1.vue组件传值几种方式

- 父组件通过prop向子组件传值
- 子组件通过this.$emit()触发父组组件传递过来的方法向父组件传值
- 兄弟组件之间不能直接传值，需要通过父组件来做间接传值，在这种情况下推荐使用vuex
- 中央事件总线

```
//Bus.$emit 发送消息
Bus.$emit('inceptMessage', this.msg)

//Bus.$on  接收消息
Bus.$on('inceptMessage',(msg) => {
    this.fromComponentAMsg = msg
})
```

- provide和inject
- `this.$refs`和`this.$parent`

具体例子请看官方文档

2.vue-router原理

说简单点，vue-router的原理就是监听URL地址变化，从而渲染不同的组件。

```
vue-router的模式主要有hash模式和history模式。

1.hash模式的原理(url带有#号部分)：
    在vue-router.js的2.8版本之前，在路由的hash部分发生了任何变化，都会执行window.onhashchange方法，在这个方法内部我们可以根据当前匹配到的hash去加载对应的组件
    在vue-router.js的2.8版本之后，内部使用window.history.pushState来完成相应的功能
    hash模式的特点：在切换路由的时候，不会向服务器发送请求，但是刷新网页的时候，此时会向服务器发送请求，在向服务器发送请求的时候，hash部分的信息是不会发送到服务器的，所以此时刷新网页没有问题

   localhost:8888          --->加载spa单页面应用程序
   localhost:8888#index    --->当localhost:8888切换到localhost:8888#index 没有发送请求
   localhost:8888#login    --->当localhost:8888#index切换到localhost:8888#login 没有发送请求

   localhost:8888#login    --->当我们直接刷新浏览器的时候，此时会向服务器发送请求，但是我们请求url的中hash部分不会提交到服务器，所以真正的请求地址是localhost:8888，就相当于此时重新去加载spa单页面应用程序

2.history模式的原理(url中通过/表示路径)
    内部使用window.history.pushState来处理url的变化，切换对应的组件
    history模式的特点：在切换路由的时候，不会向服务器发送请求，但是当刷新网页的时候，此时会向服务器发送请求，如果后端没有对应的接口与此匹配，此时会报资源找不到的错误
history模式一般情况下不能刷新网页


    localhost:8888      --->加载spa单页面应用程序
```

```
    localhost:8888/index --->当localhost:8888切换到localhost:8888/index没有发送请求
    localhost:8888/login --->当localhost:8888/index切换到localhost:8888/login没有发送请求

    localhost:8888/login  --->当我们直接刷新浏览器的时候，此时会向服务器发送请求，请求地址就是
localhost:8888/login，所以如果服务器没有对应的/login接口，那么此时会报404错误
```

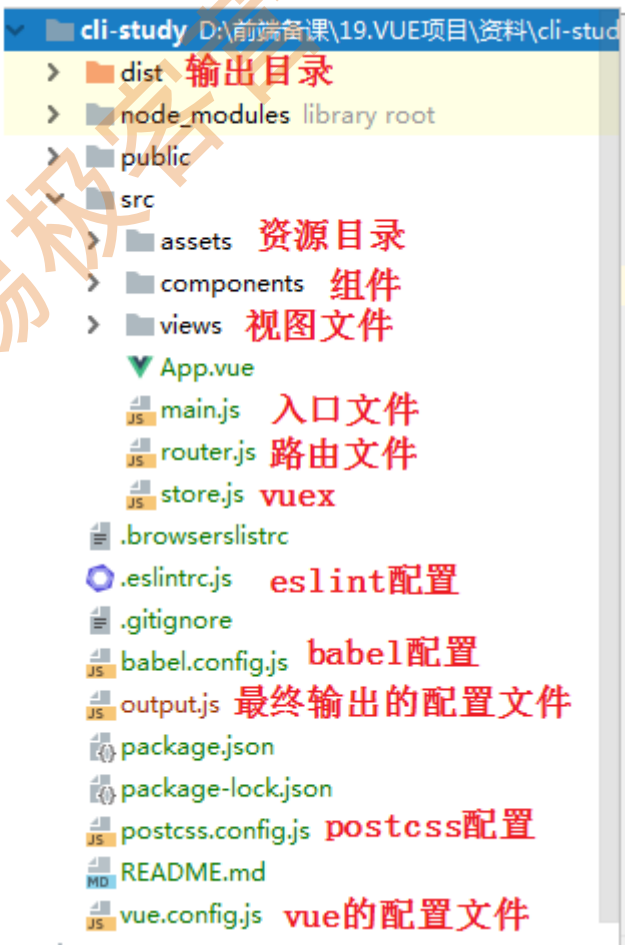
3.构建的 vue-cli 工程都到了哪些技术，它们的作用分别是什么？

```
1、vue.js：vue-cli工程的核心，主要特点是双向数据绑定和组件系统。
2、vue-router：vue官方推荐使用的路由框架。
3、vuex：专为 Vue.js 应用项目开发的状态管理器，主要用于维护vue组件间共用的一些 变量 和 方法。
4、axios（ 或者 fetch 、ajax ）：用于发起 GET 、或 POST 等 http请求，基于 Promise 设计。
5、vux  mint-UI  AntDesign  ElementUI等：为vue设计的UI组件库。
6、webpack：模块加载和vue-cli工程打包器。
7、AnimateCSS：动画库
```

4.vue-cli 工程常用的 npm 命令有哪些？

```
npm run serve
npm run build
npm run lint
npm run build --report  用于查看 vue-cli 生产环境部署资源文件大小
```

5.请说出vue-cli工程中每个文件夹和文件的用处



6.vue.config.js 的对于工程 开发环境 和 生产环境 的配置

```
const debug = process.env.NODE_ENV !== 'production'

configureWebpack: config => {
    // 开发环境配置
    if (debug) {
      // cheap-module-eval-source-map是打包文件时最快的生成source map的方法，生成的Source Map会和打包后的JavaScript文件同行显示，没有列映射
      config.devtool = '#cheap-module-eval-source-map'
    }
    // 生产环境配置
    else {
       //生成一个没有列信息（column-mappings）的SourceMaps文件，同时 loader 的 sourcemap 也被简化为只包含对应行的
       config.devtool = 'cheap-module-source-map'
    }
    // 开发生产共同配置
    Object.assign(config, {
      resolve: {
        alias: {
          '@': path.resolve(__dirname, './src'),
          'vue$': 'vue/dist/vue.esm.js'
        }
      }
    })
  },
```

7.请你详细介绍一些 package.json 里面的配置

```
{
  "name": "cli-study",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "vue": "^2.5.21",
    "vue-router": "^3.0.1",
    "vuex": "^3.0.1"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^3.3.0",
    "@vue/cli-plugin-eslint": "^3.3.0",
    "@vue/cli-service": "^3.3.0",
    "@vue/eslint-config-prettier": "^4.0.1",
    "babel-eslint": "^10.0.1",
```


```
    "eslint": "^5.8.0",
    "eslint-plugin-vue": "^5.0.0",
    "less": "^3.0.4",
    "less-loader": "^4.1.0",
    "lint-staged": "^8.1.0",
    "vue-template-compiler": "^2.5.21"
  },
  "gitHooks": {
    "pre-commit": "lint-staged"
  },
  "lint-staged": {
    "*.js": [
      "vue-cli-service lint",
      "git add"
    ],
    "*.vue": [
      "vue-cli-service lint",
      "git add"
    ]
  }
}
```

常用对象解析：

- scripts：npm run xxx 命令调用node执行的 .js 文件
- dependencies：生产环境依赖包的名称和版本号，即这些 依赖包 都会打包进 生产环境的JS文件里面
- devDependencies：开发环境依赖包的名称和版本号，即这些 依赖包 只用于 代码开发 的时候，不会打包进生产环境js文件 里面　npm install xx --save-dev

8.vue.js的核心

1、数据驱动，也叫双向数据绑定。

> Vue.js使用ES5的Object.defineProperty和存储器属性: getter和setter实现数据的监听(兼容IE9及以上版本)。核心是VM，即ViewModel，保证数据和视图的一致性。

2、组件系统。

.vue组件的核心选项:

> 1、模板（template）：模板声明了数据和最终展现给用户的DOM之间的映射关系。 2、初始数据（data）：一个组件的初始数据状态。对于可复用的组件来说，这通常是私有的状态。 3、接受的外部参数(props)：组件之间通过参数来进行数据的传递和共享。 4、方法（methods）：对数据的改动操作一般都在组件的方法内进行。 5、生命周期钩子函数（lifecycle hooks）：一个组件会触发多个生命周期钩子函数，最新2.0版本对于生命周期函数名称改动很大。 6、私有资源（directives、components、filters）：Vue.js当中将用户自定义的指令、过滤器、组件等统称为资源。一个组件可以声明自己的私有资源。私有资源只有该组件和它的子组件可以调用。

3、vuex。

> vuex是是vue中的状态管理方案，主要用于多个组件之间的数据共享。

9.对于 Vue 是一套 构建用户界面 的 渐进式框架 的理解

渐进式代表的含义是：没有多做职责之外的事。vue.js只提供了 vue-cli 生态中最核心的 `组件系统` 和 `双向数据绑定`。像 `vuex`、`vue-router` 都属于围绕 `vue.js` 开发的库，可以集成也可以不集成。

> 比如说，你要使用Angular，必须接受以下东西：
> - 必须使用它的模块机制
> - 必须使用它的依赖注入
> - 必须使用它的特殊形式定义组件（这一点每个视图框架都有，难以避免）
> 所以Angular是带有比较强的排它性的，如果你的应用不是从头开始，而是要不断考虑是否跟其他东西集成，这些主张会带来一些困扰。
>
> 比如说，你要使用React，你必须理解：
> - 函数式编程的理念
> - 需要知道什么是副作用
> - 什么是纯函数
> - 如何隔离副作用
> - 它的侵入性看似没有Angular那么强，主要因为它是软性侵入。
>
> Vue与React、Angular的不同是，但它是渐进的：
> - 你可以在原有大系统的上面，把一两个组件改用它实现，当jQuery用
> - 也可以整个用它全家桶开发，当Angular用；
> - 还可以只用它的视图，搭配你自己设计的整个下层应用。
> - 你可以在底层数据逻辑的地方用oop（对象编程），也可以函数式编程，都可以，它只是个轻量视图而已，只做了最核心的东西。

10.请说出vue几种常用的指令

- v-if：根据表达式的值的真假条件渲染元素。在切换时元素及它的数据绑定 / 组件被销毁并重建。
- v-show：根据表达式之真假值，切换元素的 display CSS 属性。
- v-for：循环指令，基于一个数组或者对象渲染一个列表，vue 2.0以上必须需配合 key值 使用。
- v-bind：动态地绑定一个或多个特性，或一个组件 prop 到表达式。
- v-on：用于监听指定元素的DOM事件，比如点击事件。绑定事件监听器。
- v-model：实现表单输入和应用状态之间的双向绑定
- v-pre：跳过这个元素和它的子元素的编译过程。可以用来显示原始 Mustache 标签。跳过大量没有指令的节点会加快编译。
- v-once：只渲染元素和组件一次。随后的重新渲染，元素/组件及其所有的子节点将被视为静态内容并跳过。这可以用于优化更新性能。
- v-html: 显示html
- v-text: 文本内容，不识别html
- v-cloak

11.请问 v-if 和 v-show 有什么区别

共同点：
v-if 和 v-show 都是动态显示DOM元素。

区别：
1、编译过程： v-if 是 真正 的 条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。v-show 的元素始终会被渲染并保留在 DOM 中。v-show 只是简单地切换元素的 CSS 属性display。
2、编译条件： v-if 是惰性的：如果在初始渲染时条件为假，则什么也不做。直到条件第一次变为真时，才会开始渲染条件块。v-show不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。
3、性能消耗： v-if有更高的切换消耗。v-show有更高的初始渲染消耗。
4、应用场景： v-if适合运行时条件很少改变时使用。v-show适合频繁切换。

