**专题1javaScript进阶攻略**

**一、课程导学**

**高级前端进阶导学**

**1.1.1前端高级课程体系梳理**

前端发展方向：

自动化 可视化（echart、d3、three）

从内部来说我们如何应对未来：

1.编辑思想 2.学会学习源码 3.关注 4.提升角度（从软件工程来考虑-->人月神话）

从外部来说我们需要哪引起技能：

基础语法：javascript typeScript

上层框架：vue react

工具链： webpack vue-cli git 测试

全栈： node 服务器 数据库

新兴技术: fluter

技术方案：最佳实践 ssr 部署

**1.1.2高效源码学习技巧**

源码阅读步骤：

单文件型：理架构-找核心类-按照顺序依次找下去-理解方法层次，寻找关键内容

模块化型：看packjson-找到入口文件的export-依次查找对应文件export

源码阅读通常的共性：

方法一般分为三部分-参数处理部分，核心操作部分，最终输出部分

一般都会有一个init方法

通常都会有分模块，然后合并的操作

通常都会有一系列工具函数

所以我们应该注意什么内容：

值得借鉴的写法

体现思想与设计模式

让你实现与源码上实现的对比

架构的和模块的划分

复习：

享元模式：减少对象数量，提取外部数据和内部数据 -- 抽取共用的代码

装饰器模式：不改变原来的函数，新增功能

 // 需求：在a方法要输出3  改动源码是大忌

function a(){  console.log(2)  }

// 方法一：直接写

function b(){ a()  console.log(3) }

// 方法二：重写

var \_a = a()

a = function(){  \_a()  console.log(3) }

// 重写array push()

var Arrpro = Array.prototype

var arrob = object.create(Array.prototype)

arrob.push = function(){  Arrpro.push()  render() }

**1.1.3单元测试**

**单元测试技术：**

测试框架（执行方法）：Qunit jasmine mocha intern Jest

断言库（检查结果）：chai should expect assert

mock库（数据、外部依赖）：sinon

test runner(提供执行环境，管理执行流程):karma buster.js

覆盖率工具:lastanbul

**前端单元测试：mocha+chai+sinon**

**mocha：**

1.安装

npm install [--global/--save-dev] mocha

2.基本语法(https://mochajs.org/)

常用语法

describe:对应测试的模块/函数

it:对应test case

only/skip：只运行/忽略一个或者一组用例

异步代码

callback

promise

async/await

hooke

before/after

beforeEach/afterEach:每个用列都会执行一遍

**chai：**

1.安装

npm install [--global/--save-dev] chai

2.基本语法

BDD( <http://chaijs.com/api/bdd> ):

1.expect

chai.expect -- > expect( approve({amount:3}) ).to.eql(“通过”)

2.should

chai.should() -->approve( { amount:3 } ).should.eql(“通过”)

TDD( http://chaijs.com/api/assert/ )

1.assert

chai.assert -- > assert.equal( approve({amount:3}).“通过”)

**sinon**

1.安装

npm install [--global/--save-dev] sinon

2.基本语法

Spies(spy)：提供了函数调用的信息，但不会改变其行为

Stubs：类似Spies，但是是完全替换目标函数

Mocks：组合spies与stubs，可替换一个完整的对象

1. **Karma**

1.安装

npm install -g karma

npm install -g karma-cli

2.执行

karma init 生成配置项

Karma start 启动karma

1. **设计模式jQuery源码分析**

**重点： 插件的设计思想 callbacks 单一职责 data数据缓存+事件系统**

**核心功能**

**2.1.1jQuery整体架构-核心功能函数揭秘**

**Sizzle**

**2.2.1选择器**

**一、常用jQuery选择器接口**

1.传入对象

$(this) $(document) 把传入的对象包装成jQuery对象

2.传入函数

$(function(){}) 这个是在页面DOM文档加载完成后加载执行的，

等效于在DOM加载完毕后执行了$(document).ready()方法

3.传入字符串（查询）

$('.box')查询DOM节点包装成jQuery对象

4.传入HTML字符串（创建）

$('<div>')创建DOM节点包装成jQuery对象

5. 空

$()创建jQuery对象

二、问题分析

1.如何把创建的DOM节点包装成jQuery对象

context.createElement创建DOM节点存储在数组中,调用merge方法把数组中

存储的DOM节点的成员添加到jQuery实例对象上

2.jQuery实例对象length属性的作用

存储DOM节点的数组对象平滑地添加到jQuery实例对象上

3.merge方法的应用场景有哪些

合并数组

把数组成员合并在length属性的对象上

4.$(document).ready()与$(function(){})的关系

$(document).ready()是对document.DOMContentLoaded事件封装

$(function(){})每次调用$()传入的参数会收集在readyList数组中，

当document.Contentedness事件触发依次执行readyList中收集的处理函数

三、课后作业（阅读源码，了解它们如何进行处理）

1.根据 [/^<(\w+)\s\*\/?>(?:<\/\1>|)$/](https://www.cnblogs.com/yina-526/p/11138199.html) ，分析其正则执行过程中如何进行过滤？包含哪些执行步骤？

var rejectExp = /^<(\w+)\s\*\/?>(?:<\/\1>|)$/;  
var parse = rejectExp.exec('<a>')  
console.log(parse[1]);//a  
分析如何过滤:  
 (1)正则表达式都包含在//中  
 (2)^< 表示字符串是以'<'开头的  
 (3)(\w+)其中\w为匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9\_]'。  
 (4)\s\* 匹配任何空白字符，包括空格、制表符、换页符等等。  
 (5)\/?>表示匹配"/"零次或一次,后面加">"  
 (6)(?:<\/\1>|)表示后面接'<\/\1>'或者不接任何内容  
 (7)<\/\1> 其中‘\/’表示匹配‘/’符号，‘\1’指前面的‘\w+’  
 (8)$结束  
包含的执行步骤：  
 (1)标签以 '<'开头  
 (2)在开始标签内进行匹配，包括  
 任意单词字符+  
 0个或多个空白字符+  
 0个或1个'\'+  
 '>'  
 +或不+'</'+上一个标签内的字符

1. 当selector传来一个函数时，怎么进行处理？

**回调对象设计**

**2.3.1 Callbacks入门&原理分析**

$.callbacks用于管理函数队列

通过add添加处理函数到队列当中，通过fire去执行这些处理函数

// 通过调用$.callbakcs,获取到一个callbacks实列

        var cb = $.callbacks( )

        // add向内部队列添加函数

        cb.add( function(){   console.log('add one') })

        // fire依次执行队列里的函数

        cb.fire() // add one

**$.callbacks通过字符串参数的形式，支持4种的特定功能:**

1.once

函数队列只执行一次

2.unique

往内部队列添加的函数保持唯一，不能重复添加

3.stopOnFalse

内部队列里的函数是依次执行的，当某个函数的返回值是false时，停止继续剩下的函数

4.memory

当函数队列fire一次过后，内部会记录当前fire的参数。当下次调用add的时候，会把记录的参数传递给新添加的函数并立即执行这个新添加的函数

**异步回调解决方案**

**2.4.1 延时对象-Deferred概念&源码剖析**

Deferred是Promise/A+规范的具体实现

**Deferred API(https://www.jquery123.com/category/deferred-object/):**

1.jQuery.Deferred()

一个构造函数，返回一个链式实用对象方法注册多个回调，回调队列，调用回调队列，并转达任何同步或异步函数的成功或失败状态

2.deferred.done()

当Deferred（延迟）对象解决时，调用添加处理程序。

3.deferred.fail()

当Deferred（延迟）对象拒绝时，调用添加的处理程序。

4.deferred.progress()

当Deferred（延迟）对象生成进度通知时，调用添加处理程序。

5.jQuery.when()

提供一种方法来执行一个或多个对象的回调函数， Deferred(延迟)对象通常表示异步事件

6..promise()

