# **[单页面与多页面间的区别及优缺点](https://www.cnblogs.com/yunyea/p/8824178.html)**

单页面应用（SPA），通俗一点说就是指只有一个主页面的应用，浏览器一开始要加载所有必须的 html, js, css。所有的页面内容都包含在这个所谓的主页面中。但在写的时候，还是会分开写（页面片段），然后在交互的时候由路由程序动态载入，单页面的页面跳转，仅刷新局部资源。多应用于pc端。

**多页面（MPA）**，就是指一个应用中有多个页面，页面跳转时是整页刷新

单页面的优点：

1，用户体验好，快，内容的改变不需要重新加载整个页面，基于这一点spa对服务器压力较小

2，前后端分离

3，页面效果会比较炫酷（比如切换页面内容时的专场动画）

单页面缺点：

1，不利于seo

2，导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）

3，初次加载时耗时多

4，页面复杂度提高很多



# Rem原理

****rem作用于非根元素时，相对于根元素字体大小；rem作用于根元素字体大小时，相对于其出初始字体大小****

****其实rem布局的本质是等比缩放，一般是基于宽度****，

# [keep-alive](https://cn.vuejs.org/v2/api/" \l "keep-alive" \o "keep-alive)：

keep-alive是Vue.js的一个内置组件。它能够不活动的组件实例保存在内存中，而不是直接将其销毁，它是一个抽象组件，不会被渲染到真实DOM中，也不会出现在父组件链中。

Props：

* include - 字符串或正则表达式。只有名称匹配的组件会被缓存。
* exclude - 字符串或正则表达式。任何名称匹配的组件都不会被缓存。
* max - 数字。最多可以缓存多少组件实例。

注意：<keep-alive> 要求被切换到的组件都有自己的名字，不论是通过组件的 name 选项还是局部/全局注册。

与缓存有关的生命周期钩子activated和deactivated来得知当前组件是否处于活动状态

## **深入keep-alive组件实现**

### **created与destroyed钩子**

created钩子会创建一个cache对象，用来作为缓存容器，保存vnode节点。

created () { */\* 缓存对象 \*/* this.cache = Object.create(null) },

destroyed钩子则在组件被销毁的时候清除cache缓存中的所有组件实例。

destroyed(){

for (const key in this.cache) {

pruneCacheEntry(this.cache[key])

}

},

### **render**

render () {

/\* 得到slot插槽中的第一个组件 \*/

const vnode: VNode = getFirstComponentChild(this.$slots.default)

const componentOptions: ?VNodeComponentOptions = vnode && vnode.componentOptions

if (componentOptions) {

// check pattern

/\* 获取组件名称，优先获取组件的name字段，否则是组件的tag \*/

const name: ?string = getComponentName(componentOptions)

/\* name不在inlcude中或者在exlude中则直接返回vnode（没有取缓存） \*/

if (name && (

(this.include && !matches(this.include, name)) ||

(this.exclude && matches(this.exclude, name))

)) {

return vnode

}

const key: ?string = vnode.key == null

// same constructor may get registered as different local components

// so cid alone is not enough (#3269)

? componentOptions.Ctor.cid + (componentOptions.tag ? `::${componentOptions.tag}` : '')

: vnode.key

/\* 如果已经做过缓存了则直接从缓存中获取组件实例给vnode，还未缓存过则进行缓存 \*/

if (this.cache[key]) {

vnode.componentInstance = this.cache[key].componentInstance

} else {

this.cache[key] = vnode

}

/\* keepAlive标记位 \*/

vnode.data.keepAlive = true

}

return vnode

}

首先通过getFirstComponentChild获取第一个子组件，获取该组件的name（存在组件名则直接使用组件名，否则会使用tag）。接下来会将这个name通过include与exclude属性进行匹配，匹配不成功（说明不需要进行缓存）则不进行任何操作直接返回vnode，

/\* 检测name是否匹配 \*/

function matches (pattern: string | RegExp, name: string): boolean {

if (typeof pattern === 'string') {

/\* 字符串情况，如a,b,c \*/

return pattern.split(',').indexOf(name) > -1

} else if (isRegExp(pattern)) {

/\* 正则 \*/

return pattern.test(name)

}

/\* istanbul ignore next \*/

return false

}

检测include与exclude属性匹配的函数很简单，include与exclude属性支持字符串如”a,b,c”这样组件名以逗号隔开的情况以及正则表达式。matches通过这两种方式分别检测是否匹配当前组件。

if (this.cache[key]) {

vnode.componentInstance = this.cache[key].componentInstance

} else {

this.cache[key] = vnode

}

接下来的事情很简单，根据key在this.cache中查找，如果存在则说明之前已经缓存过了，直接将缓存的vnode的componentInstance（组件实例）覆盖到目前的vnode上面。否则将vnode存储在cache中。

最后返回vnode（有缓存时该vnode的componentInstance已经被替换成缓存中的了）。

watch

用watch来监听pruneCache与pruneCache这两个属性的改变，在改变的时候修改cache缓存中的缓存数据。

watch: {

/\* 监视include以及exclude，在被修改的时候对cache进行修正 \*/

include (val: string | RegExp) {

pruneCache(this.cache, this.\_vnode, name => matches(val, name))

},

exclude (val: string | RegExp) {

pruneCache(this.cache, this.\_vnode, name => !matches(val, name))

}

},

1

2

3

4

5

6

7

8

9

来看一下pruneCache的实现。

/\* 修正cache \*/

function pruneCache (cache: VNodeCache, current: VNode, filter: Function) {

for (const key in cache) {

/\* 取出cache中的vnode \*/

const cachedNode: ?VNode = cache[key]

if (cachedNode) {

const name: ?string = getComponentName(cachedNode.componentOptions)

/\* name不符合filter条件的，同时不是目前渲染的vnode时，销毁vnode对应的组件实例（Vue实例），并从cache中移除 \*/

if (name && !filter(name)) {

if (cachedNode !== current) {

pruneCacheEntry(cachedNode)

}

cache[key] = null

}

}

}

}

/\* 销毁vnode对应的组件实例（Vue实例） \*/

function pruneCacheEntry (vnode: ?VNode) {

if (vnode) {

vnode.componentInstance.$destroy()

}

}

遍历cache中的所有项，如果不符合filter指定的规则的话，则会执行pruneCacheEntry。pruneCacheEntry则会调用组件实例的$destroy方法来将组件销毁。

## **Vuex是什么**

state => 基本数据   
 getters => 从基本数据派生的数据   
 mutations => 提交更改数据的方法，同步！   
 actions => 像一个装饰器，包裹mutations，使之可以异步。   
 modules => 模块化Vuex

一个 Promise有以下几种状态:

* pending: 初始状态，既不是成功，也不是失败状态。
* fulfilled: 意味着操作成功完成。
* rejected: 意味着操作失败。

# javascirpt中的数字在计算机内存储为多少Byte？

**1.Javascript中，由于其变量内容不同，变量被分为基本数据类型变量和引用数据类型变量。基本类型变量用八字节内存，存储基本数据类型(数值、布尔值、null和未定义)的值，引用类型变量则只保存对对象、数组和函数等引用类型的值的引用(即内存地址)。**

**2.** **JS中的数字是不分类型的，也就是没有byte/int/float/double等的差异。**

1. 计算机最小的计量单位是bit,翻译为位
2. 8bit = 1byte,byte翻译为字节