12.vue常用的修饰符

v-on 指令常用修饰符：

- .stop - 调用 event.stopPropagation()，禁止事件冒泡。
- .prevent - 调用 event.preventDefault()，阻止事件默认行为。
- .capture - 添加事件侦听器时使用 capture 模式。
- .self - 只当事件是从侦听器绑定的元素本身触发时才触发回调。
- .{keyCode | keyAlias} - 只当事件是从特定键触发时才触发回调。
- .native - 监听组件根元素的原生事件。
- .once - 只触发一次回调。
- .left - (2.2.0) 只当点击鼠标左键时触发。
- .right - (2.2.0) 只当点击鼠标右键时触发。
- .middle - (2.2.0) 只当点击鼠标中键时触发。

```
//注意： 如果是在自己封装的组件或者是使用一些第三方的UI库时，会发现一些事件并不起效果，这时就需要用.native修饰符了，如当我们使用element-ui中的el-input组件时的@keyup.enter事件：
<el-input
  v-model="inputName"
  placeholder="搜索你的文件"
  @keyup.enter.native="searchFile(params)"
  >
</el-input>
```

v-bind 指令常用修饰符：
- .prop 被用于绑定 DOM 属性 (property)。(差别在哪里？)
- .camel (2.1.0+) 将 kebab-case 特性名转换为 camelCase. (从 2.1.0 开始支持)
- .sync (2.3.0+) 语法糖，会扩展成一个更新父组件绑定值的 v-on 侦听器。

v-model 指令常用修饰符：
- .lazy 取代 input 监听 change 事件
- .number 输入字符串转为数字
- .trim 输入首尾空格过滤

13.v-on可以监听多个方法吗？

```
v-on可以监听多个方法，例如：
<input type="text" :value="name" @input="onInput" @focus="onFocus" @blur="onBlur" />

也可以给一个事件绑定多个方法，如以下代码：
<p @click="one(),two()">点击</p>
```

14.vue中 key 值的作用

key值：用于 管理可复用的元素。因为`Vue` 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染。这么做使 Vue 变得非常快，但是这样也不总是符合实际需求。

> 2.2.0+ 的版本里，当在组件中使用 v-for 时，key 现在是必须的，key的取值需要是number或者string，而且需要在同级唯一。

15.vue事件中如何使用event对象

注意在事件中要使用 $ 符号

```
//html部分
<a href="javascript:void(0);" data-id="12" @click="showEvent($event)">event</a>

//js部分
showEvent(event){
    //获取自定义data-id
    console.log(event.target.dataset.id)
   //阻止事件冒泡
    event.stopPropagation();
    //阻止默认
    event.preventDefault()
}
```

16.什么是$nextTick

`$nextTick` 下一次dom更新完毕之后执行

因为`Vue`的异步更新队列，`$nextTick`是用来知道什么时候`DOM`更新完成的。

17.Vue 组件中 data 为什么必须是函数

```
//为什么data函数里面要return一个对象
<script>
    export default {
        data() {
            return {  // 返回一个唯一的对象，不要和其他组件共用一个对象进行返回
                menu: MENU.data,
                poi: POILIST.data
            }
        }
    }
</script>
```


因为一个组件是可以共享的，同一个组件类可以创建很多歌组件对象，但每一个组件对象的data应该是私有的，所以每个组件都要return一个新的data对象，返回一个唯一的对象，不要和其他组件共用一个对象。

18.v-for 与 v-if 的优先级

当它们处于同一节点，`v-for`的优先级比`v-if`更高，这意味着`v-if`将分别重复运行于每个`v-for`循环中。

19.vue中子组件调用父组件的方法

主要步骤：

```
1、在父组件创建子组件实例的时候，通过v-on给子组件传递一个自定义事件。
2、在子组件 中 通过'$emit'触发 当前实例上的 自定义事件。
```

示例：

父组件：

```
<template>
    <div class="fatherPageWrap">
          <h1>这是父组件</h1>
          <!-- 引入子组件，v-on监听自定义事件 -->
          <emitChild v-on:emitMethods="fatherMethod"></emitChild>
    </div>
</template>

<script type="text/javascript">
    import emitChild from '@/page/children/emitChild.vue';
    export default{
        data () {
            return {}
        },
        components : {
            emitChild
        },
        methods : {
            fatherMethod(params){
                alert(JSON.stringify(params));
            }
        }
    }
</script>
```

子组件：

```
<template>
    <div class="childPageWrap">
         <h1>这是子组件</h1>
    </div>
</template>
```

```
<script type="text/javascript">
    export default{
        data () {
           return {}
        },
        mounted () {
           //通过 emit 触发
           this.$emit('emitMethods',{"name" : 123});
        }
    }
</script>
```

结果：子组件 会调用 父组件的`fatherMethod`方法，该并且会`alert`传递过去的参数：`{"name":123}`

20.vue中 keep-alive 组件的作用

`keep-alive：`主要用于`保留组件状态`或`避免组件重新渲染`。

属性：

- `include:`字符串或正则表达式。只有匹配的组件会被缓存。
- `exclude：`字符串或正则表达式。任何匹配的组件都不会被缓存。

21.vue中如何编写可复用的组件

组件由*状态*、事件和嵌套的片断组成。

状态 props，是组件当前的某些数据或属性，如 video 中的 src、width 和 height。

事件 events，是组件在特定时机触发一些操作的行为，如 video 在视频资源加载成果或失败时会触发对应的事件来执行处理。

片段 slots，指的是嵌套在组件标签中的内容，该内容会在某些条件下展现出来，如在浏览器不支持 video 标签时显示提示信息。

在编写组件的时候，要时刻考虑组件的复用性，良好的可复用组件应当定义一个清晰的公开接口。

- **props** 允许外部环境传递数据给组件
- **events** 允许组件触发外部环境的副作用
- **slots** 允许外部环境将额外的内容组合在组件中。

```
<my-video
  :playlist="playlist"
  width="320"
  height="240"
  @load="loadHandler"
  @error="errorHandler"
  @playnext="nextHandler"
  @playprev="prevHandler">
  <div slot="endpage"></div>
</my-video>
```

22.什么是vue生命周期和生命周期钩子函数

`vue` 的生命周期是： vue 实例从创建到销毁，也就是从`开始创建`、`初始化数据`、`编译模板`、`挂载Dom→渲染`、`更新→渲染`、`卸载`等一系列过程。

在这个过程中也会`运行`一些叫做`生命周期钩子的函数`，这给了`用户`在不同阶段`添加自己的代码`的机会。

23.vue生命周期钩子函数有哪些

生命周期钩子函数（11个）	类型	详细
beforeCreate	Function	在`实例初始化之后`，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。
created	Function	在`实例创建完成后`被立即调用。在这一步，实例已完成以下的配置：`数据观测 (data observer)`，`属性和方法的运算`，`watch/event 事件回调`。然而，挂载阶段还没开始，$el 属性目前不可见。
beforeMount	Function	在`挂载开始之前`被调用：相关的 render 函数首次被调用。
mounted	Function	`el` 被新创建的 `vm.$el` 替换，并`挂载到实例上去之后`调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。
beforeUpdate	Function	`数据更新时调用`，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。**该钩子在服务器端渲染期间不被调用，因为只有初次渲染会在服务端进行。**
updated	Function	由于数据更改导致的`虚拟 DOM 重新渲染和打补丁`，在这`之后会调用`该钩子。
activated	Function	`keep-alive 组件激活时调用`。**该钩子在服务器端渲染期间不被调用。**
deactivated	Function	`keep-alive 组件停用时调用`。**该钩子在服务器端渲染期间不被调用。**
beforeDestroy	Function	实例销毁之前调用。在这一步，实例仍然完全可用。**该钩子在服务器端渲染期间不被调用。**
destroyed	Function	Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。**该钩子在服务器端渲染期间不被调用。**
errorCaptured（2.5.0+新增）	(err: Error, vm: Component, info: string) => ?boolean	当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播。