返回一个 Promise 对象，用来观察当某种类型的所有行动绑定到集合，排队与否还是已经完成。

**事件绑定**  
**2.5.1 事件绑定-bind/delegate/on**  
**2.5.2 事件绑定-体系结构/委托设计**

jQuery事件处理处理机制解决问题：

1.不同浏览器下的事件兼容

2.一个事件类型添加多个事件函数，一次添加多个事件类型的处理函数

3.简洁的定义方式

4.允许自定义事件



**2.5.3 事件绑定-自定义设计&模拟事件**

**自定义事件**

在实际开发中，或许原生的javaScript事件或者jQuery事件不能满足开发的要求，这时候就需要我们自己编写相应的事件

添加自定义事件：

通过jQuery的bind( )或者on( )方法添加自定义事件如下代码，自定义一个createEvent事件

$('#Id').on('createEvent',function(){

            console.log('createEvent')

        })

触发自定义事件：

通过jQuery的trigger( )方法触发自定义事件createEvent

$('#Id').trigger('createEvent')

**模拟事件**

trigger( )方法触发事件后，会执行浏览器默认行为

$('input').trigger('focus')

以上代码不仅会触发input元素绑定的focus事件，还会使得input元素本身得到焦点(这是浏览器的默认行为)

如果只想触发绑定的focus事件，而不想执行浏览器的默认行为，可以使用另一种类似的方法triggerHandler( )方法

**DOM操作 & 样式操作**

**2.6.1 DOM操作方法-html,text,css**

html( ) 用于读取和修改元素的HTML标签

text( ) 用来读取或修改元素的纯文本内容

css( ) 设置或返回被选元素的一个或多个样式属性

text( ),html( ),csss( )都是通过jQuery.access提供底层支持

jQuery.access( )是一个多功能方法，作为set和get值来使用

**2.6.2 - 2.6.3 DOM操作核心 & CSS / addClass原理分析**

addClass( )

对元素的样式操作，底层的实现修改元素的className值

实现的功能:

增加一条样式规则 addClass(“box”)

增加多条样式规则 addClass(“box item”)

**高级进阶-动画队列**

**2.7.1 动画队列 / 动画设计 / 动画实现**

**1.动画队列 （Queue队列）**

Quenu队列，如同data数据缓存与Deferred异步模型一样,都是jQuery库内部实现的基础设施

Quenu队列是animate动画依赖的基础设施，整个jQuery中队列仅供给动画使用

Quenu数据存储重度依赖data函数

动画会调用队列，队列会调data数据接口还保存队列里面的动画数据

**2.数据缓存接口.data( )**

****

**3.数据缓存&队列&动画**



**4.出列 - 入列**



1. **函数式编程underscore源码分析**

**重点：迭代器设计 模板函数 数据流的处理**

**Underscore设计篇**

**3.1.1函数式编程思想概述**

 纯函数：

        定义：对于相同的输入，永远会得到相同的输出，而且没有任何可观察的副作用，也不依赖外部

         环境的状态

        在JavaScript中，对于数组的操作有些是纯的，有些是不纯的

  var arr = [1,2,3,4,5]

    // 纯函数

    arr.slice(0.3) // [1,2,3]   arr.slice(0.3) // [1,2,3]

    // 非纯函数

    arr.splice(0,3) // [1,2,3] arr.splice(0,3) // [4,5]

函数式编辑为何排斥不纯的函数？

         非纯函数中，函数的行为需要由外部的系统环境决定。也就是说此函数行为不仅取决于输入的参数age，还取决于一个外部的变量 timeOfLife，这种对于外部状态的依赖，是造成系统复杂性大大提高的主要原因

var timeOfLife = 20

   // 纯函数

   function test(age){ return age > 20 }

   // 非纯函数

   function test(age){ return age > timeOfLife }

 函数柯里化：

        定义： 向函数传递一部分参数来调用它，让它返回一个函数去处理剩下的参数

             事件上柯里化是一种，‘预加载‘函数的方法，通过传递较少的参数，得到一个已经记住了这些参数的新函数，某种意义上讲，这是一种对参数的’缓存‘，是一种非常高效的编写函数的方法

var timeOfLife = 20

    function test(timeOfLife){

       return function(age){ return age > timeOfLife }

    }

    var testing = test(20)

    testing(18) // false

 函数组合：

        为避免写出不优雅的包菜式代码h(g(f(x))),我们需要用到函数组合

        我们定义的compose就像双面胶一样，可以把任何两个纯函数结合到一起，也可以扩展出组合n个函数的n面胶

  // 两个函数的组合

   var compose = function(f,g){ return function(x){  return f(g(x)) } }

   var mult = function(x){ return x\*5 }

   var add = function(x){ return x+1  }

   compose(mult,add)(2) // 5

命令代码：

        通过编写一条又一条指令，让计算机执行一些动作，其中一般会涉及许多繁杂的细节声明式代码

        通过写表达式的方式，声明我们想干什么，而不是通过一步一步的指示

        声明式代码，是函数式编程的一个明显好处一一编写、优化代码时能更专注

  // 命令式

    var rest = [] , arr = [4,9,16,25,4,16]

    for(var i = 0;i<arr.length;i++){

       if(rest.indexOf(arr[i]) === -1){  rest.push(arr[i]) }

    }

    // 声明式

    var rest = arr.map(Math.sqrt)

 总结：

        函数对于外部状态的依赖，是造成系统复杂性大大提高换主要原因

        代码书写中让函数尽可能地纯净

        函数式编程不是万能，它与opp一样，只是一种编程范式

        为降低软件复杂度，opp的方式是靠良好的封装、继承、多态以及接口定义。函数式编程则是靠纯函数以及它们的组合、柯里化等技术

**3.1.2underscore整体结构&面向对象**

 \_对象：

underscore有下划线的意思，所以underscore通过一个下划线变量\_来标识自身

\_是一个函数对象，之后所有的api都会被挂载到这个对象上，如\_.each , \_.map

 \_( )：

虽然underscore推崇函数式编程，但也支持面向对象风格的函数调用，仅需要通过\_()来包裹对象即可

\_([2,3,4])会创建一个新的underscore对象（从而能够调用underscore提供的方法）, 并在this.\_wrapped中存储传入的数据

mixin：

mixin模式是增加代码复用度的一个广泛使用的设计模式

链接式调用：

开启链接式调用方法：\_.chain( )

关启链接式调用方法：\_.value( )

**3.1.3undefined的处理/函数式编程(回调)iteratee设计**

void 运算符通常只用于获取 undefined的原始值

void 0和undefined是等价，因此可以用来判断undefined

**3.1.4 rest参数/underscore创建对象方式**

\_ . restArguments：

underscore的官方实现，它暴露了一个\_.restArguments方法，通过给该方法传递一个函数，能够使得函数支持rest参数

\_ . baseCreate(创建对象)：

underscore利用baseCreate创建对象时，会先检查当前环境是否已支持Object.create，如不支持，会创建一个简易的polyfill

**Underscore辅助功能篇**

**3.2.1Map-reduce真值检测函数**

**3.2.2 indexOf lastIndexOf条件查询-背后索引查询器实现**

\_.createdIndexFinder:

underscore中通过内置的工厂函数createIndexFinder来创建一个索引查询器

\_.indexOf及\_.lastIndexOf 正是由该函数所创建的

createdIndexFinder( dir ,predicateFind , sortedIndex )接受3个参数

dir查询方向 \_.indexOf是正向查询 \_.lastIndexOf是反向查询

**predicateFind真值检测函数，该函数只有在查询元素不是数字(NaN)时才会使用**

sortedIndex有序数组的索引获得函数。如果设置了该参数，将假定数组已有序，从而更加高效地通过针对有序数组的查询函数（比如二分查找等）来优化查询性能

**3.2.3乱序数组-洗牌算法**

洗牌算法：

算法思路在宏观上可以概括为：将集合视为牌堆，不停地从牌堆中抽牌构成新的牌堆，直至新牌堆的牌数到达预设数量

从underscore 1.9版本开始，洗牌算法通过\_.sample实现

\_.sample(array,n):从array随机取出n个样本

underscore中的抽样函数正基于洗牌算法

**Underscore数组篇**

**3.3.1 underscore数组定位&摊平数组**

数组定位

\_.initial( array,n ) : 获得array的除了最后n个元素以外的元素

\_.rest( array ,n ) : 返回array中除了前n个元素外的所有元素

摊平数组

.flatten(array,shallow) : 摊平array，通过shallow指明是深度摊平还是浅摊平

**3.3.2 数组运算uniq去重函数 原生对象扩展Underscore函数篇**

\_.unique: 重写数组去重函数

\_.computed: 去掉数组中所有的假值

\_.range: 返回某一个范围内的数值组成的数组

**Underscore函数篇**

**3.4.1偏函数(partial)-缓存(memoize)使用场景源码分析**

\_.partial : 反映了新函数是原函数的一部分

应用一个函数，填充在任意个数的参数，不改变其动态this值，和bind方法很相近

\_.memoize : memoize方法可以缓存某函数的计算结果（我搞不懂）

**3.4.2延时执行函数- compose-escape字符串逃逸**

字符串逃逸：防止用户注入代码，造成xss攻击，所以使用正则把字符转为实体名称

**3.4.3throttle与debounce 概念解析源码实现**

throttle节流函数：在一定时间内，调用了n次，只会执行一次

debounce防抖函数：在一定时间内，调用了n次，只会根据最后一次调用重新计算时间

**Underscore对象篇**

**3.5.1 属性操作 & createAssigner**

\_.has(object,key): 检测object对象是否包含给定的key属性 == Object.hasOwnProperty(key)

\_.keys(object): 获取object对象所有的属性名称 == Object.keys(obj)(但不兼容IE)

\_.invert(object)：返回一个object副本，使其键(keys)和值(values)对象

createAssigner( ): 对象属性的扩展 == Object.assign

**3.5.2 deepclone 深度克隆 & pick**

**3.5.3template模板引擎设计**

1. **模块化编程自研模块加载器**

**模块加载器设计概要**

**4.1.1 模块系统概述-自定义模块规范-书写约定**

**4.1.2 加载器结构设计导论**

****

模块部分：每个模块创建都先初始化数据，存储在缓存对象中

数据初始化：加载器中设计了一个名为module的构造函数，每个模块都是此构造函数实例对象

模块存储：加载器中设计了一个名为cache缓存对象，每个文件（模块）都会存储在cache对象中

具体存储方式：{ 当前模块绝对路径：new Module( ) }

注意：当前模块的绝对路径是通过资源部分，资源定位方法实现的

资源部分: 资源定位、依赖管理是加载器设计两大核心

资源定位：加载器中设计了一个resolve()的方法把模块名解析成绝对路径格式

模块名的获取：startUp.use( ['a.js','b.js ')

startUp.use()加载器启动方法，启动时会去调用传入数组列表中的模块

动态加载script文件：通过加载器resolve()方法把模块名解析成绝对路径格式

动态创建script document.create('script') src指向当前模块绝对路径地址

加载文件同时，模块加载器解析当前模块所依赖的模块以数组的形式存储

依赖管理：已知当前模块在cache中的形态，{ '当前模块绝对路径' : new Module( ) }

换算： {'当前模块绝对路径' : { url:"当前模块绝对路径" , deps:[ ] } }

deps存储当前模块的依赖列表，依赖列表通过动态加载script文件正则解析获取

重点：解析依赖 -》 获取依赖模块绝对路径地址 -》 动态加载 -》 提取依赖 -》解析依赖

递归方式加载所有模块，直至模块全部加载完毕

**模块加载器设计核心**

**4.2.1 Module构造器设计 模块数据初始化**

****

****

**4.2.2 模块资源定位-异步加载**

****

**4.2.3 依赖解析-依赖管理处理方案**

****

**4.2.4 模块(路径)短名称配置解决方案**

**专题2单页应用开发**

**一、MV\* & 组件化开发 Vue专题**

**Vue基础**

**1.1.1 初识vue**

**1.1.2 模块语法,常用指令,事件绑定**

v-cloak：

解决初始化慢导致页面闪动的最佳实践，它会在vue实例结束编译时从绑定的html元素上移除，在简单项目中，使用 v-cloak 指令是解决屏幕闪动的好方法

修饰符：

1.事件修饰符   .stop .prevent  .capture   .self   .once   .passive

2.系统修饰符    .ctrl   .alt   .shift   .meta   .left   .right    .middle

3.按键修饰符 .enter  .tab .delete (捕获“删除”和“退格”键)   .esc  .space

             .up   .down  .left   .right

**1.1.3 计算属性及watch**

 计算属性的作用及使用

   使用场景：一个数据受多个数据影响

        计算属性(computed)与method方法的区别

                计算属性

                    1、包括getter和setter

                    2、基于依赖缓存

                    3、依赖数据改变 =》 触发setter和getter =》 计算属性改变

                    4、可以依赖其他计算属性

                    5、可以依赖其他实例数据

                methods方法

                    本质是实例上定义的普通函数，组件得新渲染，就会调用

            如何选择

                    是否需要缓存，比如遍历大数组和做大量计算，使用计算属性

    watch的作用及使用

            使用场景：一个数据影响多个数据

**1.1.4 条件渲染和列表渲染**

 列表渲染

 1.维护状态( :key )

   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

         跟虚拟DOM diff算法相关

2.数组更新检测

Vue将被侦听的数组的变异方法进行了包裹，所以它们也将会触发视图更新，被包裹过的方法有：push( )、pop( )、shift( )、unshift( )、splice( )、sort( )、reverse( )

两种情况不能监听：当利用索引直接设置一个数组项时

当修改数组的长度时

3.注意事项

不推荐同时使用v-if 和 v-for 同时使用，v-for具有比v-if更高级的优化级

**1.1.5 表单处理**

常用修饰符

v-model.lazy 从input事件中转变为在change事件中同步数据

v-model.number 可以将输入转换为Number类型

V-model.trim 可以自动过滤输入的首尾空格

**1.1.6 组件**

组件基础

1、为什么要使用组件

组件是对数据和方法的简单封装，可以提高html代码的复用

2、组件使用

组件命名：kebab-case(短横线分隔命名) PascalCase(首字大写命名)

组件注册:

 // 全局注册

  Vue.component('my-component',{ template:"<div>这是全局组件</div>" })

  // 局部注册

  const app = new Vue({

      el:'#app',

      components:{

       'com-a':{

             template:'<div>这是局部组件</div>',

              data:function(){ return {} } }

        }

   })

3、自定义v-model事件 接收一个value属性 在有新的value时触发input事件

4、.sync修改符 父组件监听自定义事件按需更新数据

Slot分发内容

1、单个slot 在父组件模板里，插入在子组件标签内的所有内容将替代子组件的<slot>标签及它的内容

2、具名slot 给<slot>元素指定一个name后，可以分发多个内容 name=“ke” v-slot:ke

具名slot可以与单个slot共存

1. 作用域slot 作用域slot是一种特殊的slot,使用一个可以复用的模板替换已渲染元素

<slot msg=“数据”></slot>

<template v-slot:default =“props”>{{props.msg}}</template>

注： v-slot只能用于template中

单文件组件及自定义组件和动态组件及异步组件

1、单文件组件

一个后缀名为.vue的文件，使用.vue文件需要先安装vue-loader、vue-template-compiler等加载器并做webpack配置。因为要使用ES6语法，还需要安装配置babel和babel-loader等编译器

vue-loader：将文件名后缀为.vue的文件抽离出html、css、js交给其他的loader处理，重点在抽取、分离

vue-template-compiler：将 Vue 2.0 模板预编译为渲染函数（template => ast => render），以避免运行时编译开销和 CSP （内容安全策略是一个附加的安全层，用于帮助检测和缓解某些类型的攻击，包括跨站脚本 (XSS) 和数据注入等攻击）限制

