**微信小程序：**

**app.json: 主页面配置文件**

pages: 设置各个页面的路径，当文件夹下文件文件名都一样时可以用一个文件名，系统自动调用.js, .json, .wxss, .wxml.四个文件

window: 设置默认页面的窗口属性

tabBar: 设置底部tab的属性

networkTimeout: 设置网络超时时间

debug: 设置是否开启debug模式

**window相关属性：**

navigationBarBackgroundColor: 导航栏的标题颜色

navigationBarTextStyle: 导航标题颜色，仅支持black，white

navigationBarTitleText: 导航栏标题文字内容

navigationStyle: 导航栏样式，仅支持default/custom

backgroundColor: 窗口的背景色

backgroundTextStyle: 下拉loading的样式，仅支持dark/light

backgroundColorTop: 顶部窗口的背景色，仅IOS支持

backgroundColorButtom: 底部窗口的背景色，仅IOS支持

enablePullDownRefresh: 是否开启下拉刷新

onReachBottomDistance: number页面上拉触底事件

**tabBar相关属性:**

color: tab上文字默认颜色

selectedColor: 文字选中时的颜色

backgroundColor: tab的背景颜色

borderStyle: tab的边框上的颜色，默认值为black

list: tab列表，设置每个tab具体的属性

position: 默认值bottom，可选top

**list中包含一个数组，数组中每一项都是一个对象**

pagePath: 页面路径，必须在pages中先定义

text: tab上按钮中文

iconPath: 图片路径，限制大小为40kb，建议尺寸81px\*81px，当position为top时，不 需要图片，即路径无效

selectedIconPath: 选中时图片路径，同样position为top时，该参数无效

**networkTimeout: 默认时常都为60000**

request: wx.request超时

connectSocket: wx. connectSocket超时

uploadFile: wx. uploadFile超时

downloadFile: wx. downloadFile超时

**每个页面的json文件：**

navigationBarBackgroundColor: 导航栏背景颜色

navigationBarTextStyle: 导航栏标题颜色

navigationBarTitleText: 导航栏标题文字内容

backgroundColor: 窗口的背景颜色

backgroundTextStyle: 下拉loading的样式，仅支持dark/light

onReachBottomDistance: 页面上拉触底事件

disableScoll: 唯一与window设置中不同的属性，是否阻止上下滚动

enablePullDownRefresh: 是否开启下拉刷新

**每个页面的js文件中Page:**

data: 初始化数据

onLoad: 监听页面加载，一个页面只会调用一次

onReady: 监听页面初次渲染完成，一个页面只调用一次

onShow: 监听页面显示，每次切换回来都会触发，首次触发在页面渲染完成之前触发

onHide: 监听页面隐藏，当页面加载后，每次切换到其他页面时触发

onUnload: 监听页面卸载

onPullDownRefresh: 监听用户下拉动作

onReachBottom: 页面上拉触底事件的处理函数

onShareAppMessage: 用户点击右上角分享

onPageScoll: 页面滚动触发事件的处理函数

onTabItemTap: 当前是tab页时，点击tab时触发

自定义处理函数

**vue**

通过new Vue(options)构建Vue对象，options是一个对象用来配置vue的相关属性

vue的混入写法，就是将某对象，或者数组单独抽离出来，例如：

new Vue({

data: data

})

data = {

…

}

vue.$emit(‘事件名’，参数)， 触发自定义事件(类似click)，并将参数作为默认参数传入

这种方式可以更加方便的添加和删除对象中的数据。混入的属性（methods，components，directives）在vue内部进行浅合并（不同名合并，同名取代前面的值，层次为一层），当与组件数据冲突时以组件数据优先。