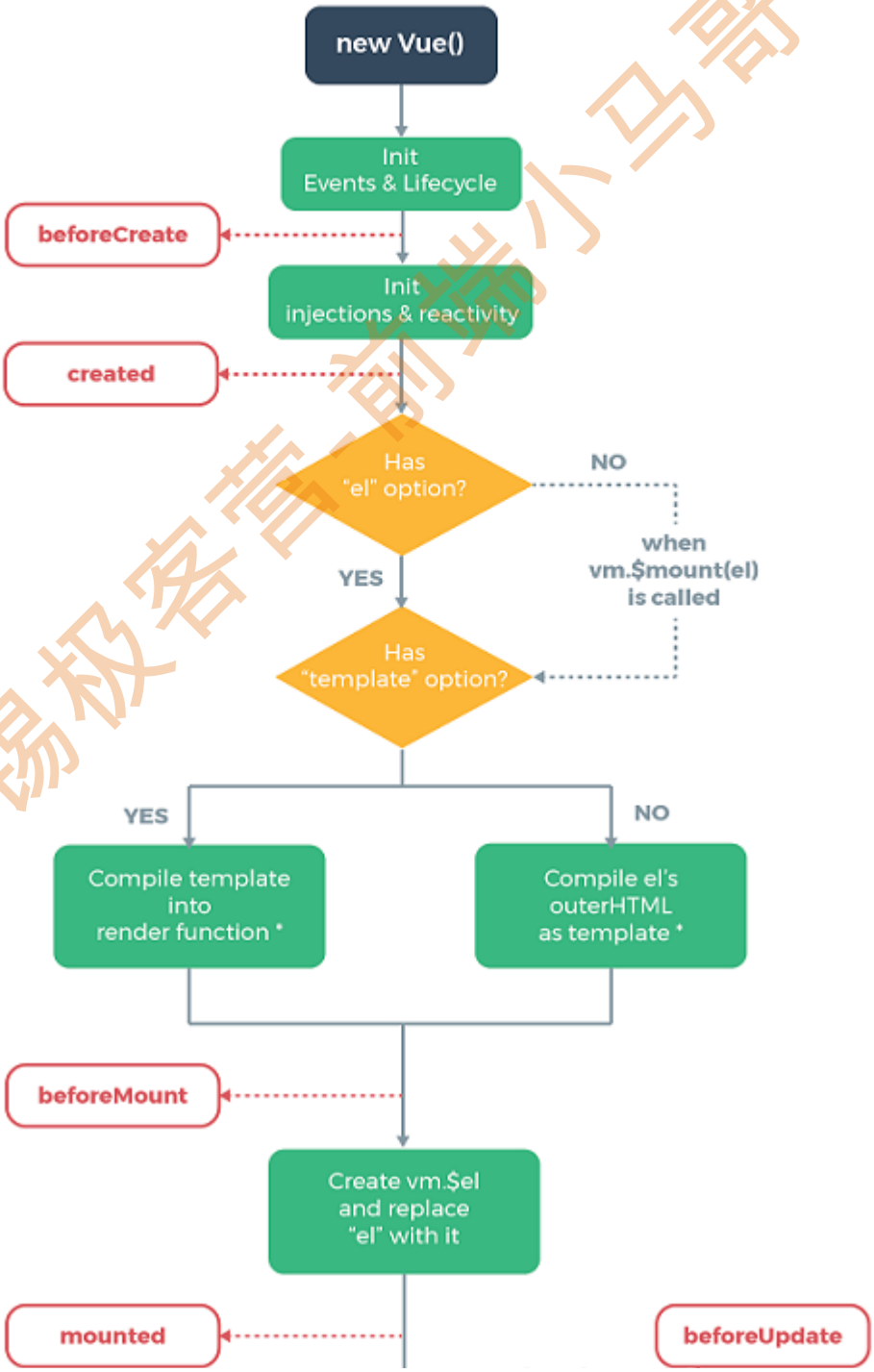
注意：

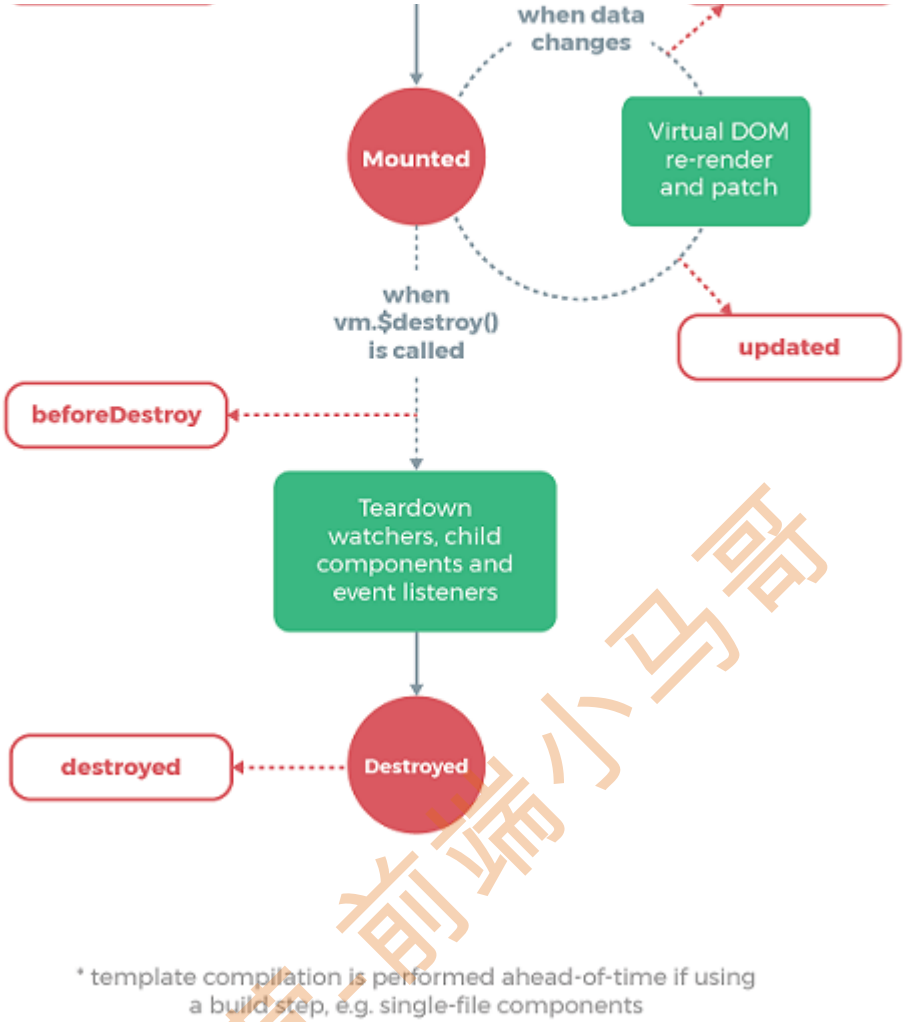
1、mounted、updated不会承诺所有的子组件也都一起被挂载。如果你希望等到整个视图都渲染完毕，可以用vm.$nextTick替换掉mounted、updated：

```
updated: function () {
    this.$nextTick(function () {
        // Code that will run only after the
        // entire view has been re-rendered
    })
}
```

2、http请求建议在 created 生命周期内发出

vue生命周期图示：





24.vue如何监听键盘事件中的按键

我们可以通过按键修饰符来监听键盘事件中的按键。`Vue`允许为`v-on`在监听键盘事件时添加`按键修饰符`：

```
<input v-on:keyup.enter="submit">

<!-- 缩写语法 -->
<input @keyup.enter="submit">
```

全部的按键别名：

```
- .enter
- .tab
- .delete (捕获"删除"和"退格"键)
- .esc
- .space
- .up
- .down
- .left
- .right
```

可以通过全局 `config.keyCodes` 对象`自定义按键修饰符别名`：

```
// 可以使用 `v-on:keyup.f1`
Vue.config.keyCodes.f1 = 112
```

系统修饰键：

> 2.1.0 新增

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。

```
- .ctrl
- .alt
- .shift
- .meta

<!-- Alt + C -->
<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->
<div @click.ctrl="doSomething">Do something</div>
```

> 2.5.0 新增

`.exact`修饰符允许你控制由精确的系统修饰符组合触发的事件。

```
<!-- 即使 Alt 或 Shift 被一同按下时也会触发 -->
<button @click.ctrl="onClick">A</button>

<!-- 有且只有 Ctrl 被按下的时候才触发 -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- 没有任何系统修饰符被按下的时候才触发 -->
<button @click.exact="onClick">A</button>
```

鼠标按钮修饰符：

> 2.2.0 新增

```
- .left
- .right
- .middle

这些修饰符会限制处理函数仅响应特定的鼠标按钮。
```

25.vue更新数组时触发视图更新的方法

Vue 包含一组观察数组的变异方法，所以它们也将会触发视图更新。这些方法如下：

- push()
- pop()
- shift()
- unshift()
- splice()
- sort()
- reverse()

当我们调用数组的上面这些方法修改数组的时候，页面会更新

filter(), concat()和 slice() 。这些不会改变原始数组，但总是返回一个新数组，所以页面不会更新。

26.vue中对象更改检测的注意事项

由于 JavaScript 的限制，Vue 不能检测对象属性的添加或删除：

```
var vm = new Vue({
  data: {
    a: 1
  }
})
// `vm.a`  响应式属性

vm.b = 2
// `vm.b`  非响应式属性
```

对于已经创建的实例，Vue 不能动态添加根级别的响应式属性。但是，可以使用 `Vue.set(object, key, value)`方法向嵌套对象`添加响应式属性`。例如，对于：

```
var vm = new Vue({
  data: {
    userProfile: {
      name: 'Anika'
    }
  }
})
```

你可以添加一个新的 age 属性到嵌套的 userProfile对象：

```
Vue.set(vm.userProfile, 'age', 27)
```

你还可以使用 vm.$set实例方法，它只是全局Vue.set 的别名：

```
vm.$set(vm.userProfile, 'age', 27)
```

有时你可能需要为已有对象赋予多个新属性，比如使用 `Object.assign()`或 `_.extend()`。在这种情况下，你应该用两个对象的属性创建一个新的对象。所以，如果你想`添加新的响应式属性`，不要像这样：

```
Object.assign(vm.userProfile, {
  age: 27,
  favoriteColor: 'Vue Green'
})
```

应该这样做：

```
vm.userProfile = Object.assign({}, vm.userProfile, {
  age: 27,
  favoriteColor: 'Vue Green'
})
```

27.如何解决非工程化项目，网速慢时初始化页面闪动问题？

使用`v-cloak`指令，`v-cloak`不需要表达式，它会在`Vue`实例结束编译时从绑定的HTML元素上移除，经常和CSS的`display:none`配合使用。

```
<div id="app" v-cloak>
        {{message}}
</div>

<script>
var app = new Vue({
    el:"#app",
    data:{
        message:"这是一段文本"
    }
})
</script>


[v-cloak]{
    display:none;
}
```

在一般情况下，`v-cloak`是一个解决初始化慢导致页面闪动的最佳实践，对于简单的项目很实用。

28.v-for产生的列表，如何实现active样式的切换？

通过设置当前 currentIndex 实现：

```
<template>
        <div class="toggleClassWrap">
         <ul>
            <!--li是否有clicked样式取决于当前的currentIndex等于多少-->
                <li @click="currentIndex = index" v-bind:class="{clicked: index ===
currentIndex}" v-for="(item, index) in desc" :key="index">
                        <a href="javascript:;">{{item.ctrlValue}}</a>
                </li>
```

```
        </ul>
        </div>
</template>

<script type="text/javascript">
        export default{
                data () {
                        return {
                                desc:[
                    {
                        ctrlValue:"test1"
                    },
                    {
                        ctrlValue:"test2"
                    },
                    {
                        ctrlValue:"test3"
                    },
                    {
                        ctrlValue:"test4"
                    }
                                ],
                                currentIndex:0
                        }
                }
        }
</script>

<style type="text/css" lang="less">
.toggleClassWrap{
        .clicked{
                color:red;
        }
}
</style>
```

29.vue-cli工作中如何自定义一个过滤器？

1.可以在组件内部定义私有过滤器

```
<script>
    // 创建 Vue 实例，得到 ViewModel
    var vm = new Vue({
      el: '#app',
      data: {
        msg: '曾经，我也是一个单纯的少年，单纯的我，傻傻的问，谁是世界上最单纯的男人'
      },
      methods: {},
          filters: {
        // 过滤器的三个参数：1.要过滤的文本    2.过滤的内容    3.替换的内容
        // 过滤器调用的时候，采用的是就近原则，如果私有过滤器和全局过滤器名称一致了，这时候优先调用私有过滤器
        msgFormat: function (msg, arg, arg2) {
```


```
            // 字符串的  replace 方法，第一个参数，除了可写一个 字符串之外，还可以定义一个正则
                    return msg.replace(/少年/g, arg + arg2)
                }
            },
        });
    </script>
```

2.可以使用Vue.filter定义全局过滤器

```
    // 定义一个 Vue 全局的过滤器，名字叫做  msgFormat
        Vue.filter('msgFormat', function (msg, arg, arg2) {
            // 字符串的  replace 方法，第一个参数，除了可写一个 字符串之外，还可以定义一个正则
            return msg.replace(/单纯/g, arg + arg2)
        })
```