Babel：是一个 JavaScript 编译器,主要用于将 ECMAScript 2015+ 版本的代码转换为向后兼容的 JavaScript 语法，以便能够运行在当前和旧版本的浏览器或其他环境中

bable-loader：是Babel与webpack协同工作的模块（使webpck通过babel-laoder使用Babel）

@babel/corel:：是babel编译器的核心模块，将 js代码分析成 ast ，方便各个插件分析语法进行相应的处理。有些新语法在低版本 js 中是不存在的，如箭头函数，rest 参数，函数默认值等，这种语言层面的不兼容只能通过将代码转为 ast，分析其语法后再转为低版本 js

1. 自定义组件(像element-ui一样)

创建一个文件夹，里面创建myComponent.vue和myComponents.js文件

在vue项目中的入口文件main.js中或者需要使用的文件中，进行导入，使用vue.use( )

1. 动态组件

Vue.js提供一个特殊的元素<component>用来动态地挂载不同的组件，使用is特性来选择要挂载的组件

1. 异步组件

Vue.js允许将组件定义为一个工厂函数，动态地解析组件。vue只在组件需要渲染时触发工厂函数，并且把结果缓存起来，用于后面的再次渲染

**1.1.7 渲染函数及JSX语法**

渲染函数

1. 渲染函数(render函数)

通过createElement函数来创建虚拟DOM，这个虚拟DOM包含的信息会告诉Vue页面个需要渲染出什么元素

1. 渲染函数解析

render函数的返回值：VNode，也就是我们要渲染的节点

render函数的参数：createElement是render函数的参数，它本身也是了函数，并且有三个参数。createElement函数的返回值也是VNode虚拟节点

 render:function(createElement){

// HTML标签名 attribute 子级虚拟节点

    return createElement( 'h1', {} , [])

 }

认识节点、树以及虚拟DOM

节点 树 VNode VDOM的概念

// VNode对象源码

export interface VNode {

    tag:string; //当前节点的标签名

    data:VNodeData; // 当前节点的数据对象

    children:VNode[];//子节点，数组，也是VNode类型

    text:string;// 当前节点的文本，一般文本节点或注释节点会有该属性

    elm:Node; // 当前虚拟节点对应的真实的DOM节点

    ns:string;// 节点的namespace

    context:Vue; //编译作用域

    key:string | number; // 节点的key属性，用于作为节点的标识，有利于patch的优化

    componentOptions:VNodeComponentOptions; // 创建组件实例时会用到的选项信息

    componentInstance:Vue; // Vue实例

    parent:VNode; // 父节点对象，组件的占位节点

    raw:boolean; // 是否要将它转为原始的html

    isStatice:boolean; // 是否为静态

    isRootInsert:boolean; // 是否在根节点插入

    isComment:boolean; // 是否注释类型

}



JSX语法

javascript和XML结合的一种格式。最早React发明了JSX，利用HTML语法来创建虚拟DOM。JSX执行更快，因为它在编译为JavaScript代码后进行了优化。它是类型安全的，在编译过程中就能发现错误。使用JSX编写模板更加简单快速

// JSX语法

 import List from './list.vue'

new Vue({

    el:"#app",

    render:function(h) {

     return ( <List level={1}> <span>Hello world!</span></List> )

    }

 })

// 需要babel插件转译为普通的HTML

 <button class="btn btn-active"><span>jsx</span></button>

 React.createElement(

"button",

     { class: "btn btn-active"},

     React.createElement("span", null, "jsx")

 )

函数式组件

1. 使用场景(比较简单的组件、无状态)

[https://cn.vuejs.org/v2/guide/render-function.html#函数式组件](https://cn.vuejs.org/v2/guide/render-function.html" \l "%E5%87%BD%E6%95%B0%E5%BC%8F%E7%BB%84%E4%BB%B6)

2、可以把函数式组件想像成组件里的一个函数，入参是渲染上下文(render context)，返回值是渲染好的HTML

3、特性：Stateless(无状态)：组件自身是没有状态的

Instanceless(无实例)：组件自身没有实例，也就是没有this

 // 普通函数式组件

 Vue.component('my-function-button',{

    functional:true,

    render(createElement,context){

        return createElement('button',context.data,context.children)

    }

})

 // 模板函数式组件

`<template functional>

   <button class='btn btn-primary' v-bind='data.attrs' v-on='listeners'>

       <slot/>

   </button>

 </template>`

**1.1.8 过渡及动画**

请参考官网：<https://cn.vuejs.org/v2/guide/transitions.html>

**1.1.9 vue的生命周期**

钩子函数(Hook)在Vue中是一种事件劫持机制

它会比定义的事件更早进行执行处理，而且可以让你自己去配置

beforeCreate 实例创建前 el与data都为undefined

created 创建完毕状态 el为undefined data里面已经有数据

beforeMount 挂载前状态 el与data都有相对应的属性

mounted 挂载后状态 el与data都有相对应的属性

beforeUpdate 更新前状态 data里面的属性值改变

update 更新完成状态 data里面的属性值改变

beforeDestroy 销毁前状态 实例仍然可用

destroyed 销毁状态 vue实例指示的所有东西都会解绑

所有的事件监听器会被移除

所有的子实例也会被销

注：beforeMount和mounted是给vue实例对象添加$el成员，并且替换掉挂载的DOM元素

**1.1.10 自定义指令**

使用场景： 当你需要对普通 DOM 元素进行底层操作，这时候就会用到自定义指令

官网文档：<https://cn.vuejs.org/v2/guide/custom-directive.html>

**Vue进阶**

**1.2.1 初识vue-cli**

了解vue-cli vue-cli结构与webpack

官方网档：<https://cli.vuejs.org/zh/>

**1.2.2 组件化思想的深入**

**组件化**

组件创建：

1. 调用Vue.extend( )

var myComp = Vue.extend({ template:'<div>Vue.extend创建的组件</div>' })

1. <template>标签创建，需要加上id属于

<template id='myComp'><div>这是template标签构建的组件</div></template>

1. 标签创建

<script type="text/x-template" id="myComp">

<div>这是script标签构建的组件</div>

</script>

组件注册：

1. 调用Vue.extend(),创建全局注册组件

Vue.component('my-comp',myComp)

2、template及script标签构建的组件全局注册

Vue.component('my-comp',{ template:'#myComp' })

处理边界情况：

1. 访问根实例
2. 访问父级组件实例
3. 访问子组件实例或子元素
4. 依赖注入

**混入（mixin）**

1、官网文档：<https://cn.vuejs.org/v2/guide/mixins.html>

2、使用场景： 如果我们有大量的表格页面，仔细一扒拉你发现非常多的东西都是可以复用的例如分页，表格高度，加载方法， laoding声明等一大堆的东西

3、基础：全局混入 组件混入

注：请谨慎使用全局混入，因为它会影响每个单独创建的 Vue 实例 (包括第三方组件)。大多数情况下，只应当应用于自定义选项，就像上面示例一样。推荐将其作为插件发布，以避免重复应用混入

4、选项合并:

钩子函数：将合并为一个数组，因此都将被调用。另外，混入对象的钩子将在组件自身钩子之前调用。

值为对象的选项（methods、components 和 directives）：Object.assign()方式进行合并，发生冲突时以组件优先

**1.2.3 Vuex的使用及设计思想**

**Vuex是什么**

Vuex是一个专门为Vue.js应用程序开发的状态管理模式

它使用集中式存储管理应用的所有组件的状态，并以相应的规则，保证状态以一种可预测的方式发生变化

**Vuex的使用**

默认的五种基本对象：

state：存储状态（变量）

gettres：对数据获取之前的再次编译，可以理解为state的计算属性

mutations：修改状态，并且是同步的。在组件中使用$store.commit(“”,params)。这个和我们组件中的自定义事件类拟

actions：异步操作，在组件中使用是$store.dispatch(‘’)

modules：store的子模块，为了开发大型项目，方便状态管理而使用的

**理解Vuex的设计思想**

**vuex使用单一状态树，用一个对象state包含了整个应用层级的所有状态，你可以理解为这些状态就是一堆全局变量和数据**

**简而言之，vuex采用类似全局对象的形式来管理所有组件的公用数据，如果想修改这个全局对象的数据，得使用vuex提供的方式来修改（不能随意用自已的方式来修改）**

**隔离变化 约定优于配置**

**1.2.4 Vue-router**

官网文档：<https://router.vuejs.org/zh/installation.html>

**1.2.5 组件通信**

**父子组件通信**

1、使用props和$emit父子组件相互通信

2、父组件$children操作子组件

3、父组件$refs操作子组件

4、子组件$parent访问父组件

5、对象传参，不改变引用，缺点：改变了单向数据流的简单洁性及可维护性

6、V-model 和 sync双向绑定

**非父子组件通信**

1、使用中央事件总线(eventbus)

2、依赖注入(provide、inject)

3、使用$attrs和$listeners实现祖孙组件通信

4、$root直接访问根组件

**1.2.6 自定义插件**

**原理**

1. Vue.use(MyPlugin) 本质上是调用MyPlugin.install(Vue)
2. 使用插件必须在new Vue()启动应用之前完成，实例化之前就要配置好
3. 如果使用Vue.use多次注册相同插件，那只会注册成功一次

**1.2.7 单元测试**

**官方教程**

<https://vue-test-utils.vuejs.org/zh/>

**单元测试常用的工具**

1. Mocha 一个JavaScript测试框架，就是运行测试的工具
2. Jest 是 Facebook 出品的一个测试框架，相对其他测试框架，其一大特点就是就是内置了常用的测试工具，比如自带断言、测试覆盖率工具，实现了开箱即用（vue官方文档使用的测试框架）
3. Chai 一个Moach可以使用的断言库，判断源码的实际执行结果与预期结果 是否一致
4. Vue Test Utils 是Vue.js官方的单元测试实用工具
5. supertest代理HTTP请求

**创建**

vue create vue-unit-test --》 Manually select features --》 Babel、Unit Testing --》

### Jest --》 In package.json

**Vue源码分析**

**1.3.1 Vue 编译器原理探秘（一）开篇**

**1.3.2 Vue 编译器原理探秘（二）**

**1.3.3 Vue 编译器原理探秘（三）**

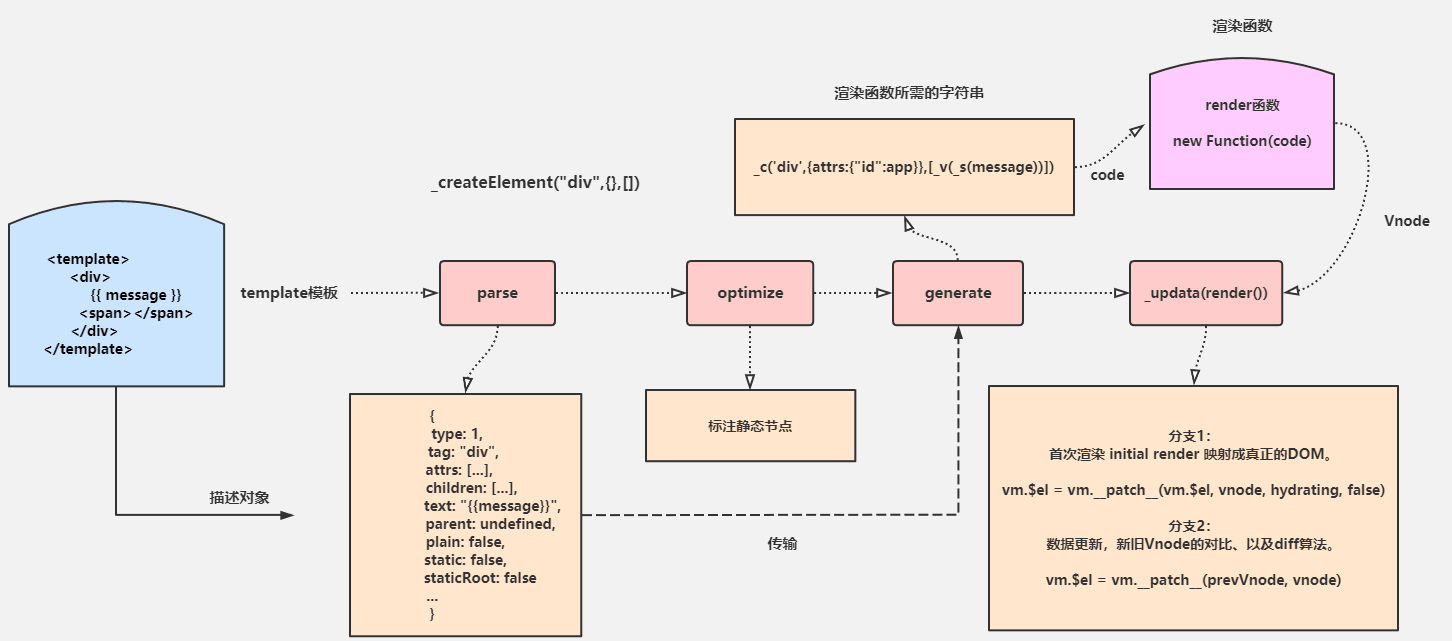
**1.3.4 Vue 编译器原理探秘（四）**

**Vue源码文章**

<https://www.zhihu.com/column/c_1162017153112363008>

**Vue编译器思维导图**

<https://www.processon.com/view/link/5cb4b6dce4b02a2858ed34a3>



注意：

optimize：在vue2.0中主要是通过AST标记静态结点

在vue3.0中提出一个区块树概念，主要是静态中找到动态

**Vue最佳实践**

**1.4.1 项目全面深度优化及打包上线之网易实践**

**github地址**

<https://github.com/vusion>

<https://github.com/vusion-templates/cloud-admin-lite>

**1.4.2 数据层的高级封装之网易实践**

**1.4.3 SSR的作用及Vue-SSR在网易中的实践**

**二、MV\* & 组件化开发 React专题**

**React基础**

**2.1.1 React介绍与React代码规范**

**2.1.2 虚拟DOM和JSX**

**2.1.3 生命周期与数据操作**

**2.1.4 组件通信**

**2.1.5 ReactDOM与表单**

**2.1.6 React与es6**

**2.1.7自定义组件**

**React进阶**

**2.2.1 高阶组件  
2.2.2 React Hooks**

**2.2.3 React 与 Typescript**

**2.2.4 数据管理(Redux)**

**2.2.5 数据管理(Mobx)**

**2.2.6 常用组件库ant-design**

**React生态**

**2.3.1 路由(React-Router)原理**

**2.3.2 React动画**

**2.3.3 可视化**

**2.3.4 企业级框架umi + dva**

**2.3.5 工程化与webpack**

**2.3.6 单元测试**

**React补充课程**

**2.4.1 实现React（上）**

**2.4.2 实现React（下）-state**

**2.4.2 实现React（下）-so-diff**

**2.4.3 Redux源码解析分享**

**三、现代前端工程实践方案 解锁webpack**

**webpack 的概念和基础使用**  
**3.1.0 Webpack导学**

**Webpack学习目标**

1. 知道webpack能做哪些，不能做哪些

了解并学会使用webpack的一些常用功能

1. 了解webpack的大致原理

知道webpack怎么工作，webpack匹配文件怎么阅读

1. 能根据业务配置合理的webpack

通过实战，根据业务配置合理的webpack配置

**3.1.1 webpack的概念与安装**

**什么是前端工程化**

1、工程化是系统化、模块化、规范化的过程

2、工程化主要解决如何提高整个系统生产效率的问题

**webpack安装**

1. 全局安装 npm install webpack wepback-cli -g
2. 新建一个项目文件夹
3. 通过npm命令初始化package.json文件
4. 创建编写webpack.config.js文件
5. 命令行运行打包 webpack