特殊的，当同名钩子函数混入一个vue对象时，会被混合为一个数组，一起被调用，且混入的函数会在组件自身钩子函数之前被调用

refs属性，通过在DOM中添加ref属性，并给一个string值，通过Vue实例.$refs.string直接 选中标记的DOM

options = {

el: “css选择器”, //表示将vue对象挂载到哪个DOM元素上，除了这种挂载方式还可以通过(new Vue() ).$mount(el)手动挂载

data: {

//可定义任意变量来存储数据，并且可以在挂在的对象上通过{{变量名}}来使用，只能在标签的text区域使用，无法解析标签字符串

//data中的属性vue将其都直接挂载到了vue对象上，所以function中访问时直接用this.prop访问

//html中使用data时需要使用{{}}，里面填的是一个表达式，vue自动处理该表达式

}，

Props: {

Name: {

type: String || Number || … //定义数据类型

default: //定义默认值

required: true || false //true表示必须要的属性，不添加就会报错

}，

Name:type

}

将父级的属性传给子组件，单项数据流，并且父级数据改变时，相应传给子组件的值也会发生改变，但是子组件的值 改变时，父级数据不更改，并会报错，想要使用可以更改的数据，可以将数据传给data中的变量，在使用这个变量。 如果传输的数据是引用值，怎会例外，因为子组件可以修改父组件传过来的该数据，并且可以同步修改

template: ‘表示标签的字符串’，识别解析字符串并替换原来的标签，同时也可以使用vue相关语法

computed: { //没有set属性，不能直接设置值

funName() { return 相应操作后的值}

} //计算属性，使用时直接使用funName，computed与methods的区别在于，当computed中的使用的data中的属性，当其属性值发生变化，就会自动调用该方法但是methods方法不会

watch: {

propName (newValue, oleValue) { 进行相应操作}

} //监听属性发生的变化，监听什么属性名就是什么，但是监听对象属性变化时不能直接监听对象变化，因为对象属性变化但是对象地址并没有变化，所以需要通过其他方式监听对象属性的变化：通过computed，computed。 **！**vue重写了一些array方法，使得在改变数组内容时可以监听到数组的变化：push, pop, shift, unshift, splice, sort, reverse, 另外几个方法filter，concat， slice会返回新数组，所以vue可以监听到数组的变化

methods: {

fun (event) {

//这里的this指向的是vue的对实例象

//当在v-on:click=”方法名(…args)”，若此处手动传入了参数，要想使用事件对象，就需要手动将事件对象传进去，vue将事件对象绑定在$event上

}

} //里面的函数对应绑定事件的方法

}

**！**methods和computed使用的差别：computed方法使用的时候不需要将方法执行，直接使用方法名，计算方法只有当其中依赖的变量发生改变的时候才会重新计算，计算方法存在是为了防止计算花费大量的事件，所以计算方法在计算后会将结果放入缓存，再次调用时如果依赖没有发生改变，计算方法就会将缓存直接返回。而method在绑定事件的时候是不需要执行，但是在其他地方使用的时候就需要将其执行才可以，methods方法每次调用都会重新计算

Vue的相关指令：指令写在标签的属性区域

v-bind = prop //绑定属性让属性可以动态渲染{{}}里面的东西

v-text = “变量名” //效果和{{}}一样

v-html = “变量名” //弥补v-text无法识别html标签的缺点，但是实际中尽量不要使用

v-once = “变量名” //只进行一次渲染，即渲染一次后不再改变

v-on:事件名=”方法名”，也可以用@代替v-on，也可以把函数体直接写在函数的位置（支持ES6），可以通过$event传入事件源对象。如果在此不传参数，可以在函数中用一个参数接收。

v-if=”condition” //条件渲染，只有condition为true时，才会进行渲染

v-show=”condition” //与condition为false时，节点display: none，虽然与v-if视觉效果相同但是，v-if直接删除节点，v-show是隐藏节点

v-else //当最近的v-if是false的情况下才渲染

v-for=”item in [ array | object ]” //循环数组或者对象的每一项，并赋给item，item的格式也可以是（value，index|key），value表示每一项的值，第二个参数分别表示对象中的key值，或者数组中的index值，在对象中还可以添加第三个参数，表示索引。v-for还可以循环便利数字，v-for=”num in n”该操作会循环1到n。v-for=”charactor in ‘str’ ”，character会循环str中每一个字符。例外需要注意的是，使用v-for时最好加上v-bind：key来避免模板复用带来的弊端，该key值不能是引用值