3.在页面中使用管道修饰符来用过滤器

```
<p>{{ msg | msgFormat('疯狂+1', '123') | test }}</p>
```

30.vue-cli工作中常用的过滤器

创建一个filter.js，用来存放各种过滤器

```
//1.去除空格  type 1-所有空格  2-前后空格  3-前空格 4-后空格
function trim(value, trim) {
    switch (trim) {
        case 1:
            return value.replace(/\s+/g, "");
        case 2:
            return value.replace(/(^\s*)|(\s*$)/g, "");
        case 3:
            return value.replace(/(^\s*)/g, "");
        case 4:
            return value.replace(/(\s*$)/g, "");
        default:
            return value;
    }
}

//2.任意格式日期处理
//使用格式：
// {{ '2018-09-14 01:05' | formaDate(yyyy-MM-dd hh:mm:ss) }}
// {{ '2018-09-14 01:05' | formaDate(yyyy-MM-dd) }}
// {{ '2018-09-14 01:05' | formaDate(MM/dd) }} 等
function formaDate(value, fmt) {
    var date = new Date(value);
    var o = {
      "M+": date.getMonth() + 1, //月份
      "d+": date.getDate(), //日
      "h+": date.getHours(), //小时

      "m+": date.getMinutes(), //分
```

```
      "s+": date.getSeconds(), //秒
      "w+": date.getDay(), //星期
      "q+": Math.floor((date.getMonth() + 3) / 3), //季度
      "S": date.getMilliseconds() //毫秒
    };
    if (/(y+)/.test(fmt)) fmt = fmt.replace(RegExp.$1, (date.getFullYear() +
"").substr(4 - RegExp.$1.length));
    for (var k in o) {
      if(k === 'w+') {
        if(o[k] === 0) {
          fmt = fmt.replace('w', '周日');
        }else if(o[k] === 1) {
          fmt = fmt.replace('w', '周一');
        }else if(o[k] === 2) {
          fmt = fmt.replace('w', '周二');
        }else if(o[k] === 3) {
          fmt = fmt.replace('w', '周三');
        }else if(o[k] === 4) {
          fmt = fmt.replace('w', '周四');
        }else if(o[k] === 5) {
          fmt = fmt.replace('w', '周五');
        }else if(o[k] === 6) {
          fmt = fmt.replace('w', '周六');
        }
      }else if (new RegExp("(" + k + ")").test(fmt)) {
        fmt = fmt.replace(RegExp.$1, (RegExp.$1.length == 1) ? (o[k]) : (("00" +
o[k]).substr(("" + o[k]).length)));
      }
    }
    return fmt;
  }


//3.字母大小写切换
/*type
 1:首字母大写
 2：首页母小写
 3：大小写转换
 4：全部大写
 5：全部小写
 * */
function changeCase(str, type) {
    function ToggleCase(str) {
        var itemText = ""
        str.split("").forEach(
            function (item) {
                if (/^([a-z]+)/.test(item)) {
                    itemText += item.toUpperCase();
                } else if (/^([A-Z]+)/.test(item)) {
                    itemText += item.toLowerCase();
                } else {
                    itemText += item;

                }
```


```
            });
        return itemText;
    }
    switch (type) {
        case 1:
            return str.replace(/\b\w+\b/g, function (word) {
                return word.substring(0, 1).toUpperCase() +
word.substring(1).toLowerCase();
            });
        case 2:
            return str.replace(/\b\w+\b/g, function (word) {
                return word.substring(0, 1).toLowerCase() +
word.substring(1).toUpperCase();
            });
        case 3:
            return ToggleCase(str);
        case 4:
            return str.toUpperCase();
        case 5:
            return str.toLowerCase();
        default:
            return str;
    }
}

//4.字符串循环复制,count->次数
function repeatStr(str, count) {
    var text = '';
    for (var i = 0; i < count; i++) {
        text += str;
    }
    return text;
}

//5.字符串替换
function replaceAll(str, AFindText, ARepText) {
    raRegExp = new RegExp(AFindText, "g");
    return str.replace(raRegExp, ARepText);
}

//字符替换*，隐藏手机号或者身份证号等
//replaceStr(字符串,字符格式， 替换方式,替换的字符（默认*））
//ecDo.replaceStr('18819322663',[3,5,3],0)
//result：188*****663
//ecDo.replaceStr('asdasdasdaa',[3,5,3],1)
//result：***asdas***
//ecDo.replaceStr('1asd88465asdwqe3',[5],0)
//result：*****8465asdwqe3
//ecDo.replaceStr('1asd88465asdwqe3',[5],1,'+')
//result："1asd88465as+++++"

function replaceStr(str, regArr, type, ARepText) {

    var regtext = '',
```

```
        Reg = null,
        replaceText = ARepText || '*';
    //repeatStr是在上面定义过的（字符串循环复制），大家注意哦
    if (regArr.length === 3 && type === 0) {
        regtext = '(\\w{' + regArr[0] + '})\\w{' + regArr[1] + '}(\\w{' + regArr[2] +
'})'
        Reg = new RegExp(regtext);
        var replaceCount = this.repeatStr(replaceText, regArr[1]);
        return str.replace(Reg, '$1' + replaceCount + '$2')
    }
    else if (regArr.length === 3 && type === 1) {
        regtext = '\\w{' + regArr[0] + '}(\\w{' + regArr[1] + '})\\w{' + regArr[2] + '}'
        Reg = new RegExp(regtext);
        var replaceCount1 = this.repeatStr(replaceText, regArr[0]);
        var replaceCount2 = this.repeatStr(replaceText, regArr[2]);
        return str.replace(Reg, replaceCount1 + '$1' + replaceCount2)
    }
    else if (regArr.length === 1 && type === 0) {
        regtext = '(^\\w{' + regArr[0] + '})'
        Reg = new RegExp(regtext);
        var replaceCount = this.repeatStr(replaceText, regArr[0]);
        return str.replace(Reg, replaceCount)
    }
    else if (regArr.length === 1 && type === 1) {
        regtext = '(\\w{' + regArr[0] + '}$)'
        Reg = new RegExp(regtext);
        var replaceCount = this.repeatStr(replaceText, regArr[0]);
        return str.replace(Reg, replaceCount)
    }
}

//6.格式化处理字符串
//ecDo.formatText('1234asda567asd890')
//result："12,34a,sda,567,asd,890"
//ecDo.formatText('1234asda567asd890',4,' ')
//result："1 234a sda5 67as d890"
//ecDo.formatText('1234asda567asd890',4,'-')
//result："1-234a-sda5-67as-d890"
function formatText(str, size, delimiter) {
    var _size = size || 3, _delimiter = delimiter || ',';
    var regText = '\\B(?=(\\w{' + _size + '})+(?!\\w))';
    var reg = new RegExp(regText, 'g');
    return str.replace(reg, _delimiter);
}

//7.现金额大写转换函数
//ecDo.upDigit(168752632)
//result："人民币壹亿陆仟捌佰柒拾伍万贰仟陆佰叁拾贰元整"
//ecDo.upDigit(1682)
//result："人民币壹仟陆佰捌拾贰元整"
//ecDo.upDigit(-1693)
//result："欠人民币壹仟陆佰玖拾叁元整"

function upDigit(n) {
```


```
    var fraction = ['角', '分', '厘'];
    var digit = ['零', '壹', '贰', '叁', '肆', '伍', '陆', '柒', '捌', '玖'];
    var unit = [
        ['元', '万', '亿'],
        ['', '拾', '佰', '仟']
    ];
    var head = n < 0 ? '欠人民币' : '人民币';
    n = Math.abs(n);
    var s = '';
    for (var i = 0; i < fraction.length; i++) {
        s += (digit[Math.floor(n * 10 * Math.pow(10, i)) % 10] + fraction[i]).replace(/
零./, '');
    }
    s = s || '整';
    n = Math.floor(n);
    for (var i = 0; i < unit[0].length && n > 0; i++) {
        var p = '';
        for (var j = 0; j < unit[1].length && n > 0; j++) {
            p = digit[n % 10] + unit[1][j] + p;
            n = Math.floor(n / 10);
        }
        s = p.replace(/(零.)*零$/, '').replace(/^$/, '零') + unit[0][i] + s;
        //s = p + unit[0][i] + s;
    }
    return head + s.replace(/(零.)*零元/, '元').replace(/(零.)+/g, '零').replace(/^整$/,
'零元整');
}

//8.保留2位小数
function toDecimal2(x){
  var f = parseFloat(x);
  if (isNaN(f)) {
    return false;
  }
  var f = Math.round(x * 100) / 100;
  var s = f.toString();
  var rs = s.indexOf('.');
  if (rs < 0) {
    rs = s.length;
    s += '.';
  }
  while (s.length <= rs + 2) {
    s += '0';
  }
  return s;
}

export{
    trim,
    changeCase,
    repeatStr,
    replaceAll,

    replaceStr,
```


```
    checkPwd,
    formatText,
    upDigit,
    toDecimal2,
    formaDate
}
```

31.vue等单页面应用及其优缺点

单页Web应用（single page web application，SPA）：就是只有一个页面的应用。单页应用程序 (SPA) 是加载单个HTML 页面并在用户与应用程序交互时动态更新该页面的Web应用程序。浏览器一开始会加载必需的HTML、CSS和JavaScript，所有的操作都在这张页面上完成，都由JavaScript来控制。因此，对单页应用来说模块化的开发和设计显得相当重要。

单页Web应用的优点：
1、提供了更加吸引人的用户体验：具有桌面应用的即时性、网站的可移植性和可访问性。
2、单页应用的内容的改变不需要重新加载整个页面，web应用更具响应性和更令人着迷。
3、单页应用没有页面之间的切换，就不会出现“白屏现象”，也不会出现假死并有“闪烁”现象
4、单页应用相对服务器压力小，服务器只用出数据就可以，不用管展示逻辑和页面合成，吞吐能力会提高几倍。
5、良好的前后端分离。后端不再负责模板渲染、输出页面工作，后端API通用化，即同一套后端程序代码，不用修改就可以用于Web界面、手机、平板等多种客户端。

单页Web应用的缺点：
1、首屏加载慢
2、SEO问题，不利于百度，360等搜索引擎收录。
3、容易造成CSS命名冲突。
4、前进、后退、地址栏、书签等，都需要程序进行管理，页面的复杂度很高，需要一定的技能水平和开发成本高。

32.什么是vue的计算属性？

计算属性：对于任何复杂的计算逻辑，当前属性值是根据其他属性计算出来的，都应当使用计算属性。

例子：

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>

var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // 计算属性的 getter
    reversedMessage: function () {
      // `this` 指向 vm 实例
      return this.message.split('').reverse().join('')
    }
```


```
  }
})
```

33.vue-cli提供的几种脚手架模板

> 在使用之前，可以先用 vue-list命令查询可用的模板。

```
[hanxumingdeMacBook-Pro:~ hanxuming$ vue list

  Available official templates:

  ★  browserify - A full-featured Browserify + vueify setup with hot-reload, lin
ting & unit testing.
  ★  browserify-simple - A simple Browserify + vueify setup for quick prototypin
g.
  ★  pwa - PWA template for vue-cli based on the webpack template
  ★  simple - The simplest possible Vue setup in a single HTML file
  ★  webpack - A full-featured Webpack + vue-loader setup with hot reload, linti
ng, testing & css extraction.
  ★  webpack-simple - A simple Webpack + vue-loader setup for quick prototyping.
```

vue-cli提供了的常用的脚手架模板：
1.webpack：基于 webpack 和 vue-loader 的目录结构，而且支持热部署、代码检查、测试及 css 抽取。
2.webpack-simple：基于 webpack 和 vue-loader 的目录结构。
3.browerify：基于 Browerfiy 和 vueify(作用于 vue-loader 类似)的结构，支持热部署、代码检查及单元测试。
4.browerify-simple：基于 Browerfiy 和 vueify 的结构。
5.simple：单个引入 Vue.js 的 index.html 页面。

这里我们主要会使用 webpack 作为常用脚手架，可以运行vue init webpack my-project 来生成项目。

34.vue父组件如何向子组件中传递数据

1.父组件中使用子组件的时候通过 v-bind 向子组件传递数据：

```
<!-- 动态赋予一个变量的值 -->
<blog-post v-bind:title="post.title"></blog-post>
```

2.子组件中声明props来接收父组件传递过来的数据：

```
export default {
  props : ["title"]
}

//或者
export default {
  props : {
        title:{
        type:string,
        default:""
    }
  }
}
```

34.如何在组件中使用全局常量

第一步，在 src 下新建 const 文件夹下 新建 const.js，并在const.js中设置常量

```
.
├── src
|   ├── const
|   |    ├── const.js
|   |
|   └── main.js
└── ...


//const.js
export default {
    install(Vue,options){
        Vue.prototype.global = {
            title:'全局',
            isBack: true,
            isAdd:  false,
        };
    }
}
```

第二步，在 main.js 下全局引入：

```
//引入全局常量
import constant from './const/const.js'
Vue.use(constant);
```

第三步，即可在 .vue 组件中使用：

```
//通过js方式使用：
this.global.title
//或在 html 结构中使用
{{global.title}}
```


35.vue如何禁止弹窗后面的滚动条滚动

1.设置document的overflow为hidden

2.给document绑定touchmove事件，阻止默认事件

```
methods : {
    //禁止滚动
    stop(){
         var mo=function(e){e.preventDefault();};
         document.body.style.overflow='hidden';
         document.addEventListener("touchmove",mo,false);//禁止页面滑动
     },
     //取消滑动限制
     move(){
         var mo=function(e){e.preventDefault();};
         document.body.style.overflow='';//出现滚动条
         document.removeEventListener("touchmove",mo,false);
     }
}
```

36.请说出计算属性(computed)的缓存和方法(method)调用的有什么区别？

1. 计算属性必须返回结果
2. 计算属性是基于它的依赖缓存的。一个计算属性所依赖的数据发生变化时，它才会重新取值。
3. 使用计算属性还是methods取决于是否需要缓存，当遍历大数组和做大量计算时，应当使用计算属性，除非你不希望得到缓存。
4. 计算属性是根据依赖自动执行的，methods需要事件调用。

37.什么是vue.js中的自定义指令

Vue里面有许多内置的指令，比如 `v-if` 和 `v-show` ，这些丰富的指令能满足我们的绝大部分业务需求，不过在需要一些特殊功能时，我们仍然希望对 `DOM` 进行底层的操作，这时就要用到自定义指令。

```
Vue.directive('focus', {
    bind: function (el) {
        // 每当指令绑定到元素上的时候，会立即执行这个bind 函数，只执行一次
        // 注意： 在每个函数中，第一个参数永远是el，表示被绑定了指令的那个元素，这个el参数，是一个原生的JS对象
        // 在元素刚绑定了指令的时候还没有插入到DOM中去的时候调用focus方法没有作用。因为，一个元素只有插入DOM之后才能获取焦点
        // el.focus()
    },
    inserted: function (el) {
        // inserted 表示元素 插入到DOM中的时候，会执行 inserted 函数【触发1次】。和JS行为有关的操作，最好在 inserted 中去执行，放置 JS行为不生效
        el.focus()
    },
    updated: function (el) {
        // 当VNode更新的时候，会执行 updated， 可能会触发多次
    }
})
```

38.自定义指令的几个钩子函数

