**【vue前端框架说明手册】**

**修订历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 版本 | 内容 | 编写人 | 审核人 |
| 2018/12/18 | V1.0 | 制定设计模板 | 郭俊秋 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# **概述**

## 导言

前端框架vue是一个JavaScriptMVVM库，是一套构建用户界面的渐进式框架，它是以数据驱动和组件化的思想构建的，采用自底向上增量开发的设计。相比于Angular.js，Vue.js提供了更加简洁、更易于理解的API，使得我们能够快速地上手并使用Vue.js

## 适用范围

* 对浏览器兼容要求不高，vuejs是到IE9,偏向于主流浏览器
* MVVM开发模式(双向数据绑定)
* 单页面应用快速开发
* 对性能要求比较高
* 需要组件化开发
* 对原生js对象进行底层操作

## 目标读者

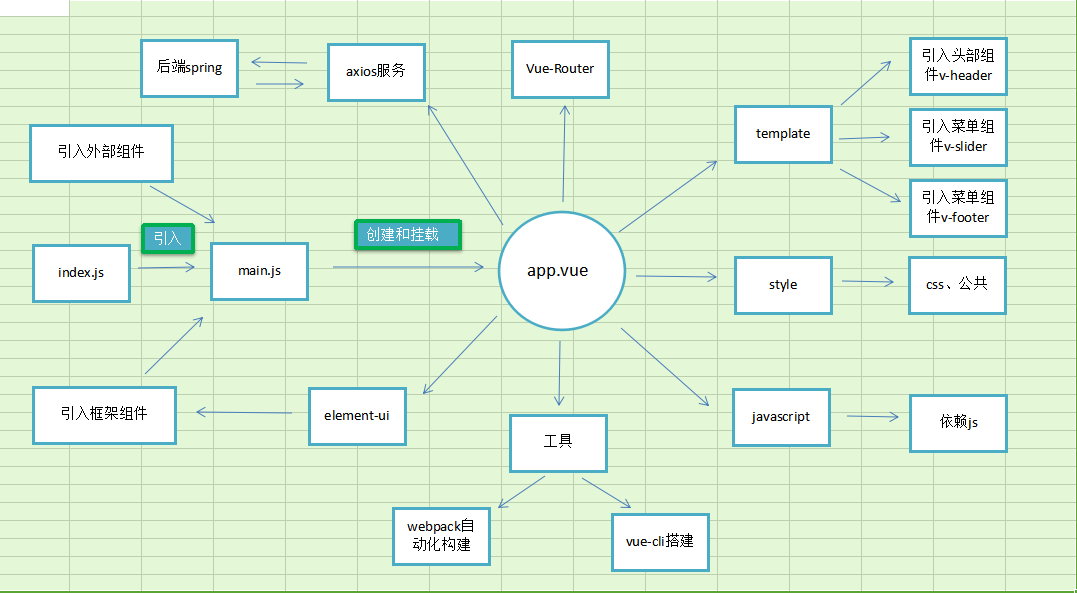
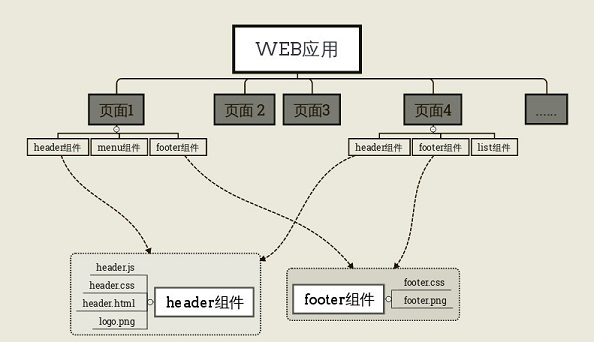
适用于前端开发人员、后台开发人员等等;

## 参考文档

* Vue.js: <https://cn.vuejs.org/v2/api/>
* vueRouter: <https://router.vuejs.org/zh/>
* Vuex: <https://vuex.vuejs.org/zh/>
* ui库：<http://element-cn.eleme.io/#/zh-CN> (pc端)
* 服务axios: <https://www.npmjs.com/package/axios>
* css预编译器: less <http://lesscss.cn/> **sass** <https://www.sass.hk/>
* ES6:<http://es6.ruanyifeng.com/>