v-model=”” //双向的数据绑定

v-on事件修饰符：事件名后缀

.right, .left: 鼠标右键左键

.prevent: 阻止默认事件

.once: 事件只触发一次

.stop: 阻止事件冒泡

Input中使用的：

.13=>.enter: 按下enter键

.space: 按下是space键

$emit() -> 触发当前组建上的事件。

父组件向子组件传递数据：通过v-bind:prop，通过单项数据流

子组件向父组件传递数据：通过在父组件上创建自定义事件，该事件可以更改父组件中的数据，在子组件上通过$emit来触发自定义事件，并将数据传入事件

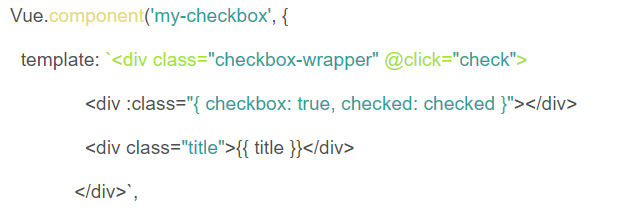
**vue组件：**

有关组件创建的方法

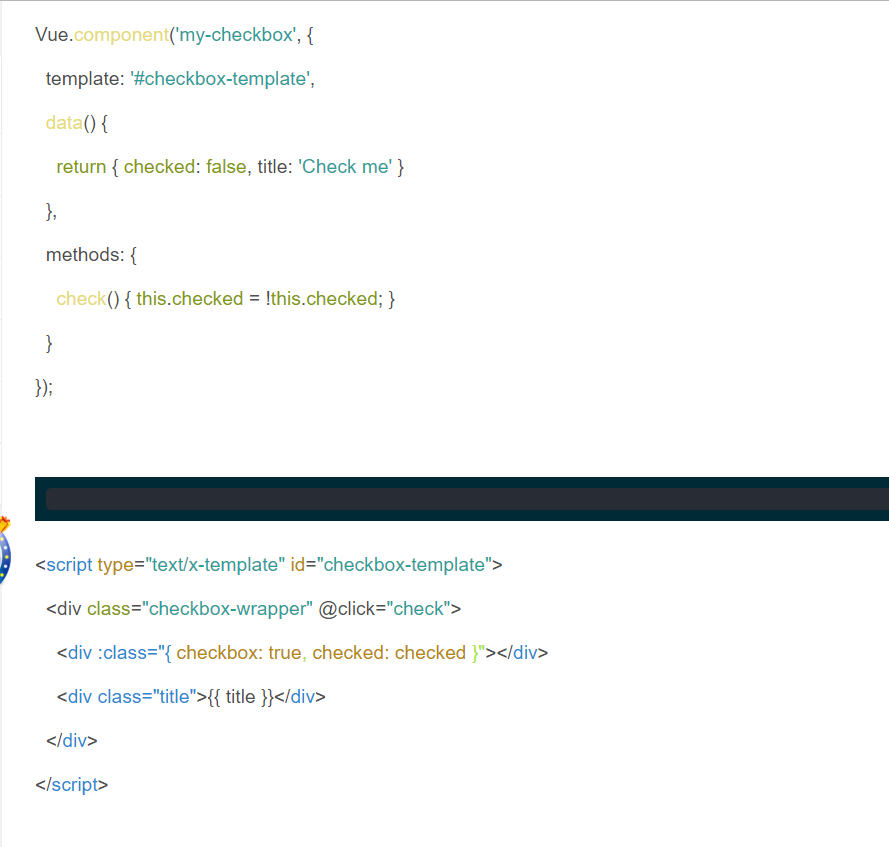
1. 字符串



1. 模板字符串



1. X-template



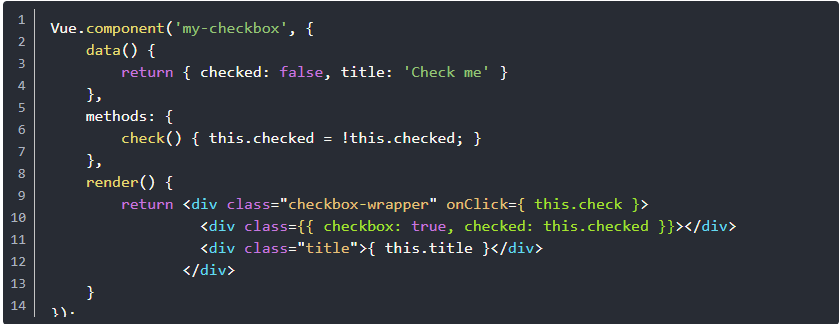
1. 内敛模板



1. Render函数



1. Jsx



1. 单文件组件：在构建工具中使用



组件的两种写法：

//全局组件，定义在全局下，必须在vue挂载的区域使用

Vue.component("name", {

template: `<div>hello</div>`,

data: () {

return {

…

}

},

Computed: { //组件中计算属性有两种写法

1. aa: function () {} //无法改变计算属性
2. aa: {  
    get: function () {return }, //获取属性