```
- bind：只调用一次，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：被绑定元素插入父节点时调用 （仅保证父节点存在，但不一定已被插入文档中）。
- update：所在组件的 VNode 更新时调用，但是可能发生在其子 VNode 更新之前。指令的值可能发生了改变，也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 。
- componentUpdated：指令所在组件的 VNode 及其子 VNode 全部更新后调用。
- unbind：只调用一次，指令与元素解绑时调用。

bind(vnode,binding){
    //vnode是绑定指定的那个元素
    //binding里面包含指定所绑定的信息
}
```

39.自定义指令钩子函数参数

> 在自定义指令钩子函数的参数中，除了 el 之外，其它参数都应该是只读的，切勿进行修改。如果需要在钩子之间共享数据，建议通过元素的 dataset 来进行。

指令钩子函数会被传入以下参数：

- `el`：指令所绑定的元素，可以用来直接操作 DOM 。
- `binding`：一个对象，包含以下属性：
  - name：指令名，不包括 v- 前缀。
  - value：指令的绑定值，例如：`v-my-directive="1 + 1"` 中，绑定值为 `2`。
  - oldValue：指令绑定的前一个值，仅在 `update` 和 `componentUpdated` 钩子中可用。无论值是否改变都可用。
  - expression：字符串形式的指令表达式。例如 `v-my-directive="1 + 1"` 中，表达式为 `"1 + 1"`。
  - arg：传给指令的参数，可选。例如 `v-my-directive:foo` 中，参数为 "foo"。
  - modifiers：一个包含修饰符的对象。例如：`v-my-directive.foo.bar` 中，修饰符对象为 `{ foo: true, bar: true }`。
- `vnode`：Vue 编译生成的虚拟节点。
- `oldVnode`：上一个虚拟节点，仅在 `update` 和 `componentUpdated` 钩子中可用。

40.vue-router如何响应 路由参数 的变化

> 问题：当使用路由参数时，例如从 `/content?id=1` 到 `content?id=2`，此时原来的组件实例会被复用。这也意味着组件的`生命周期钩子不会再被调用`，此时vue应该如何响应`路由参数`的变化？

解决方案：

1.用 :key 来阻止“复用”

```
//在父组件中使用
<router-view :key="key"></router-view>

computed: {
        key() {
                return this.$route.name !== undefined? this.$route.name +new Date():
this.$route +new Date()
        }
}

//这种办法实质上是让每次路由跳转时重新构建该组件，我们在它的生命周期中写一个打印语句就能看出来。
```

2.复用组件时，想对路由参数的变化作出响应的话， 可以`watch （监测变化） $route 对象`：

```
const User = {
  template: '...',
  watch: {
    '$route' (to, from) {
      // 对路由变化作出响应...
    }
  }
}
```

3.通过`vue-router`的钩子函数`beforeRouteEnter` `beforeRouteUpdate` `beforeRouteLeave`

```
//localhost:3000/login/1   --->localhost:3000/login/2
beforeRouteEnter (to, from, next) {
    // 在渲染该组件的对应路由被 confirm 前调用
    // 不！能！获取组件实例 `this`
    // 因为当钩子执行前，组件实例还没被创建
},
beforeRouteUpdate (to, from, next) {
    // 在当前路由改变，但是该组件被复用时调用
    // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，
    // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。
    // 可以访问组件实例 `this`
},
beforeRouteLeave (to, from, next) {
    // 导航离开该组件的对应路由时调用
    // 可以访问组件实例 `this`
}
```

41.完整的 vue-router 导航解析流程

```
当由A路由 --> B路由的时候：
1、在A组件里调用离开守卫。            A组件中的  beforeRouteLeave
2、调用全局的 beforeEach 守卫。      router.beforeEach
3、再执行B路由配置里调用 beforeEnter。
      routes: [
        {
```


```
            path: '/b',
            component: B,
            beforeEnter: (to, from, next) => {
            }
        }
    ]
```

4、再执行B组件的进入守卫。 B组件中 beforeRouteEnter。
5、调用全局的 beforeResole 守卫 (2.5+)。 router.beforeResolve
6、导航被确认。
7、调用全局的 afterEach 钩子。 router.afterEach
8、触发 DOM 更新。

42.vue-router有哪几种导航钩子（ 导航守卫 ）？

1、全局守卫： router.beforeEach router.beforeResolve router.afterEach

```
const router = new VueRouter({ ... });
router.beforeEach((to, from, next) => {
    // do someting
});
//to:代表要进入的目标，它是一个路由对象
//from:代表当前正要离开的路由，同样也是一个路由对象
//next:这是一个必须需要调用的方法，而具体的执行效果则依赖 next 方法调用的参数

//全局后置钩子，后置钩子并没有 next 函数，也不会改变导航本身
router.afterEach((to, from) => {
    // do someting
});
```

2、路由独享的守卫： beforeEnter

```
cont router = new VueRouter({
    routes: [
        {
            path: '/file',
            component: File,
            beforeEnter: (to, from ,next) => {
                // do someting
            }
        }
    ]
});
```

3、组件内的守卫： beforeRouteEnter、beforeRouteUpdate (2.2 新增)、beforeRouteLeave

```
const File = {
    template: `<div>This is file</div>`,
    beforeRouteEnter(to, from, next) {
        // do someting
        // 在渲染该组件的对应路由被 confirm 前调用
    },
    beforeRouteUpdate(to, from, next) {
        // do someting
        // 在当前路由改变，但是依然渲染该组件是调用
    },
    beforeRouteLeave(to, from ,next) {
```

```
            // do someting
            // 导航离开该组件的对应路由时被调用
        }
    }
```

43.vue-router的几种实例方法以及参数传递

实例方法	说明
this.$router.push(location, onComplete?, onAbort?)	这个方法会向 history 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的 URL。并且点击 `<router-link :to="...">` 等同于调用 `router.push(...)`。
this.$router.replace(location, onComplete?, onAbort?)	这个方法不会向 history 添加新记录，而是跟它的方法名一样 —— 替换掉当前的 history 记录，所以，当用户点击浏览器后退按钮时，并不会回到之前的 URL。
this.$router.go(n)	这个方法的参数是一个整数，意思是在 history 记录中向前或者后退多少步，类似 `window.history.go(n)`。

参数传递方式：vue-router提供了`params`、`query`、`meta`三种页面间传递参数的方式。

```
// 字符串，不带参数
this.$router.push('home')

// 对象，不带参数
this.$router.push({ path: 'home' })

// params（推荐）：命名的路由，params 必须和 name 搭配使用
this.$router.push({ name:'user',params: { userId: 123 }})

// 这里的 params 不生效
this.$router.push({ path:'/user',params: { userId: 123 }})

// query：带查询参数，变成 /register?plan=private
this.$router.push({ path: 'register', query: { plan: 'private' }})

//meta方式：路由元信息
export default new Router({
    routes: [
        {
            path: '/user',
            name: 'user',
            component: user,
            meta:{
                title:'个人中心'
            }
        }
```


```
    ]
})
```

在组件中获取路由参数：

```
//通过 $route 对象获取，注意是route，不是$router
this.$route.params
this.$route.query
this.$route.meta
```

44.*route*和router 的区别

`$route`是“路由信息对象”，包括path，params，hash，query，fullPath，matched，name等路由信息参数。

`$router`是“路由实例”对象包括了路由的跳转方法，钩子函数等。

45.vue-router的动态路由匹配以及使用

动态路径匹配：即把某种模式匹配到的所有路由，全都映射到同个组件。使用动态路由参数来实现。

```
const User = {
  template: '<div>User</div>'
}

const router = new VueRouter({
  routes: [
    // 动态路径参数 以冒号开头
    { path: '/user/:id', component: User }
  ]
})

//  这样，像/user/foo和/user/bar都将映射到相同的路由。
```

```
//一个“路径参数”使用冒号 :标记。当匹配到一个路由时，参数值会被设置到 this.$route.params，可以在每个组件内使用。

const User = {
  template: '<div>User {{ $route.params.id }}</div>'
}
```

46.vue-router如何定义嵌套路由

嵌套路由：是路由的多层嵌套。

第一步：需要在一个被渲染的组件中嵌套`<router-view>`组件用于呈现子路由。

```
const User = {
  template: `
    <div class="user">
      <h2>User</h2>
      <router-view></router-view>
    </div>
  `
}
```

第二步：在嵌套的出口中渲染组件，在 `VueRouter` 的参数中使用 `children` 配置：

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:id',
      component: User,
      children: [
        {
          // 当 /user/:id/profile 匹配成功，
          // UserProfile 会被渲染在 User 的 <router-view> 中
          path: 'profile',
          component: UserProfile
        },
        {
          // 当 /user/:id/posts 匹配成功
          // UserPosts 会被渲染在 User 的 <router-view> 中
          path: 'posts',
          component: UserPosts
        }
      ]
    }
  ]
})
```

47. `<router-link></router-link>` 组件及其属性

`<router-link>` 组件：用于支持用户在具有路由功能的应用中 (点击)跳转导航。

可以通过 `to` 属性 指定目标地址，默认渲染成带有正确链接的 `<a>` 标签，可以通过配置 `tag` 属性 生成别的标签.。

另外，当目标路由成功激活时，链接元素自动设置一个表示激活的 CSS 类名。

常用属性：

属性	类型	说明	示例
to	string \ Location	表示目标路由的链接。当被点击后，内部会立刻把 `to` 的值传到 `router.push()` ，所以这个值可以是一个字符串或者是描述目标位置的对象。	`<router-link to="home">Home</router-link>`
replace	boolean (默认flase)	设置 `replace` 属性的话，当点击时，会调用 `router.replace()` 而不是 router.push()，于是导航后 不会留下 `history` 记录。	`<router-link :to="{ path: '/abc'}" replace></router-link>`
append	boolean (默认flase)	设置 `append` 属性后，则在当前 (相对) 路径前添加基路径。例如，我们从 `/a` 导航到一个相对路径 `b` ，如果 没有配置 `append` ，则路径为 `/b` ，如果 配了 ，则为 `/a/b`	`<router-link :to="{ path: 'relative/path'}" append></router-link>`
tag	string (默认 'a')	有时候想要 渲染成某种标签，例如 `<li>` 于是我们使用 `tag` prop 类指定何种标签，同样它还是 会监听点击 ，触发导航。	`<router-link to="/foo" tag="li">foo</router-link>`
active-class	string (默认 "router-link-active")	设置 链接激活时 使用的 `CSS` 类名。默认值可以通过路由的构造选项 `linkActiveClass` 来全局配置。	
exact	boolean (默认 false)	"是否激活" 默认类名的依据是 `inclusive match` （全包含匹配）。 举个例子，如果当前的路径是 /a 开头的，那么 也会被设置 CSS 类名。	这个链接只会在地址为 / 的时候被激活： `<router-link to="/" exact>`
event	string \ Array （默认 'click' ）	声明可以用来 触发导航的事件 。可以是一个 字符串 或是一个包含 字符串的数组 。	
exact-active-class	string 默认 'router-link-exact-active'	配置当链接被精确匹配的时候应该激活的 `class` 。注意默认值也是可以通过路由构造函数选项 `linkExactActiveClass` 进行全局配置的。	

48.vue-router实现动态加载路由组件（ 懒加载 ）

当打包构建应用时，Javascript 包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

结合 Vue 的异步组件和 Webpack 的代码分割功能，轻松实现路由组件的懒加载。

第一步：定义一个能够被 Webpack 自动代码分割的异步组件。

```
//在src/router/index.js里面引入异步引入组件
const index = () => import('../page/list/index.vue');
```

第二步：在路由配置中什么都不需要改变，只需要像往常一样使用 index。

```
const router = new VueRouter({
  routes: [
    { path: '/index', component: index,name:"index" }
  ]
})
```

第三步：在build/webpack.base.conf.js下的output属性，新增chunkFilename。

```
output: {
    path: config.build.assetsRoot,
    filename: '[name].js',
    //新增chunkFilename属性
    chunkFilename: '[name].js',
    publicPath: process.env.NODE_ENV === 'production'
      ? config.build.assetsPublicPath
      : config.dev.assetsPublicPath
  },
```

49.什么是vuex？

Vuex 是一个专为 Vue.js 应用程序开发的状态管理器，采用 集中式存储 管理所有组件的状态。

vuex的原理其实非常简单，它为什么能实现所有的组件共享同一份数据？
因为vuex生成了一个store实例，并且把这个实例挂在了所有的组件上，所有的组件引用的都是同一个store实例。
store实例上有数据，有方法，方法改变的都是store实例上的数据。由于其他组件引用的是同样的实例，所以一个组件改变了store上的数据，导致另一个组件上的数据也会改变，就像是一个对象的引用。

