Pro:nodehelloworld qinliu$ node server.js
Server running at http://127.0.0.1:8888/
```



Hello World

# Node.js基本使用 – 网站列表服务器

- 网站列表服务器
  - Vue.js章节中访问网站列表是通过第三方服务器得到的数据。如果我们想从自己的Node.js服务器得到数据，我们只需要向我们自己的服务地址发出REST请求。

# Node.js基本使用 – 网站列表服务器

- 网站列表服务器

- 修改之前创建第一个Vue.js项目的HelloWorld项目中router文件夹下的index.js。

```
import Vue from 'vue'
import Router from 'vue-router'
import WebSites from '@/components/WebSites'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'OurWebSites',
      component: WebSites
    }
  ]
})
```

# Node.js基本使用 – 网站列表服务器

- 网站列表服务器
  - 在component文件夹下创建我们新的Vue component: WebSites.vue。



# Node.js基本使用 – 网站列表服务器

```
<template>
```

```
<div >
```

```
<h1>网站列表</h1>
```

```
<div v-for="site in info" v-bind:key="site">{{ site.name }}</div>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import axios from 'axios'
```

```
export default {
```

```
  name: 'OurWebSites',
```

```
  data () {
```

```
    return {
```

```
      info: null
```

```
    }
```

```
  },
```

```
  mounted () {
```

```
    axios
```

```
    .get('http://localhost:8081/getWebSites')
```

```
    .then(response => (this.info = response.data.sites))
```

```
    .catch(function (error) { // 请求失败处理
```

```
      console.log(error)
```

```
    })
```

```
  }
```

```
}
```

```
</script>
```

```
<style scoped>
```

```
</style>
```

# Node.js基本使用 – 网站列表服务器

- Express简介
  - Express 是一种保持最低程度规模的灵活Node.js Web应用程序框架。
  - Express框架核心特性：
    - 设置中间件来响应HTTP请求
    - 定义了路由表用于执行不同的HTTP请求动作
    - 可以通过向模板传递参数来动态渲染HTML 页面

# Node.js基本使用 – 网站列表服务器

- 网站列表服务器
  - 服务器根据URL映射到具体的处理函数，通过req得到请求，经过业务逻辑处理后，通过res设置回复。通过JSON.stringify()将数据转成包含 JSON 文本的字符串，并返回。
  - \_\_dirname获得当前目录名称。

# Node.js基本使用 – 网站列表服务器

```
var express = require('express');
var app = express();
var fs = require("fs");
var cors = require('cors')

//解决跨域问题
app.use(cors({
  credentials: true,
  origin: '*',
}))

//映射URL和处理函数
app.get('/getWebSites', function (req, res) {
  fs.readFile( __dirname + "/" + "data.json", 'utf8', function (err, data) {
    data = JSON.parse( data );
    console.log( data);
    res.end( JSON.stringify(data));
  });
})

//设置监听端口，启动服务器
var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("应用实例， 访问地址为 http://%s:%s", host, port)

})
```

# Node.js基本使用 – 网站列表服务器

- 网站列表服务器

- 服务器是从本地一个JSON文件中读取的数据，data.json文件如下：

```
{
  "sites":[
    { "name":"Sina", "info":[ "News", "Blog" ] },
    { "name":"Baidu", "info":[ "Search", "Netdisk" ] },
    { "name":"Taobao", "info":[ "Shopping" ] }
  ]
}
```

# Node.js基本使用 – 网站列表服务器

- 网站列表服务器

- 最终显示效果



# 思考题

1. 前端路由和后端路由的区别是什么，它们各自的优缺点是什么？

# 本章总结

- 本章对Web前端的基础内容HTML、CSS和JS进行了简单的介绍。
- 介绍了Vue.js，教会大家基础的使用方法，并举了几个编程实例。
- 介绍了Node.js的基本使用，并介绍了Node.js编写服务器的案例。