**webpack主要功能**

1. 编译，包括javascript的编译，css的编译
2. 文件的压缩，打包，合并，公共模块提取等
3. 图片等资源的处理如压缩，合并雪碧图等
4. Tree-shaking等优化javascript工具
5. Webpack-dev-server，Eslint，热更新等帮助开发的工具

**3.1.2 前端模块化**

**webpack支持的模块化规范**

commonJS AMD ES6 Module

在实际的项目开发中，建议webpack配置相关的使用commonJS，业务相关的使用ES6 Module

**3.1.3 webpack的核心概念**

**Entry和Output**

1. Entry是webpack的打包入口

代码从这开始编译

程序开始的起点

1. Output是webpack的打包出口

最终的打包结果会根据output的定义输出

会影响到资源的路径

**Loader**

loader是webpack的编译方法

webpack自身只能处理javascript，所以对于别的资源需要loader

webpack自身只能负责打包，相关的编译等操作，需要loader

loader本质是一个方法，使用时大多需要额外安装

常用loader列举

css-loader，style-loader等css处理loader

url-loader，image-loader等图片字体文件等处理loader

less-loader，sass-loader，babel-loader等编译loader

vue-loader等语法糖loader

**plugin**

plugin是webpack的额外扩展

一些插件的额外功能由plugin定义，帮助webpack优化代码，提供功能

plugin有 webpack自带的，也有需要额外安装的

常用的plugin例举

commonsChunkPlugin(抽取公共模块)，uglifyjsWebpackPlugin(压缩优化js文件)，purifyCSS（去除多余的css)等优化文件体积的插件

htmlWebpackPlugin(为html文件中引入的外部资源如,生成创建html入口文件)，hotModuleReplacementPlugin(热更新)等额外功能的插件

**3.1.4 用webpack开始一次打包**

**不利用配置文件打包**

webpack-cli --entry <entry> --output-filename <output>

**利用配置文件打包**

利用默认的配置文件打包：webpack

指定一个文件作为配置文件打包：webpack --config configfile

**局部webpack与全局webpack**

1. 全局webpack

1、什么是全局的webpack?

通过npm install webpack -g命令安装的webpack就是全局webpack，-g指令就是在npm全局目录下安装包

2、全局webpack的作用

全局webpack是必须安装的，我们在命令行直接执行webpack指令时，使用的就是全局webpack去完成的

1. 局部webpack

1、什么是局部的webpack

通过npm install [webpack@3.10.0](mailto:webpack@3.10.0) --save命令安装的webpack，会在项目文件夹下安装的webpack，即安装在项目文件夹里的node\_modules里

2、局部webpack的作用

全局的webpack有且只有一个，但是实际情况中，不同项目的webpack版本可能完全不同。当我们项目需求的webpack版本和我们全局的webpack不一致时，就需要安装局部webpack

3、利用局部的webpack打包

局部安装 npm install [webpack@3.10.0](mailto:webpack@3.10.0) --save

配置package.json文件（ scripts: { build : webpack  } ）

通过npm run build命令执行的webpack，会优化使用局部的webpack命令进行打包

在命令行执行的webpack，会默认使用全局的webpack

命令行运行package.json定义的命令

npm run build

**3.1.5 javascript的编译**

**编译es6**

1、编译需要安装的loader(只编译语法)

npm install babel-loader @babel/core @babel/preset-env -D

2、Babel-loader，babel-core和babel-preset之间关系

编译ES6需要的babel-loader 与 @babel/core ，@babel/core 是 babel-loader 的核心。babel-preset 是储存 javascript 不同标准的插件，通过使用正确的 babel-presets ，告诉 babel 按照哪个规范编译。否则 babel-loader 不会进行ES6的编译

3、es6方法的编译

常用的loader

babel-polyfill：

npm install babel-polyfill -D

Babel-transform-runtime:

npm install @babel/plugin-transform-runtime @babel/runtime -D

babel-polyfill的生效方式

生成一个全局对对象

一般用于项目开发

babel-transform-runtime的生效方式

生成一个局部对象

一般用于框架开发

**3.1.6 css的编译**

**css的引入**

1. 如何引入css文件

webpack是以js文件为入口打包的，css可以通过js文件引入，但必须使用相应的loader

1. css-loader，让css可以被js正确的引入
2. style-loader，让css被引入后可以被正确的以一个style标签插入页面
3. 两者的顺序很重要，要先进行css-loader处理，再由style-loader处理
4. 核心配置

style-loader insert 插入指定的dom

css-loader modules 开启css模块化

注意webpack+loader版本上的差异，因为有些loader的options在新版本是没的

**提取css代码为单独的文件**

1. 安装对应的插件

webpack4.0+

extract-text-webpack-plugin

npm install extract-text-webpack-plugin@next -D

webpack5.0+

mini-css-extract-plugin

npm install  mini-css-extract-plugin -D

注意：

不管extract-text-webpack-plugin还是mini-css-extract-plugin只能在局部webpack中使用，因此都要局部安装webpack webpack-cli

npm install webpack webpack-cli -D

mini-css-extract-plugin官网原话：

不要一起使用style-loader和mini-css-extract-plugin

1. 改造loader处的写法

  // webpack4.0+

        rules: [

            {

                test: /\.css$/,

                use: ExtractTextCss.extract({

                    fallback: { loader: 'style-loader', options: {} },

                    use: [{ loader: 'css-loader' }]

                })

            }

        ]

        // webpack5.0+

        rules: [

            {

                test: /\.css$/i,

                use: [MiniCssExtractPlugin.loader, 'css-loader'],

            },

        ]

1. 在plugin添加 extract-text-webpack-plugin加入到plugin里

  plugins: [

        // webpack4.0+

        new ExtractTextCss({ filename: '[name].min.css' }),

        // webpack5.0+

        new MiniCssExtractPlugin({ filename: '[name].min.css' })

    ]

**css代码加前缀**

npm install postcss postcss-loader autoprefixer postcss-cssnext -D

**3.1.7 html文件的打包**

**需要用到的plugin**

npm install html-webpack-plugin -D

**相关配置**

filename 打包生成后的html文件名字

template 指定一个html文件为模版

minify 压缩html

inject 是否把js、css文件插入到html，插入到哪

chunks 多入口时，指定引入chunks 多页面开发时用到

如不指定，会同时引入多个文件

**3.1.8 webpack的环境**

**为什么要区分环境**

因为在不同的场景下可能需要不同的配置，使用不同的功能，所以要区分环境

**webpack 实战演练**  
**3.2.1 webpack-dev-server的使用**

**什么是webpack-dev-server**

1、webpack-dev-server

项目最终都要打包上线，所以最好能模拟线上环境进行开发调试

Webpack-dev-server就是一个让我们可以模拟线上环境进行项目调试的工具

2、webpack-dev-server提供的常用额外功能

路径重定向 浏览器中显示编译错误 接口代理 热更新

3、devServer常用配置

Inline 服务的开启模式 historyApiFallback 路径重定向

port 代理接口 hot 热更新(不刷新浏览器，更新代码，从而保留状态)

lazy 懒编译 overlay 错误遮罩

proxy 代理请求

**source-map**

1、什么是source-map

为方便调试，我们需要知道打包后的代码对应于原文件的位置

如果代码有一处错误，无source-map只能追踪到错误发生在打包后的哪个位置，但是打包后的文件不方便阅读

有了source-map，就可以查看错误发生在原模块的哪个地方

2、source-map小结

|  |  |  |  |
| --- | --- | --- | --- |
| **模式** | **种类** | **特点** | **适用情况** |
| 开发  模式 | eval | 构建最快，只能定位到打包后的代码 | 无需source-map |
| eval-source-map | 构建较快，生成一个DataUrl形式sourcemap | 需要简单的调试 |
| cheap-eval-source-map | 构建快，重新构建慢，定位到转换后的代码 | 较为详细的调试 |
| cheap-module-source-map | 构建慢，重构快，能定位到原始代码 | 开发过程中一般用这个 |
| 生产模式 | source-map | 构建和重构都很慢，能定位原始代码 | 上线后一般用这个 |
| hidden-source-map | 构建和重构都很慢，能定位原始代码，但不追注释 | 一般不使用 |
| nosource-source-map | 构建和重构都很慢，不能定位到源码 | 一般不使用 |