set: function () {} //设置属性会进行相应操作

}

}

Props: {

Name: {

type: String || Number || … //定义数据类型

default: //定义默认值

required: true || false //true表示必须要的属性，不添加就会报错

}

}

})

//局部组件，定义在vue对象中，必须在vue实例挂载的区域使用

new Vue({

components: {

‘componentA-name’ : component

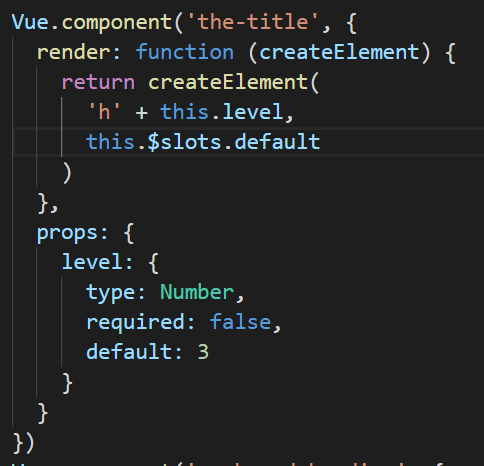
//若组件命名和组件原名相同可采用ES6的键值表示方法，直接用组件名表示，使用前需要import引入组件

}

})

//另外在vue实例中使用template，该实例挂载的区域内容会被template内容取代

**渲染函数render => 函数式组件**相对于写死的组件函数式组件灵活性更大



!createElement参数有三个：使用的组件，组件属性对象，组件子节点(可在此用createElement创建新的vNode)

若是两个参数，则一般为组件和组件子节点，

**插槽**：<slot/>，插槽是写在组件中的，为了让使用者自行在组件中加入元素(放在html标签中的位于组建头尾标签之间的元素)，插入的内容写在首尾标签中

1. 默认插槽：标签中没有slot属性的标签会自动插到默认插槽的地方
2. 具名插槽：标签中含有slot属性，查到对应name值的slot标签位置
3. 作用域插槽：将自定义内容作为子组件，将模板中的值传入该子组件，通过slot-scope=’name’中的name获取相应的参数名，及属性

例：<li slot-scope=’name’>{{name.list}}</li>

Template中：<slot v-for=’list in array’ :list=’list’></slot>

1. 插槽默认值：可以在把默认内容写在slot标签中，作为slot的默认内容

**动态组件：**

通过原组件<component></component>的is属性，通过绑定is属性(:is=’componentName’)，改变is属性值(属性值为哪个组件名就显示哪个组件)

默认情况下切换的组件会重新创建，但是可以通过keep-alive标签包括，来实现组件内容的缓存，当组件切换回来时，切换前的该组件的内容依然会存在

eg: <keep-alive>

<component></component>

</keep-alive>

**动态显示/隐藏组件**：<transition> 组件 </transition>，一下的6个是在组件不同状态事的className

v-enter: 出现前的状态

v-enter-to:完全出现的状态

v-enter-active: 出现过程

v-leave: 要消失时的状态

v-leave-to: 完全消失的状态

v-leave-active: 消失过程

作用：不是让组件消失或者显示，而是为本来要消失或出现的元素添加动态过程->以什么形式消失，以什么形式出现

transition属性name: 用来替换className中的v -> v-enter: name-enter

transition属性mode: 动态效果方式 -> out-in in-out 默认同时进行

meta: { index } //通过添加额外信息来方便识别各组件先后顺序

**//自定义指令**

Vue.directive(“name”,{ //定义指令, 通过v-name来使用

bind: function (el, binding, vnode, oldValue) {

//之调用一次，指令第一次绑定到元素时调用，在这里可以进行一次性的初始化设置

},

Inserted: function (el, binding, vnode, oldValue) {

//被绑定元素插入到父节点时调用

},

update: function (el, binding, vnode, oldValue) {

//所在组件的vnode更新时调用

},

componentUpdate: function (el, binding, vnode, oldValue) {

//指令所在的组件的vnode及其子vnode去全部更新后调用

},

unbind: function (el, binding, vnode, oldValue) {

//只调用一次，指令与元素解绑时调用

}

})