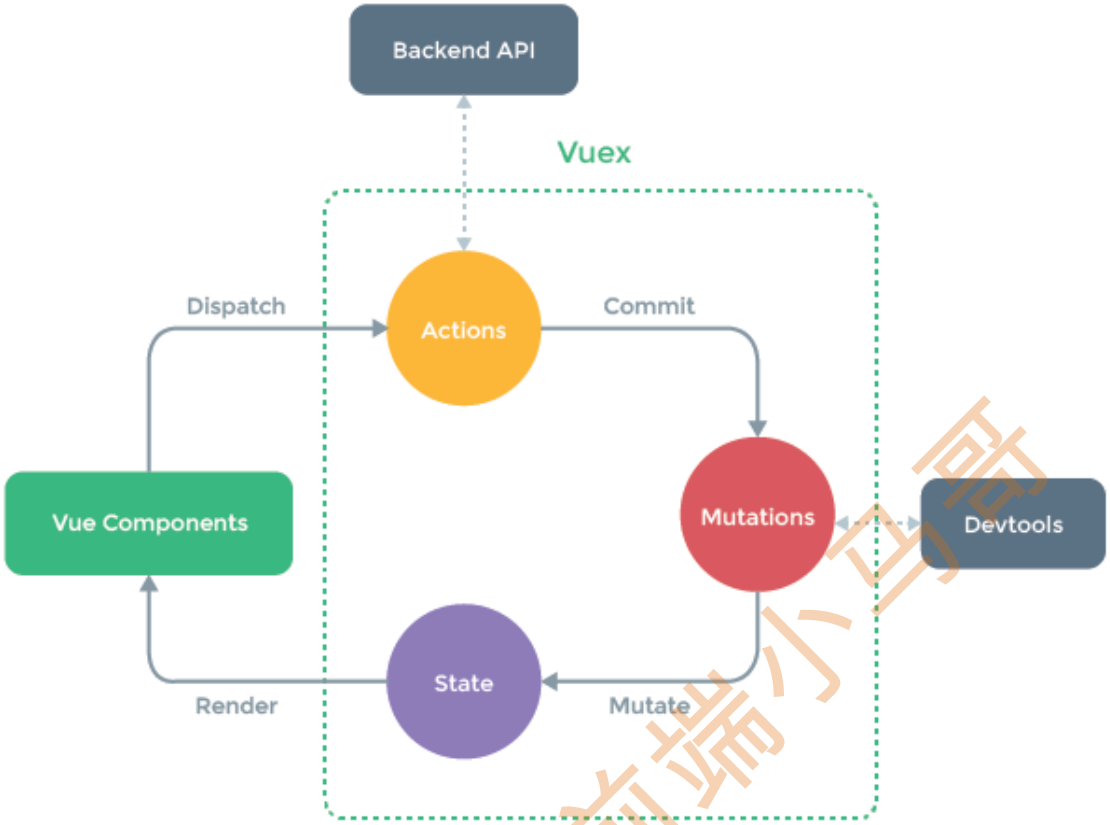
50.使用vuex的核心概念

每一个 Vuex 应用的核心就是 store（仓库）。“store”基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。

//vuex的核心概念和核心概念图：
1、state - Vuex store实例的根状态对象，用于定义共享的状态变量。
2、Action -动作，向store发出调用通知，执行本地或者远端的某一个操作（可以理解为store的methods）
3、Mutations -修改器，它只用于修改state中定义的状态变量。
4、getter -读取器，外部程序通过它获取变量的具体值，或者在取值前做一些计算（可以认为是store的计算属性）

//Vuex的应用场景：
Vuex主要用于：
1、多层嵌套的组件之间进行状态传递
2、兄弟组件间进行状态传递时（当然也可以使用中央事件总线BUS）
3、多组件共享状态时

更为具体的场景：组件之间的状态、音乐播放、登录状态、加入购物车...




```
//vuex的使用

#1.安装vuex
 npm i vuex -S

#2.main.js
import Vue from 'vue'
import Vuex from 'vuex'

// 注册vuex到vue中
Vue.use(Vuex)

// new Vuex.Store()实例，得到一个数据仓储对象
// 可以在组件中通过this.$store.state.xx 来访问store中的数据
var store = new Vuex.Store({
    //state相当于组件中的data
    state: {
        count: 0
    },
    //如果要修改store中state的值，需要调用 mutations提供的方法，可以通过this.$store.commit('方法名')来调用
    mutations: {

        increment(state) {
```

```
            state.count++
        },
        //mutations函数参数列表中最多支持两个参数，其中参数1是state； 参数2是通过commit提交过来的参数；
        subtract(state, obj) {
            console.log(obj)
            state.count -= obj.step;
        }
    },
    getters: {
        //这里的getters只负责对外提供数据，不负责修改数据，如果想要修改 state 中的数据需要在mutations中修改
        optCount: function (state) {
            return '当前最新的count值是：' + state.count
        }
    }
})

// 总结：
// 1. state中的数据，不能直接修改，如果想要修改，必须通过 mutations
// 2. 如果组件想要直接 从 state 上获取数据： 需要 this.$store.state.***
// 3. 如果组件想要修改数据，必须使用 mutations 提供的方法，需要通过 this.$store.commit('方法的名称'， 唯一的一个参数)
// 4. store中state上的数据在对外提供的时候建议做一层包装，推荐使用 getters。调用的时候则用 this.$store.getters.***


import App from './App.vue'

const vm = new Vue({
    el: '#app',
    render: c => c(App),
    //将vuex创建的store挂载到VM实例上，只要挂载到了 vm 上，任何组件都能使用store来存取数据
    store
})

#3.index.html
<body>
    <div id="app"></div>
</body>

#4.App.vue
<template>
    <div>
        <h1>这是 App 组件</h1>
        <hr>
        <counter></counter>
        <hr>
        <amount></amount>
    </div>
</template>


<script>
```

```
    import counter from "./components/counter.vue";
    import amount from "./components/amount.vue";

    export default {
        data() {
            return {};
        },
        components: {
            counter,
            amount
        }
    };
</script>

#5.components/amount.vue
<template>
  <div>
    <h3>{{ $store.getters.optCount }}</h3>
  </div>
</template>

#6.components/counter.vue
<template>
  <div>
    <input type="button" value="绑定事件-减少" @click="sub">
    <input type="button" value="绑定事件-增加" @click="add">
    <br>
    <input type="text" v-model="$store.state.count">
  </div>
</template>

<script>
export default {
  data() {
    return {
    };
  },
  methods: {
    add() {
       this.$store.commit("increment");
    },
    sub() {
      this.$store.commit("subtract",{ step:3});
    }
  }
};
</script>
```


51.如何在vuex中使用异步修改？

```
//我们可以在vuex的action中实现对vuex的异步修改

const actions = {
    asyncInCrement({ commit }, n){
        return new Promise(resolve => {
            setTimeout(() => {
                commit(types.TEST_INCREMENT, n);
                resolve();
            },3000)
        })
    }
}
```

52.Promise对象是什么？

Promise对象是ES6（ ECMAScript 2015 ）对于异步编程提供的一种解决方案，比传统的解决方案—回调函数和事件—更合理和更强大。
Promise本身不是异步的，只不过Promise中可以有异步任务，new Promise()的第一个函数参数是立马执行的。

```
function func1(a){
    return new Promise((resolve,reject) => {
        if(a > 10){
            resolve(a)
        }else{
            reject(b)
        }
    })
};

func1('11').then(res => {
    console.log('success');
}).catch(err => {
    console.log('error');
})


//Promise构造函数接受一个函数作为参数，该函数的两个参数分别resolve 和 reject。它们是两个函数，由 JavaScript 引擎提供。
1.resolve函数的作用是： 将Promise对象的状态从"未完成"变为"成功"（即从 pending 变为 fulfilled），在异步操作成功时调用，并将异步操作的结果，作为参数传递出去；

2.reject函数的作用是： 将Promise对象的状态从"未完成"变为"失败"（即从 pending 变为 rejected），在异步操作失败时调用，并将异步操作报出的错误，作为参数传递出去。

//Promise对象实例的方法，then 和 catch：
1   .then方法： 用于指定调用成功时的回调函数。
    then方法返回的是一个新的Promise实例（注意，不是原来那个Promise实例），因此可以采用链式写法，即then方法后面再调用另一个then方法。

2   .catch方法： 用于指定发生错误时的回调函数。
```


53.axios、fetch与ajax有什么区别？

```
//1.Ajax
```

Ajax指的是XMLHttpRequest（XHR），最早出现的发送后端请求技术，核心使用XMLHttpRequest对象，如果多个请求之间如果有先后关系的话，就会出现回调地狱。

JQuery ajax 是对原生XHR的封装，除此以外还增添了对JSONP的支持。经过多年的更新维护，真的已经是非常的方便了，优点无需多言

Ajax的缺点：

1.本身是针对MVC的编程，不符合现在前端MVVM的浪潮
2.基于原生的XHR开发，XHR本身的架构不清晰。
3.JQuery整个项目太大，单纯使用ajax却要引入整个JQuery非常的不合理（采取个性化打包的方案又不能享受CDN服务）
4.容易出现回调地狱的问题
5.不符合关注分离的原则

```
//2.axios
```

Vue2.0之后，尤雨溪推荐我们使用axios替换JQuery ajax。
axios 是一个基于Promise的请求库，用于浏览器和nodejs中，本质上浏览器的axios也是对原生XHR的封装，只不过它是Promise的实现版本，符合最新的ES规范，它本身具有以下特征：

1.从浏览器中创建 XMLHttpRequest
2.支持 Promise API
3.客户端支持防止CSRF
4.提供了一些并发请求的接口（重要，方便了很多的操作）
5.从node.js创建 http 请求
6.拦截请求和响应
7.转换请求和响应数据
8.取消请求
9.自动转换JSON数据

```
//3.fetch
```

fetch号称是AJAX的替代品，是在ES6出现的，使用了ES6中的promise对象。Fetch是基于promise设计的，Fetch的代码结构比起ajax简单多了，参数有点像jQuery ajax。但是一定记住fetch不是ajax的进一步封装，而是原生js，没有使用XMLHttpRequest对象。

fetch的优点：

1.符合关注分离，没有将输入、输出和用事件来跟踪的状态混杂在一个对象里
2.更好更方便的写法
3.基于标准 Promise 实现，支持 async/await
4.更加底层，提供的API丰富（request, response）
5.脱离了XHR，是ES6规范里新的实现方式

54.axios有什么特点？

```
1、Axios 是一个基于 promise 的 HTTP 库，支持promise所有的API
2、它可以拦截请求和响应
3、它可以转换请求数据和响应数据，并对响应回来的内容自动转换成 JSON类型的数据
4、安全性更高，客户端支持防御 XSRF
5、在客户端和服务器均可使用
```

55.vue组件的scoped属性的作用

```
当 <style> 标签有scoped 属性时，它的 CSS 只作用于当前组件中的元素
你可以在一个组件中同时使用有 scoped 和 非scoped 样式：


<style>
/* 全局样式 */
</style>

<style scoped>
/* 本地样式 */
</style>
```

56.vue中集成的UI组件库

```
常用的UI组件库有：
vux： Vue.js 移动端 UI 组件库
Amaze ~ 妹子 UI
Element：饿了么组件库，适用于开发应用后台
mint-ui：移动端 UI 组件库
Ant-design ： 阿里的UI组件库


1.安装第三方组件库
2.引入组件库的包      Vue.use()
3.引入组件的css文件
4.使用
5.按需加载     安装babel插件     配置babelrc文件      用什么组件引入什么组件
```

57.如何适配移动端？【 经典 】

在css样式兼容性方面，我们可以使用autoprefixer插件

```
postcss可以被理解为一个平台，可以让一些插件在上面跑。它提供了一个解析器，可以将CSS解析成抽象语法树。通过PostCSS这个平台，我们能够开发一些插件，来处理CSS。热门插件如autoprefixer。

vue-cli已经自动集成了postcss，所以我们可以在postcss.config.js这个配置文件中直接添加autoprefixer这个插件
```

在屏幕大小适配方面，我们可以使用下面两种方案

a) 使用flexible和 postcss-px2rem

之前使用rem适配的思路：使用媒体查询，确定不同屏幕下html标签的font-size （即1rem单位），然后在写css样式的时候，就可以使用1rem ， 2rem这样的单位来做适配了。