**3.2.2 webpack原理解析**

**webpack是利用什么来打包的**

webpack依赖于node的环境与文件操作系统

webpack的打包过程，其实就是利用node去读取文件，然后进行一些字符串处理后，再利用node去写入文件

**webpack打包流程解析**

读取配置文件 注册内部插件 loader编译 组织模块 生成最终文件导出

与配置插件

**loader原理解析**

loader其实是一个方法，接一个字符串，方法内部处理完后再返回字符串

**结果文件分析**

// 打包代码组成

(function(modules){

  function \_\_webpack\_require\_\_(moduleId){}

  return \_\_webpack.require(  \_\_webpack\_require\_\_.s = "./app.js")

})({

  "./app.js":(function(){}),

  "mode1.js":(function(){}),

  "mode2.js":(function(){})

})

**webpack-dev-server原理**

利用express和一个中间件webpack-dev-middleware来开启服务，然后开启的serve会执行打包出来的代码

**热更新原理**

发生代码改变，服务通过websocket通知到客户端

客户端替换新代码

开启服务

建立一个websocket链接

**3.2.3 图片等资源的处理**

**需要的loader**

file-loader url-loader img-loader

**雪碧图的合成**

postcss-sprites

属于postcss-loader的插件，全自动把css文件中引入的背景图合成雪碧图，并修改css文件

webpack-spritesmith

属于一个独立的插件，会按照指定的路径的指定图片，生成一个雪碧图，和一个雪碧图相关的css，不会修改原css

**3.2.4 代码分割**

**一切为了优化**

1. 要解决的问题
   1. 减少加载代码大小
   2. 提取公共资源，减少加载次数
2. 多页面应用

提取公共依赖

把几个页面之中都用到的依赖给打包为一个单独文件

A

A

CC

C

B

B

C

1. 单页面应用

减少文件体积，拆分应用

把需要异步加载的内容改成异步加载

CC

A

A

CC

CC

CC

4、为了业务代码的纯净

有时不希望业务代码里混入第三方代码，或webpack配置代码

把第三方的代码和webpack配置代码拆分为单独文件

业务代码 app.js

第三方代码 vendor.js

webpack代码 manifest.js

5、所以一般这么打包

多页面应用 主业务代码+公共依赖+第三方包+webpack运行代码

单页面应用 主业务代码+异步模块+第三方包+webpack运行代码

**如何进行代码分割**

webpac4所有优化相关的操作（代码分割、压缩...）都在optimization

**3.2.5 js代码大小控制**

**压缩**

Optimization.minimize : true

把mode的模式设置为production，会自动开启压缩和tree-shaking

**tree-shaking**

是基于export的文档流

函数式编程更有利于tree-shaking

注意：不支持面向对象写法

**3.2.6 webpack打包速度优化**

**可视化打包结果分析**

1. 官方版本

webpack --profile --json >stats.json

stats.json文件分析工具：<http://webpack.github.io/analyse/>

1. 社区版本

webpack-bundle-analyzer

**优化打包速度**

1. 项目本身

减少依赖嵌套深度

使用尽可能少的处理

1. webpack层面

（1）Dll处理 先把不会变动的第三包给打包好，然后后续打包就只需要处理业务代码

（2）通过include减少loader范围

（3）happyPack 通过node开工作进程

（4）减少resolve， sourcemap，cache-loader，用新版本的node和webpack

1. 长缓存优化

（1）把hash改为chunkhash

通过修改hash为chunkhash，保证文件发生改变时，只修改变文件的hash值  利用浏览器的长缓存

缺点：当模块引用发生改变时，vendors的hash也会发生改变（在vye-cli中不会）

1. NamedChunksPlugin 和 NamedModulesPlugin

注意：任何的优化工作都需要处理时间，当代码量不大时，进行优化，优化消耗的时间明显给正常打包的 还要多

**案例  
3.3.1 网易的Webpack工程化实践**

**Eslint**

Eslint相关包

1. eslint + eslint-loader 核心内容
2. eslint-plugin-html / eslint-plugin-vue
3. eslint-friendly-formatter 友好提示
4. eslint-config-standard 代码风格标准

**一些特别的plugins**

webpack.DefinePlugin 注入全局变量

webpack.ProvidePlugin 引入项目常用的插件（不用多个文件引入）

**项目问题解决**

不要把配置当配置（当成一程序）

解决文法归纳

如果是要对模块内容进行处理 ---- loader是第一个解决方案

如果要增加一些特殊的功能 ----- 可以自定义插件

项目上的打包简化，可变性配置等 ---- 通过编写相应的操作函数

1. **VuePress项目实战**

**4.1.0-VuePress课程导学**

请参考官网：<https://www.vuepress.cn/guide/>

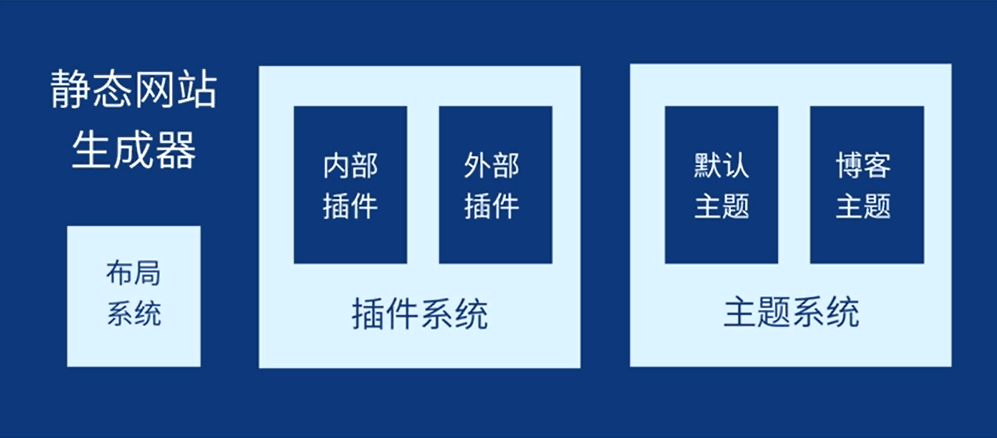
**什么是VuePress?**

VuePress由两部分组成

第一部分是一个极简静态网站生成器，它包含由Vue驱动的主题系统和插件API

第二部分是为书写技术文档而优化的默认主题，它的诞生初衷是为了支持Vue及其子项目的文档需求

**4.1.1 设计-运行原理**

****

**创建项目**

1. npm create vuepress
2. npm install -D vuepress（不再推荐全局安装 VuePress）
3. npm run dev （vuepress dev docs --temp .temp）