钩子函数参数：->生命周期函数

el: 所绑定的元素

binding: 包含(name, value(指令绑定的值，可以是引用值), oldValue, expression, arg, modifiers)等参数的对象

vnode: vue编译生成的虚拟节点

oldVnode: 上一个虚拟节点，只能在componentUpdate和update中使用

自动化构建项目：

->vue init webpack filename //此处webpack可以用vue list查看其他方式

**//vueRouter**

当前页面的路由信息都记录在this.$route(this.$router.history.current)中(this表示当前组件)

<router-view>: 替换为组件

<router-link to=”/path || /name(路由名)” tag=”在页面中渲染成什么标签(默认渲染为a标签)”>: 路由

！路由除了上述的声明式写法，还有编程式写法：router.push(route) //router指Router的实例对象,在组件中可以通过this.$router访问。

参数route写法：

1. 字符串，路由名
2. {path: ‘ 路由’}
3. 命名路由{name: ‘user’, params: {userId: 123} } =>user的路由/: userId=123
4. 参数查询{path: ‘route’, query: {key: value} } ({ path:`route/${key=value}` }) => route?key=value

<router-link replace> ==router.replace() 相对于router.push()，replace不会向history添加记录

redirect(to){ //重定向，用于处理加载未定义的路由

return //方法的返回值就作为重定向的路由{name:’’} || ‘path’

}

若直接使用a标签做路由跳转：

默认情况下router的mode字段为hash，即跳转路由时，不发生页面刷新，

若mode字段为history，则跳转路由会发生页面刷新

若使用router-link来实现路由跳转，即使是history模式下，页面也不会发生跳转，不同的是路由的渲染方式不同，mode下为/#/route，history模式下为/route

路由监听器：

Watch:{

$route(to, from) {

to -> route对象，表示去哪个路由

from -> route对象，表示从哪个路由来

}

}

**嵌套路由：**

在组件内部通过children承载子组件，子组件格式与组件格式相同，唯一不同是路由不需要加上“/”，访问时会自动添加上

当组件处于活跃状态时会自动添加router-link-active类名，当子组件处于活跃状态时会自动添加router-link-exect-active类名，并且该类名可以在router中的js文件里面通过LinkActiveClass替换，同样router-link-exect-active可以通过LinkActiveExactClass进行修改

active-class=”” //设置当子组件活跃时添加的类名

exact-active-class=”” //设置当活跃时的类名

**命名视图：**

即在一个路由下有多个组件构成，(将component换成components)，

components{

default: component-name 默认组件

router-name: component-name

}

**导航守卫:vue-router提供导航守卫主要用来通过跳转或取消导航，守卫都是异步解析执行的**

1. beforeEach(function (to, from, next) ), 全局守卫，针对对象( new VueRouter()，写在router实例中 )的守卫，参数有三个：to即将进入的路由, from即将离开的路由，next管理导航的函数，next()导航到to，next(false)终端导航，next({path})导航到path
2. beforeResolve()，全局解析守卫，与beforeEach相似，区别是在导航确定之前，同时所有组件内的守卫和异步路由组件被解析后，解析守卫被调用
3. beforeEnter(), 针对路由(routes数组中的值)的守卫，
4. beforeRouteEnter(to, from, next), 针对组件的守卫,在组件create函数执行前执行,写在组件中，同理还有beforeRouteUpdate，beforeRouteLeave()，注意beforeRouteEnter执行时组件实例还没有创建，不能访问this



**vuex**

vuex是为vue开发的状态管理模式，和vueRouter一样通过vue.use()使用

vuex对象中包含的属性有：

state：数据管理

getters： 根据state当众的数据派生出来的新值（类似vue实例中computed）

mutations：定义同步修改数据的方法，mutations中定义的方法默认需要传入一个参数（调用时自动传入）state，在组件中使用时通过this.$store.commit(“funName”, 另外传入的参数)

actions：定义异步修改数据的方法,异步执行的结果调用mutations方法，同步改变state中的值

modules：定义模块，即子store实例

**辅助函数：(组件实例上的方法)**

mapState: 用computed接收接受一个字符串(字符串代表state中定义的变量)数组作为参数，返回一个对象，作用是将将this.prop映射到this.$store.state.prop,相当与复制一份到this上

computed：{

…mapState({

newVal: state => state.val

})