但是这种适配的问题是需要手动把px单位换算成rem单位，比较麻烦。对应的，我们可以使用flexible和 postcss-px2rem来解决这个问题

```
//1.安装flexible。 flexible主要是实现在各种不同的移动端界面实现一稿搞定所有的设备兼容自适应问题
npm install lib-flexible --save

//2.main.js引入flexible
import 'lib-flexible'

//此时运行程序会看到html中自动加上了font-size    font-size的默认值为viewport的十分之一
//在页面中引入flexible.js后，flexible会在<html>标签上增加一个data-dpr属性和font-size样式（如下图）。

//3.安装postcss-pxtorem
npm install postcss-pxtorem --save-dev

//4.修改postcss.config.js
module.exports = {
  plugins: {
    //autoprefixer 自动补全css前缀的东西
    'autoprefixer': {
      //兼容的机型
      browsers: ['Android >= 4.0', 'iOS >= 7']
    },
    'postcss-pxtorem': {
      rootValue: 37.5, //换算基数，一般和html的font-size一致
      propList: ['*']  //哪些css属性需要换算
    }
  }
};
```

b) 使用postcss-px-to-viewport

`vw` 与 `vh`单位，以`viewport`为基准，`1vw` 与 `1vh`分别为`window.innerWidth` 与 `window.innerHeight`的百分之一。

vw/vh 单位其实出现比较早了，只是以前支持性不太好，现在随着浏览器的发展，大部分（92%以上）的浏览器已经支持了vw/vh

```
npm i postcss-px-to-viewport -save -dev

//修改postcss.config.js
module.exports = {
  plugins: {
      autoprefixer: {
          //兼容的机型

          browsers: ['Android >= 4.0', 'iOS >= 7']
```

```
      },
    //px转换为vw单位的插件
    "postcss-px-to-viewport": {
      //1vw = 3.2
      viewportWidth: 320,
      //1vh = 5.68
      viewportHeight: 568,
      // px to vw无法整除时，保留几位小数
      unitPrecision: 5,
      // 转换成vw单位
      viewportUnit: 'vw',
     //不转换的类名
      selectorBlackList: [],
      // 小于1px不转换
      minPixelValue: 1,
     //允许媒体查询中转换
      mediaQuery: false,
      //排除node_modules文件中第三方css文件
      exclude: /(\/|\\)(node_modules)(\/|\\)/
    },
  }
};
```

58.vue-cli工程中如何使用背景图？

> 第一种方法：通过 `import` 引入

首先，引入要使用的背景图片：

```
<script type="text/javascript">
    import  cover  from  "../assets/images/cover.png";
    export  default{
        ...
    }
</script>
```

然后，通过 `v-bind:style` 使用：

```
<div :style="{ backgroundImage:'url(' + cover + ')' }"></div>
```

> 第二种方法：通过 `require` 引入：

直接通过 `v-bind` 和 `require` 配合使用

```
<div :style="{ backgroundImage:'url(' + require('../assets/images/couver.png') + ')' }">
</div>
```

如果在css文件中使用图片作为背景，可以直接 background:url(../../logg.png)

59.vue中如何实现tab切换功能？

在 vue 中，实现 Tab 切换主要有三种方式：
1、使用 component 动态组件实现 Tab切换 [推荐移动端使用]
<component :is="组件名字"></component>
2、使用 vue-router 路由配合<router-view></router-view>标签实现
3、使用第三方组件
4、还可以通过v-if和v-show来完成

60.vue中如何利用<keep-alive></keep-alive>标签实现某个组件缓存功能？

vue-cli工程中实现某个组件的缓存功能，可用 keep-alive 标签与 vue-router的meta形式数据传递配合完成。

```
<keep-alive include="组件名字">
  <router-view></router-view>
</keep-alive>
```

```
//第一步：在 app.vue 里面 template部分 使用 <keep-alive></keep-alive> 组件：

<template>
  <div id="app">
        <keep-alive>
            <router-view v-if="$router.meta.keepAlive"></router-view>
        </keep-alive>
        <router-view v-if="!$router.meta.keepAlive"></router-view>
  </div>
</template>


//第二步：在src/router.js:
import account from '../page/demo/account.vue'
import course from '../page/demo/course.vue'

export default new Router({
  routes: [
        {
            path: '/account',
            name: 'account',
            component: Account,
            meta:{
                keepAlive:false //false为不缓存
            }
        },
        {
            path: '/course',
            name: 'course',
            component: course,
            meta:{
                keepAlive:true //true为缓存
            }
        }
    ]

})
```

61.vue中实现切换页面时为左滑出效果

左滑效果实现，需要使用 `<transition></transition>` 组件配合 css3 动画效果实现。

```
<div id="app">
      <!-- 使用transiton来规定页面切换时候的样式-->
      <transition name="slide-left">
        <router-view></router-view>
      </transition>
</div>

<style lang="less">
  /*左滑动效*/
   .slide-left-enter-active {
     animation: slideLeft 0.3s;
   }
  /*自定义动画*/
  @keyframes slideLeft {
    from {
      transform: translate3d(100%, 0, 0);/*横坐标,纵坐标,z坐标*/
      visibility: visible;
    }
    to {
      transform: translate3d(0, 0, 0);
    }
  }
</style>
```

62.在vue-cli工程中如何实现无痕刷新？

无痕刷新:在不刷新浏览器的情况下,实现页面的刷新

一般常用的两种刷新方法：
`window.location.reload()`，原生 `js` 提供的方法；
`this.$router.go(0)`，`vue` 路由里面的一种方法；

这两种方法都可以达到页面刷新的目的，简单粗暴，但是用户体验不好，相当于按 `F5` 刷新页面，页面的重新载入，会有短暂的白屏。

vue开启无痕刷新

原理：先在全局组件注册一个方法，用该方法控制`router-view`的显示与否，然后在子组件调用全局方法。

```
//第一步：在app.vue里面设置
    <template>
        <div id="app">
            <!--通过切换isRouterAlive的值来控制页面的显示与否-->
            <router-view v-if="isRouterAlive"></router-view>
        </div>
    </template>
```

```
    <script>
    export  default {
        //给子组件暴露一个方法：这里将当前组件中的reload方法暴露给子组件
        provide(){
            return{
                reload:this.reload
            }
        },
        data(){
            return {
                isRouterAlive:true
            }
        },
        methods:{
            //reload方法中先把isRouterAlive该为false，让router-view不显示
            //然后在$nextTick方法里面重新把isRouterAlive该为true，让router-view重新显示
            //$nextTick表示下一次dom更新完毕之后，在更新dom的时候我们让router-view隐藏，更新dom完毕我们让router-view显示，此时就做到了无痕刷新
            reload(){
                this.isRouterAlive = false;
                this.$nextTick(function(){
                    this.isRouterAlive = true;
                })
            }
        }
    }
    </script>


//第二步：在.vue组件中使用全局方法(先用inject注册全局方法，然后即可通过this调用)
    <script>
    export  default{
        inject:['reload'],
        mounted(){
            this.reload();
        }
    }
    </script>
```

63.vue中中央事件总线？

中央事件总线 ：就是一个名字可以叫做**Bus的vue空实例**,里边没有任何内容。

它就像一个公交车一样，来回输送人，将组件A输送到组件B，再将组件B输送到组件A；

这里A，B组件可以是父、子组件，也可以是兄、弟组件，或者两个没有任何关系的组件；

我们可以使用中央事件总线这种技术来实现vue组件之间的数据通信。

```
//1.创建中央事件总线
var bus = new Vue();

//A组件发送
//2.使用Bus中央事件总线在A组件中发送信息
Bus.$emit('自定义事件名','$on发送过来的数据');


//B组件接收
//3.使用Bus中央事件总线在B组件中接收信息
Bus.$on('自定义事件名',function(){
    //然后执行什么你自己懂的。。。
});
```

64.vue开发命令 `npm run dev` 输入后的执行过程【拓展】

1. npm run dev 是执行配置在package.json中的脚本，比如：

```
"scripts": {
    "dev": "webpack-dev-server --inline --progress --config webpack.conf.js",
    "start": "npm run dev",
    "lint": "eslint --ext .js,.vue src",
    "build": "node build/build.js"
  },
```

npm run dev 执行的就是webpack-dev-server --inline ....命令，通过webpack-dev-server开启一个本地调试服务器。

2. 在webpack.conf.js文件中找到App的入口文件 ./src/main.js

```
entry: {
   app: './src/main.js'
},
```

3. main.js用到了页面元素#app、用到了路由和根组件App，并根据这些信息创建一个vue实例

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

4. webpack-dev-server会将main.js中的代码以及所有引用打包成一个bundle.js，然后配置到内存中
5. webpack.conf.js中配置的HtmlWebpackPlugin会将index.html文件配置到内存，并且将内存中的bundle.js注入到内存中的index.html中

```
new HtmlWebpackPlugin({
      filename: 'index.html',
      template: 'index.html',
      inject: true
}),
```

6. 根据webpack.config.js中所配置的devServer的信息，会决定是否自动打开浏览器呈现网页

```
devServer: {
    open: true, // 自动打开浏览器
    port: 3000, // 设置启动时候的运行端口
    contentBase: 'src', // 指定托管的根目录
    hot: true // 启用热更新 的 第1步
},
```

65.vue打包命令是什么？

vue-cli 生成 生产环境部署资源 的 npm命令：

```
npm run build
```

用于查看 vue-cli 生产环境部署资源文件大小的 npm命令：

```
npm run build --report
```

66.vue-cli 打包后会生成哪些文件？

```
dist
  --index.html          单页面文件
  --app.[hash].css      将组件中的css编译合并成一个app.[hash].css的文件
  --app.[hash].js       包含了所有components中的js代码
  --vendor.[hash].js    包含了生产环境所有引用的node_modules中的代码
  --mainfest.[hash].js  包含了webpack运行环境及模块化所需的js代码
  --0.[hash].js         是vue-router使用了按需加载生产的js文件
```

这样拆分的好处是：每块组件修改重新编译后不影响其他未修改的js文件的hash值，这样能够最大限度地使用缓存，减少HTTP的请求数。

67.如何配置 vue 打包生成文件的路径？

```
//vue.config.js

//部署应用包时的URL，如果是生产环境，部署到 ./cli-study/dist 路径；如果是开发环境，部署到根路径
publicPath: process.env.NODE_ENV === 'production'? './cli-study/dist': './',
outputDir: 'dist',
```

68.vue如何优化首屏加载速度？

> 问题描述：

> 在Vue项目中，引入到工程中的所有js、css文件，编译时都会被打包进vendor.js，浏览器在加载该文件之后才能开始显示首屏。若是引入的库众多，那么vendor.js文件体积将会相当的大，影响首屏的体验。

几种常用的优化方法：

1. 路由的按需加载
2. 将打包生成后 index.html页面 里面的JS文件引入方式放在 body 的最后
3. 用cdn缓存代替npm安装包，将引用的外部js、css文件剥离开来，不编译到vendor.js中
4. UI组件库的按需加载
5. 项目部署上线之后，开启服务器的Gzip压缩，使服务器尽可能返回更小的资源
6. 使用更高级的SSR服务端渲染框架，比如nuxt来做首屏加载优化

69.什么是mvvm

MVVM最早由微软提出来，它借鉴了桌面应用程序的MVC思想，在前端页面中，把Model用纯JavaScript对象表示，View负责显示，两者做到了最大限度的分离，把Model和View关联起来的就是ViewModel。

ViewModel负责把Model的数据同步到View显示出来，还负责把View的修改同步回Model

View 和 Model 之间的同步工作完全是自动的，无需人为干涉（由viewModel完成）

因此开发者只需关注业务逻辑，不需要手动操作DOM， 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理

70.MVVM模式的优点以及与MVC模式的区别

MVVM模式的优点：

1、低耦合：MVVM模式中，数据是独立于UI的，ViewModel只负责处理和提供数据，UI想怎么处理数据都由UI自己决定，ViewModel不涉及任何和UI相关的事，即使控件改变（input换成p），ViewModel几乎不需要更改任何代码，专注自己的数据处理就可以了

2.自动同步数据:ViewModel通过双向数据绑定把View层和Model层连接了起来，View和Model这两者可以自动同步。程序员不需要手动操作DOM， 不需要关注数据状态的同步问题，MVVM 统一管理了复杂的数据状态维护