**4.1.2 基本配置**

**4.1.3 Markdown**

**4.1.4 上线部署 搭建你的专属博客**

**专题3 移动端App开发工程师**

1. **H5移动端开发**

**HTML5基础**

* + 1. **HTML5简介**

1、广义的HTML5是指包括HTML、CSS和JavaScript在内的一套技术组合

2、HTML5的设计目的，是在移动设备上支持多媒体

* + 1. **HTML5的新特性**

**新增标签**

header（网站头部）

nav（导航栏）

section（类似div）

article（文本内容）

aside（文件侧栏）

footer（网站脚本）

**新增表单输入框的类型及属性**

<input name="email" type="email"/>  <!-- 限制用户输入必须为Email类型 -->

        <input name="url" type="url"/>      <!-- 限制用户输入必须为URL类型 -->

        <input name="date" type="date"/>    <!-- 自动生成一个日期控件 -->

        <input name="number" type="number"/><!-- 限制用户输入必须为数字类型 -->

        <input name="range" type="range"/>  <!-- 产生一个进度条的表单 -->

        <input name="search" type="search"/><!-- 产生一个搜索意义的表单 -->

        <input name="color" type="color"/>  <!-- 生成一个颜色选择表单 -->

**Canvas**

HTML5新增的元素<canvas>只是图形的容器，必须通过本脚本（通常是JavaScript）来绘制图形

**多媒体（Video & Audio）**

<video>元素提供播放、暂停和音量控件来控件视频，提供width和height属性控制视频尺寸

<audio>元素提供音频播放

**web存储功能**

1.传统document.cookie存储方式的缺点

存储容量限制小（只有4k左右）

解析复杂

每次发送请求都携带cookie，浪费带宽，给开发带来不便

2.HTML5 Web存储功能

会放存储 生命周期为关闭浏览器窗口，同一个窗口下数据可共享

本地存储 永久生效，除非手动删除，可多窗口共享

**拖放释放功能**

1.定义

拖放是HTML5常见特性，即抓取对象以后拖到另一个位置，并且任何元素都能拖放

2.在拖动目标上会触发的事件（源对象）

ondragstart 源对象开始拖动元素时触发

ondrag 源对象正在拖动时触发

ondragend 源对象完成拖动后触发

3.在目标对象上会触发的事件

ondragenter 目标对象进入其容器范围内时触发

ondrageover 目标对象进入源对象范围内拖动时触发

ondargeleave 源对象离开其容器范围内时触发

ondrop 释放鼠标键时触发

**1.1.3选择器/history/拖放**

**选择器**

1.HTML5选择器是基于选择器规则进行查找，并且可以更方便获取DOM元素，同时编写原生javascipt更加方便

2.新增的选择

querySelector 根据选择器规则，返回相匹配的第一个元素，没有找到则返回Null

querySelectorAll 根据选择规则，返回文档中所有符合要求的元素，并且返回NodeList对象

3.选择器

子集选择器 div>p 兄弟选择器 .main\_span + p

后代选择器 div p 通用选择器 .main\_span ~ p

选中奇数项 :nth-child(odd)

选中偶数项 :nth-child(even)

**History**

1.history是历史状态管理，允许操作浏览器曾经在标签页或者框架里访问的会话历史记录

2.新增特性

pushState 每执行一次都会增加一条历史记录，浏览器在返回时，就不会返回前一个页面，并且不会刷新浏览器

3.最常用的场景之一：单页应用

单页应用定义：只有一张web页面的应用，并且一开始会加载必需的HTML、CSS和JavaScript，可以动态重写页面而不是通过服务器加载整个新页面与用户交互，可以提高用户体验

HTML5 History主要解决问题：实现网页无刷新更新数据的同时，解决浏览器无法前进/后退的问题

**1.1.4 跨文档消息通信**

**跨文档通讯概念**

跨文档消息传送，有时候也简称为XDM，指的是来自不同域的页面间传递消息

**使用场景**

内嵌框架和其父文档相互进行数据传输或者通讯

**跨文档通讯核心**

window.postMessage( message , origin , [ transfer ] )

message：消息对象 允许传入任何数据结构

origin：域消息

transfer： 转移消息对象

**1.1.5 地理信息 本地存储 离线存储**

**地理信息**

1. 定义 定位用户的位置，使用HTML5 Geolocation API用于获取用户的地理位置
2. 注意 除非用户同意，否则用户位置信息是不可用的
3. 通过对navigator.geolocation对象检测，可以判断当前设备是否支持地理信息
4. 使用getCurrentPosition( )方法来获得用户的位置

navigator.geolocation.getCurrentPosition(success,error,options)

**离线存储(应用程序缓存)**

1. 定义

HTML5引入了应用程序缓存，这意味着web应用可进行缓存，并可在没有因特网连接时进行访问

1. 离线存储优势

离线浏览：即当用户网络断开时，可以继续访问页面

访问速度快： 将资源缓存到本地，已缓存资源加载更快

减少服务器负载：浏览器将只会从服务器下载更新过或者更改过的资源

1. Cache Manifest基础

每个指定了manifest的页面在用户对其访问时都会被缓存

如果未指定manifest属性，则页面不会被 缓存

manifest文件的建议文件扩展名是：“.appcache”

1. manifest文件完整结构

CACHE MANIFEST：此标题下列出的文件将在首次下载后进行缓存，CACHE表示缓存的文件，使用相对路径

NETWORK：此标题下列出的文件需要与服务器连接，并且不会缓存

FALLBACK：标题下列出的文件规定当页面无法访问时的回退页面

使用#注释

5、html文件中的< html manifest=”cache.appcache”>标签中加上manifest=”cache.appcache”

**1.1.6 音频与视频**

**1.1.7 canvas**

**动画的基本步骤**

1、清空canvas

除非接下来要画的内容会完全充满canvas(例如背景图)，否则需要清空所有。最简单的做法就是用clearRect方法

1. 保存canvas状态

如果需要改变一些会改变canvas状态的设置(样式，变形之类的)，又要在每一帧之时都是原始状态的话，需要先保存一下

1. 绘制动画图形(animated shapes)

这一步才是重绘动画帧

1. 恢复canvas状态

如果已经保存了canvas的状态，可以先恢复它，然后重绘下一帧

**移动端适配实战演练**

**1.2.1 弹性盒子模型**

**1.2.2 移动端事件**

**1.2.3 transform详解**

**1.2.4 移动自定义滑动区域**

**1.2.5 陀螺仪操作**

**1.2.6 多指操作与兼容**

**1.2.7 上滑下载**

**1.2.8 响应式布局**

**1.2.9 滑屏组件：better-scroll**

**案例**

**网易的H5实践**

简易架构设计

Index.html 入口文件

Css 样式文件

游戏主文件(game.js)

dom操作(dom.js)

获取元素

生成dom

插入dom

游戏模型(model.js)

我方飞机模型

敌机模型

子弹模型

游戏元件(template.js)

飞机

子弹

敌机

计分板

开始界面

音效控制(music.js)

加载音乐

播放音乐

暂停音乐

计分板(score.js)

计分板效果

更新数据

移除

工具类(util.js)

更新css坐标

给dom加x,y

实例坐标转换

碰撞检测

重新开始游戏

Img ui元件

Mp3 背景音乐

**三、微信小程序**

**微信小程序入门**

**3.1.1 创建微信小程序项目**

**3.1.2 小程序工具及项目结构**

**3.1.3 配置文件**

**3.1.4 数据绑定**

**3.1.5 数据渲染  
3.1.6 WXS语法**

**3.1.7 事件系统  
3.1.8 微信小程序生命周期和运行机制**

**微信小程序进阶**  
**3.2.1 小程序页面路由及页面数据传递**

**3.2.2 小程序基础组件介绍**

**3.2.3 自定义基础组件介绍**

**小程序项目演练**  
**3.3.1 轻松部署小程序后端**

**3.3.2 微信小程序权限相关  
3.3.3 微信支付接入  
3.3.4 微信小程序第三方平台开发简介**