}

mapMutations: 接受一个字符串(字符串代表mutations中定义的同步方法)数组作为参数，返回一个对象，作用是将this.fun()映射到this.$store.commit(“fun”), 即调用this.fun来代替this.$store.commit(“fun”)，参数除了数组也可以是对象，通过对象可以给store上的方法取名字，{newName: oldName}

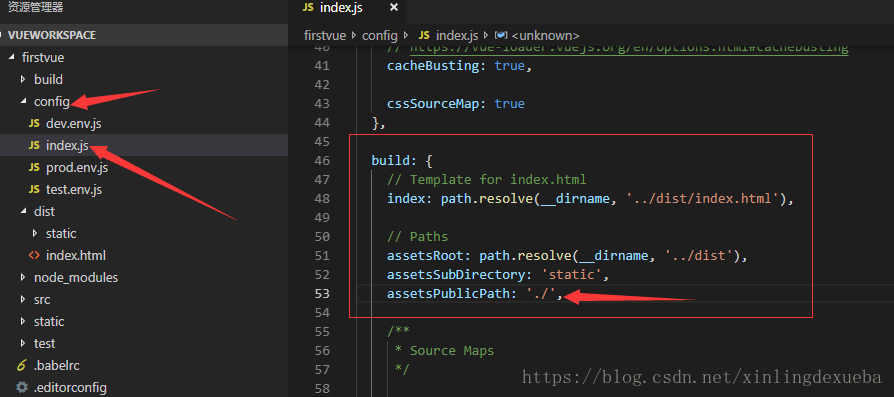
methods: {   
 …mapMutations([‘mutationsName’]) //mapMutations参数也可以写成数组，数组方法名不改变，对象可以取新的方法名

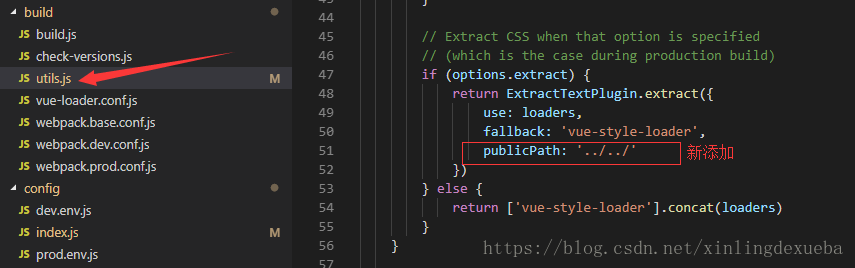
newMutation: this.$store.commit(‘mutation’, arguments)

}

mapActions: 同理

执行文件输出：





数据驱动

el: 挂载点 –> vm.mount()

插值：{{}} –> vm.data, 也可以直接内嵌js语法

data中的数据会被映射到vm上，即可以通过vm.property直接获取

指令：v-

Computed：计算属性，使用方式和属性相同

Watch：监听属性

事件定义: @/v-on: fun($event)

事件修饰符： .left .stop

.right

.prevent

.once

v-bind:

1. 实现数据的单项传递->父组件向子组件发送信息
2. :class={className1:Boolean, className2:boolean}, 建议在data中定义这个对象，将对象的引用赋给class，也可以用来定义style
3. :class=[classname1, className2]->通过这种形式，改变className的值来改变类名。当用数据来定义style时，数组中也可以添加对象，类似2中情况，vue会加以处理，变成一个对象
4. v-if v-else-if v-else v-show
5. 列表渲染 v-for=’item in arr’
6. vm.set(obj, position, value)->防止引用改变属性时，vue-mode监听不到

vm.$destroy: 解除el上的挂载，即使vm.data发生改变，页面也不会发生改变，当手动挂在后就会发生改变

LBC = (function () {

return {

add(value) {}

get(){}

remove(){}

clear(){}

}

}())