# **框架技术设计**

## vue框架设计图



## vue项目搭建

工具: vue-cli脚手架(版本2.0)

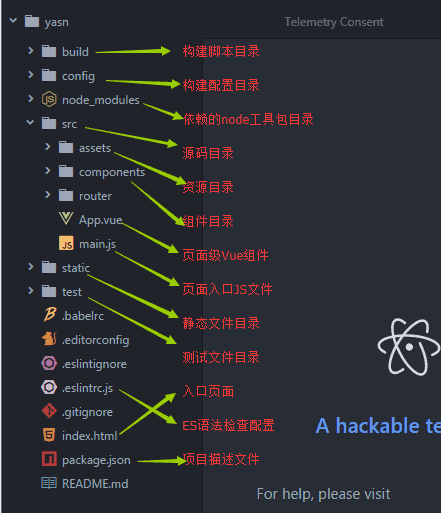
搭建步骤: 1.npm install vue-cli 安装vue-li

2.vue init webpack yourdemoname 执行这个步骤以后。就会弹出询问 “项目名称..项目描述“等等问题 直接按照提示操作

3.使用命令: cd youdemoname 然后执行npm install 就可以了，这个 时候你的项目中有多了一个node\_modules 目录

4.使用npm run dev 命令来运行项目 "npm-run-bulid" 来执行发布，会 自动生成dist文件

## vue项目结构



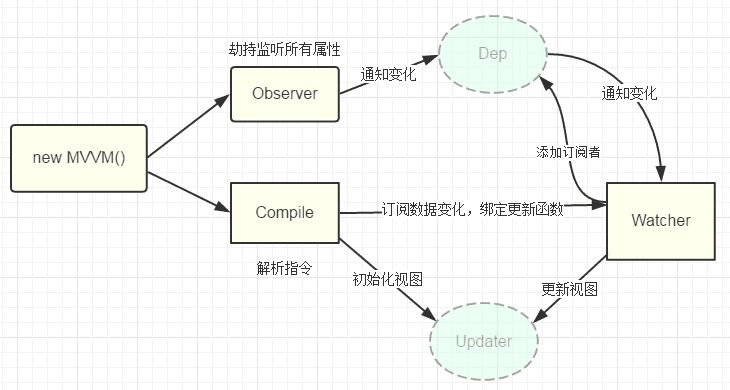
说明：前端页面的入口在src文件夹, config具体可以配置地址及相关服务, static放一些静态资源,router文件下设置单页面的路由跳转;

# **框架基础知识**

## 双向数据绑定

vue是通过数据劫持的方式来做数据绑定的，其中最核心的方法便是通过Object.defineProperty()来实现对属性的劫持，达到监听数据变动的目的;  
 要实现mvvm的双向绑定，就必须要实现以下几点：  
 1、实现一个数据监听器Observer，能够对数据对象的所有属性进行监听，如有变 动可拿到最新值并通知订阅者  
 2、实现一个指令解析器Compile，对每个元素节点的指令进行扫描和解析，根据 指令模板替换数据，以及绑定相应的更新函数  
 3、实现一个Watcher，作为连接Observer和Compile的桥梁，能够订阅并收到每 个属性变动的通知，执行指令绑定的相应回调函数，从而更新视图  
 4、mvvm入口函数，整合以上三者

上述流程如图所示：



## 组件化

在vue中,我们通过Vue.extend来创建Vue的子类,其实就是组件,也就是说Vue实例和组件的实例有差别但是差别不大因为毕竟一个是父类一个是子类,一般的应用,会拥有一个根实例,在根实例里面都是一个一个组件,因为组件是要嵌入到实例或者父组件里面的,也就是说,组件可以互相嵌套,而且所有的组件最外层必须有一个根实例,所以组件为全局组件和局部组件;全局组件在任意的实例、父级组件中都能使用,局部组件只能在创建自己的父级组件或者实例中使用,组件通过不同的注册方法成为全局、局部组件，组件通过template来确定自己的模板,template里的模板必须要有根节点,标签必须闭合组件的属性挂载通过data方法来返回一个对象作为组件的属性,这样做的目的是为了每一个组件都拥有独立的data属性:

创建组件：Vue.extend(options)

全局注册：var App = Vue.extend({

template:"<h1>hello world</h1>"

})

Vue.component('my-app',App)

## 常用指令

Vue.js的指令是已V-开头的,它们作用于HTML元素,指令提供一些特殊的特性,将指令绑在元素上,指令会为绑定的目标元素添加一些特殊的行为,我们可以将指令看作特殊的HTML特性,同时vue.js提供了一些常用的内置指令:

1. v-if : 条件渲染指令,它根据表达式的真假来删除和插入元素,它的基本语法为:v-if=”expression”, 其中expression是一个返回bool值的表达式,表达式可以是一个bool属性,也可以是一个返回bool的运算式;
2. v-show : 条件渲染指令,和v-if指令不同的是,使用v-show指令的元素始终会被渲染到HTML中,它只是简单地为元素设置css的style属性;
3. v-else: 它为v-if中添加一个else块,v-else元素必须立即和v-if或者v-show元素后面对应,否则它不被识别;
4. v-for: 基于一个数组渲染一个列表,它和javascript的遍历语法相似,基础语法为v-for=’’item in items”, 其中items是一个数组,item是当前被遍历的数组元素;
5. v-bind: 可以在其名称后面带一个参数,中间放一个冒号隔开,这个参数通常是HTML元素的特性,基础语法为: v-bind:argument=”expression”;

6.v-on: 用于给监听DOM事件,它的语法和v-bind是类似的,例如监听<a>元素的点击事件: <a v-on:click=’doSomething’>,有两种形式调用方法:绑定一个方法(让事件指向方法的引用),或者使用内联语句;

## Vuex应用

每一个Vuex应用的核心就是store(仓库),store基本上就是一个容器,它包含了你的应用中大部分状态(satate),Vuex的状态存储是响应式的,当Vue组件从store中读取状态的时候,若store中的状态发生改变,那么相应的组件也会得到高效的更新;你不能改变store中的状态,改变store的状态唯一途径就是提交mutation,这样使得我们可以方便跟踪每一个状态的变化,从而让我们实现一些工具帮助我们更好的了解我们的应用;

1.State:Vuex使用单一状态树,用一个对象包含了全部的应用层级状态,作为一个唯一数据源而存在,这也意味着,每个应用将包含一个store实例,单一状态树让我们能够直接定位任一特定的状态片段;

2.getter: Vuex 允许我们在 store 中定义“getter”（可以认为是 store 的计算属性）。就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来，且只有当它的依赖值发生了改变才会被重新计算;Getter 会暴露为 store.getters 对象，你可以以属性的形式访问这些值，Getter 也可以接受其他 getter 作为第二个参数;

3.mutation:更改 Vuex 的 store 中的状态的唯一方法是提交 mutation。Vuex 中的 mutation 非常类似于事件：每个 mutation 都有一个字符串的 事件类型 (type) 和 一个 回调函数 (handler)。这个回调函数就是我们实际进行状态更改的地方，并且它会接受 state 作为第一个参数：

const store = new Vuex.Store({

state: {

count: 1

},

mutations: {

increment (state) {

// 变更状态

state.count++

}

}})

mutation 必须是同步函数:

mutations: {

someMutation (state) {

api.callAsyncMethod(() => {

state.count++

})

}}

4.action:Action 提交的是 mutation，而不是直接变更状态，Action 可以包含任意异步操作，让我们来注册一个简单的 action：

const store = new Vuex.Store({

state: {

count: 0

},

mutations: {

increment (state) {

state.count++

}

},

actions: {

increment (context) {

context.commit('increment')

}

}})

## Router路由

路由，其实就是指向的意思，路由允许我们通过不同的 URL 访问不同的内容,当我点击页面上的home 按钮时，页面中就要显示home的内容，如果点击页面上的about 按钮，页面中就要显示about 的内容。Home按钮 => home 内容， about按钮 => about 内容，也可以说是一种映射。 通过路由，我们可以在同一个网页上浏览到不同的内容。所以在页面显示中，有两个部分，一个是需要点击的部分，一个是点击之后，显示点击内容的部分;基本使用如下:

1.定义路由组件

const home = Vue.extend({ template: '<div>home</div>' })

（当组件较多时,也可以通过在另外的文件import包含进来）

2.定义路由

const routes = [

{ path: '/home', component: home },

{ path: '/about', component: about }

]

一条路由最基本有两个属性，path和component，path是路径， component是它所对应的组件，定义了路由，也就建立了一个映射关系;

3.创建和配置路由实例

const router = new VueRouter({

routes // （缩写）相当于 routes: routes

})

4.创建和挂载根元素

const app = new Vue({

router

}).$mount('#app')

5.导航和渲染

<div id="app">

<h1>Hello App!</h1>

<p>

<!-- 使用 router-link 组件来导航. -->

<!-- 通过传入 `to` 属性指定链接. -->

<!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->

<router-link to="/home">home</router-link>

<router-link to="/about">about</router-link>

</p>

<!-- 路由出口 -->

<!-- 路由匹配到的组件将渲染在这里 -->

<router-view></router-view>

</div>

只需要在定义路由时添加上一个name属性，例如：

const routes = [

{ path: '/home', name：'home', component: home },

{ path: '/about', name：'about', component: about }

]

在导航的时候,用”:to“而不是”to“，引用的是一个对象;

<router-link to="{name:'home'}">home</router-link>

## Axios服务

axios 是一个基于Promise 用于浏览器和 nodejs 的 HTTP 客户端，它本身具有以下特征：从浏览器中创建 XMLHttpRequest、从 node.js 发出 http 请求、支持 Promise API、拦截请求和响应、转换请求和响应数据、取消请求、自动转换JSON数据、客户端支持防止 [CSRF/XSRF](http://baike.baidu.com/link?url=iUceAfgyfJOacUtjPgT4ifaSOxDULAc_MzcLEOTySflAn5iLlHfMGsZMtthBm5sK4y6skrSvJ1HOO2qKtV1ej_" \t "http://www.cnblogs.com/libin-1/p/_blank);

使用如下:

1.引入方式

$ npm install axios

$ cnpm install axios //taobao源

$ bower install axios

或者使用cdn：

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

2.执行get请求

// 向具有指定ID的用户发出请求

axios.get('/user?ID=12345')

.then(function (response) {

console.log(response);

})

.catch(function (error) {

console.log(error);

});

// 也可以通过 params 对象传递参数

axios.get('/user', {

params: {

ID: 12345

}

})

.then(function (response) {

console.log(response);

})

.catch(function (error) {

console.log(error);

})

3.执行post请求

axios.post('/user', {

firstName: 'Fred',

lastName: 'Flintstone'

})

.then(function (response) {

console.log(response);

})

.catch(function (error) {

console.log(error);

});

4.执行多个并发请求

function getUserAccount() {

return axios.get('/user/12345');

}

function getUserPermissions() {

return axios.get('/user/12345/permissions');

}

axios.all([getUserAccount(), getUserPermissions()])

.then(axios.spread(function (acct, perms) {

//两个请求现已完成

}));

5.axios API

· axios.request（config）

· axios.get（url [，config]）

· axios.delete（url [，config]）

· axios.head（url [，config]）

· axios.post（url [，data [，config]]）

· axios.put（url [，data [，config]]）

· axios.patch（url [，data [，config]]）

6.全局axios配置

axios.defaults.baseURL = 'https://api.example.com';

axios.defaults.headers.common['Authorization'] = AUTH\_TOKEN;

axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';

7.自定义实例默认值

//在创建实例时设置配置默认值

var instance = axios.create（{

   baseURL：'https://api.example.com'

}）;

//在实例创建后改变默认值

instance.defaults.headers.common['Authorization']= tokens

8.拦截器

//添加请求拦截器

axios.interceptors.request.use（function（config）{

     //在发送请求之前做某事

     return config;

   }，function（error）{

     //请求错误时做些事

     return Promise.reject（error）;

   }）;

//添加响应拦截器

axios.interceptors.response.use（function（response）{

     //对响应数据做些事

     return response;

   }，function（error）{

     //请求错误时做些事

     return Promise.reject（error）;

   }）;