3、可重用性：你可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑。

4、独立开发：开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。

5、可测试：ViewModel里面是数据和业务逻辑，View中关注的是UI，这样的做测试是很方便的，完全没有彼此的依赖，不管是UI的单元测试还是业务逻辑的单元测试，都是低耦合的

MVVM 和 MVC 的区别：

mvc 和 mvvm 其实区别并不大。都是一种设计思想，主要区别如下：
1.mvc 中 Controller演变成 mvvm 中的 viewModel
2.mvvm 通过数据来驱动视图层的显示而不是节点操作。
3.mvc中Mmodel和View是可以直接打交道的，造成Model层和View层之间的耦合度高。而mvvm中Model和View不直接交互，而是通过中间桥梁ViewModel来同步
4.mvvm主要解决了:mvc中大量的DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验。

71.常见的实现数据劫持的做法有哪些

实现数据劫持的做法有大致如下几种：

> 1.代理对象（proxy）2.Object.defineProperty()

1、代理对象

通过代理对象来访问目标对象，可以实现数据的劫持。

在代理的设计模式中，有目标对象、代理对象和事件处理程序，通过代理对象来访问目标对象，可以实现对目标对象的访问权限控制以及数据的劫持工作

```
//创建一个事件处理器
const handler = {
    get: function(obj, prop){
        console.log('A value has been accessed');
        return obj[prop];
    },

    set: function(obj, prop, value){
        obj[prop] = value;
        console.log(`${prop}is being set to${value}`);
    }
}

//目标对象
const initialObj = {
    id: 1,
    name: 'Foo Bar'
}

//代理对象
const proxiedObj = new Proxy(initialObj, handler);

//给代理对象赋值，会调用handler这个事件处理程序，然后调用事件处理程序中的set方法间接访问目标对象，给目标对象赋值
proxiedObj.age = 24
```

2、`Object.defineProperty()`：

---

vue.js 则是采用`Object.defineProperty()`来实现数据的劫持的，通过`Object.defineProperty()`来劫持各个属性的`setter`，`getter`，在数据变动时立马能侦听到从而调用setter和getter做对应的处理。

72.Object.defineProperty()方法的作用是什么？

`Object.defineProperty()` 方法会直接在一个对象上定义一个新属性，或者修改一个对象的现有属性， 并返回这个对象。

```
//语法：
Object.defineProperty(obj, prop, descriptor)

//参数说明：
obj：必需。目标对象
prop：必需。需定义或修改的属性的名字
descriptor：必需。目标属性所拥有的特性

//返回值：
传入函数的对象。即第一个参数obj
```

73.Vue项目中常用到的加载器：

```
- vue-loader -- 用于加载与编译 *.vue 文件，提取出其中的逻辑代码 script、样式代码 style、以及 HTML 模版 template，再分别把它们交给对应的 Loader 去处理。
- vue-style-loader -- 用于加载 *.vue 文件中的样式
- style-loader -- 用于将样式直接插入到页面的<style>内
- css-loader -- 用于加载 *.css 样式表文件；
- less-loader -- 用于编译与加载 *.less 文件（需要依赖less库）
- babel-loader -- 用于将 ES6 编译成为浏览器兼容的ES5
- file-loader -- 用于直接加载文件
- url-loader -- 用于加载 URL 指定的文件，多用于字体与图片的加载
- json-loader -- 用于加载 *.json 文件作为 JS 实例。
```

74.跨域问题的解决方案

在前端开发中，当我们使用ajax向服务器发送请求的时候，当协议、域名、和端口号有任何一个不一致的时候就会产生跨域问题。

跨域问题有很多中解决方案：我所用过的有：

1.jsonp 他的本质是使用script标签去发送请求，然后服务器返回一段js脚本以供客户端执行

2.cors 他需要在服务器端配置跨域访问的响应头

3.在前端的工程化项目(webpack)中,我们可以通过配置devserver的proxy来解决跨域访问的问题。他的原理是在本地开启一个服务器向数据服务器发送请求，因为服务器和服务器之间是没有跨域

4.但是因为webpack的devserver只在开发环境下有效，当项目发布上线之后仍然会有跨域问题，为了解决项目上线的跨域问题，我们配置服务器的反向代理（ngix）

5.除此之外，我还知道当项目打包成apk之后就不存在跨域问题了，所以如果项目要打包成apk，我们需要在项目中的所有请求中写全路径(此时我们可以配置axios.default.baseURL来解决)

75.Vue的双向数据绑定的原理

```
//1.由页面->数据的变化：通过给页面元素添加对应的事件监听来实现的
<input v-model="value" oninput="()=>this.handleInput($event)">
function handleInput(e){
    this.value = e.target.value
}

//2.由数据->页面的变化：通过数据劫持（Object.defineProperty） + 发布订阅模式来实现的
```

具体流程：

A.Compile解析器会将页面上的插值表达式/指定翻译成对应Watcher以添加到订阅器维护的列表中

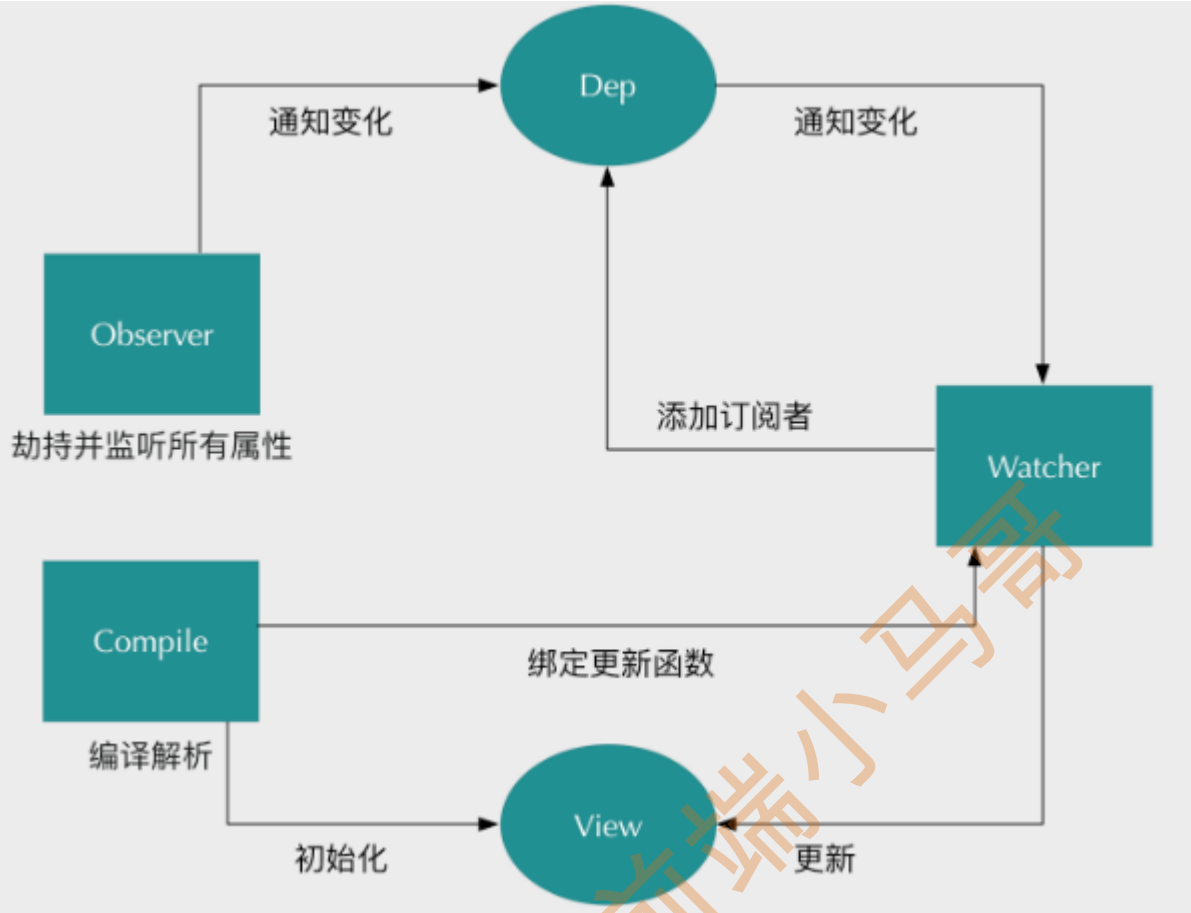
B.通过Object.defineProperty劫持数据的变化，一旦数据源发生变化会触发对应的set方法

C.在set方法中，通知订阅器(Dep)对象中维护的所有订阅者(Watcher)列表更新

D.每一个Watch会去更新对应的页面

```
//3.关于发布订阅模式
```

发布订阅模式又叫观察者模式，他定义了一种一对多的关系，让多个观察者对象同时监听某一个主体对象的变化，当这个主题对象的状态发生变化的时候就会通知所有的观察者对象，是的他们能够自动更新自己。



76.函数的节流阀和去抖

```
//1.函数去抖
```

答：函数调用n秒后才会执行，如果函数在n秒内被调用的话则函数不执行，重新计算执行时间。函数去抖主要避免快速多次执行函数（操作DOM，加载资源等等）给内存带来大量的消耗从而一定程度上降低性能问题。

函数去抖的应用场景：

1.监控键盘keypress事件，每当内容变化的时候就向服务器发送请求
2.在页面滚动的时候监控页面的滚动事件，会频繁执行scroll事件
3.监控页面的resize事件，拉动窗口改变大小的时候，resize事件被频繁的执行

上面三种场景中都会频繁触发指定事件，比如第一种情况，每当输入框内容变化之后就向服务器发送请求，可能会导致一秒钟向服务器请求很多次，这显然是不合理的，我们可以使用函数去抖来优化。

```
<input type="text" oninput="textInput()">

//模拟发送请求的方法
function ajax() {
    console.log("发送请求")
}
//文本框输入内容的时候频繁的处罚func方法
var func = debounce(ajax,1000)
function textInput() {
    func();
}

//函数去抖方法的封装
```

```
        //debounce: 当调用动作n毫秒后，才会执行该动作，若在这n毫秒内又调用此动作则将重新计算执行时间。
        function debounce(method, delay) {
            var timer = null;
            return function () {
                var context = this
                var args = arguments;
                clearTimeout(timer);
                timer = setTimeout(function () {
                    method.apply(context, args);
                }, delay);
            }
        }
```

```
//2.函数节流（throttle）：函数预先设定一个执行周期（或者节流阀），当调用动作的时刻大于等于执行周期则执行该动作，然后进入下一个新周期。

function throttle(method,duration){
        var  begin = new Date();
        return function(){
            var context = this
            var args=arguments
            var current=new Date();
            if(current-begin>=duration){
                 method.apply(context,args);
                 begin = current;
            }
        }
}
function resizehandler(){
    console.log(++n);
}
window.onresize=throttle(resizehandler,500);
```

函数节流的应用场景：
1.上拉下拉刷新，每拉动一次彻底完毕之后才可以下一次拉动
2.图片轮播动画，每一张图片动画完成之后才开始下一个图片的动画

77.vue中引入组件、注册组件、使用组件的步骤

1.引入组件
```
import App from '@/components/App.vue'
```

2.注册组件
注册全局组件 ： Vue.component("组价名字",{template:" 页面模板 "})
注册私有组件 ： 在当前组件/vue对象 使用components属性来声明私有组件

3.使用组件
当我们通过 `import App from '@/components/App.vue'` 这种方式引入组件之后，我们就可以在页面中使用组建了。
```
<App></App>
```
使用组件的时候，我们还可以通过属性给子组件传递数据,比如<App msg="123"></App>
在子组件中可以通过props来接收数据。
同样子组件也可以给父组件传递数据,....

78.单页和多页应用的优缺点

1.什么是单页面？

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.
优点：用户体验好
前后端分离
页面效果会比较炫酷（比如切换页面内容时的专场动画）
缺点：不利于seo
导航不可用，如果一定要导航需要自行实现前进，后退。
初次加载时耗时多
页面负责度提高很多

2.什么是多页面？

多页面(MPA),就是只一个应用中有多个页面，页面跳转时是整页刷新
优点：有利于seo
开发成本较低
缺点：网站的后期维护难度较大
页面之间的跳转用时较长，用户体验较差。
代码重复度大