笔记总结

原理

vue

1.vue的响应式数据的原理

原理：
对象内部通过 defineReactive 方法，使用Object.defineProperty 将属性进行劫持（只会劫持已经存在的的属性），数组是通过从写数组的七个方法（push、pop、unShift，shift、splice、sort，reverse）来实现， 对于多层的对象是通过 递归 来实现的；
vue2中的缺陷：
1. v2使用 Object.defineProperty 方法来进行数据劫持，需要对属性进行重写get、set方法，性能差
2. 当新增或删除属性时无法进行检测，需要使用 $set、$delete 实现；
3. 数组不使用 defineProperty 来进行劫持（数组元素多时劫持数组是非常消耗性能的），需要对数组进行单独的处理；
4. 无法劫持ES6中的Map、Set；
Vue3 采用了 Proxy 方法：
1. 它是惰性的，当有多层的数据结构时，如果不在 get 中获取，就不会进行递归操作，
2. 劫持的是对象本身，不是key， 可以劫持数组、Map、Set；

数组劫持:
1. 拷贝了数组的原型方法，定义需要监听的方法。
2. 数组数据的__proto__指向新拷贝的原型；
3. 通过forEach循环调用需要监听的方法，最后在调用数组原型的方法。

2.diff算法

diff算法概念： diff算法就是比较两个虚拟节点的差异，vue中的diff算法是平级比较。 内部采用深层递归+双指针的方式进行比较；
diff算的流程：
第一：比较新旧节点的 key 和 tag 是否相同；
第二：相同节点比较属性，并复用老的节点；
第三：比较儿子节点，
1. 老的没有儿子，新的有儿子，直接插入新的儿子
2. 老的有儿子，新的没有儿子，直接删除老的儿子
3. 老的儿子是文本，新的儿子也是文本，直接更新文本内容
4. 老的儿子是一个列表，新的儿子也是列表 ==> updateChildren
第四：比较新旧节点的儿子：头头、尾尾、头尾、尾头
第五：第四行不通，对比查找复用节点，（从新节点列表中的头部开始，依次去旧节点中查找）；

PS：vue3 中采用最长递增子序列来实现diff算法；

HTML

1. ☆ 重排(回流)、重绘 ☆

重排（回流）：当 DOM 的变化影响到元素的几何信息（DOM对象的位置和大小），浏览器需要重新计算元素的几何属性，将其摆放到合适的位置。

触发：

1. 添加和删除可见的DOM；
2. 修改元素的尺寸 -> 边距、填充、宽度、高度等；

优化：

1. 分离读写操作；
2. 样式集中修改；
3. 尽量只修改position：absolute或fixed元素，对其他元素影响不大；
4. 使用 transform: translate 属性；

重绘：当一个元素外观发生变化，但没有改变布局，重新将元素的外观绘制出来的过程。

触发：

改变元素的color、background、box-shadow等属性；

2. html5 新特性、语义化

语义化标签： header、nav、main、article、section、aside、footer

优点：

1. 代码结构清晰，易于阅读，利于开发。
2. 有利于 搜扫引擎（SEO），搜索引擎爬虫会根据不同的标签来赋予不同的权重
3. 方便其他设备解析（如屏幕阅读器）根据语义渲染网页

3. 网页生成过程

1. html 被HTML解析器解析成 DOM 树；
2. css 被CSS解析器解析成 CSSOM 树；
3. DOM 树和 CSSOM 树结合，形成 渲染树（Render Tree）；
4. 生成布局（flow）， 将渲染树上的所有节点进行平面合成；
5. 将布局绘制（paint）到屏幕上；

4. 标签上的title和alt的区别？

title：属性规定关于元素的额外信息，这些信息通常会在鼠标移到元素上时显示一段工具提示文本

alt：属性为图像提供了替代的文本，未显示的alt文字会代替图片

5. 前端页面由哪三层构成，分别是什么？作用是什么？

CSS

1. css基本语句构成

基本语句构成是选择器、属性、属性值

2. ☆ 盒子模型 ☆

盒子模型： （Box Model）就是在网页设计中经常使用到的css技术所使用的一种思维。

盒子模型：分为 标准盒子模型和怪异格子模型
标准盒模型下盒子的大小 = content + border + padding + margin；
怪异盒模型下盒子的大小 = width(content + border + padding) + margin

两种模式的转换：
标准：box-sizing:content-box;
怪异：box-sizing:border-box;

3. ☆ 什么是BFC？怎么形成 BFC？ 作用？ ☆

BFC(Block Formatting Content) 是块级格式化上下文， 是Web页面中盒模型布局的css渲染模式，是一个隔离的独立容器。

形成：
1. 浮动：float 除 none以外；
2. 绝对定位的元素；
3. 非块级元素具有display：inline-block、table-cell、flex；
4. 块级元素具有 overflow ，且值不是 visible；

作用：
1. 清除浮动；
2. 用于两栏自适应；
3. 防止 margin 合并；

4. display、opacity、visibility

1. 性能：
opacity、visibility： 不会触发回流；
display：会触发回流，所以性能最差；
2. 占位：

```
        opacity、visibility：视觉上消失，在文档流中占位；
        display：彻底消失，不在文档流中占位；
    3. 动画效果：
        display、visibility：对 transition 无效；
        opacity：对 transition 有效；
    4. 事件绑定：
        display、visibility：绑定事件无法触发；
        opacity：绑定事件可以触发；
    5. 子元素继承：
        opacity、visibility： 会被子元素继承 （子元素设置 visibility:visible; 会显示
）；
        display：不会被子元素继承；
```

5. CSS 优先级顺序？

```
    !important > 行内样式 > ID选择器 > 类选择器 > 标签 > 通配符 > 继承 > 浏览器默认属
性
```

6. 圣杯和双飞翼实现方式的区别?

7. 清除浮动的方法?

```
    1. 给父元素加 overflow: hidden;
    2. 在父元素最后添加空的div，style设置为 clear：both
    3. 给父元素直接添加高度；
    4. 使用伪元素 clearfix 来清理；
    5. 双伪元素方法
```

8. 元素居中的方式

9. 响应式

```
    就是一个网站兼容多个终端，而不是为每个终端做一个特定的版本。

    优点:
        1. 面对不同的分辨率的设备灵活性强；
```

```
        2. 能够快捷解决多设备显示适应问题；
    缺点：
        1. 兼容各种设备工作量大，效率低下；
        2. 代码累赘，会出现隐藏无用的元素，加载时间长；

    使用：
        @media screen and (min-width: 900px){} // 显示窗口 大于 900px
        @media screen and (min-width: 600px) and (max-width: 900px){}  // 显示窗口
大于 600px 小于 900px
        @media screen and (max-width: 600px){} // 显示窗口 显示窗口小于 600px
```

10. @import 和 link 标签的区别？

```
    1. 从属关系：
        @import是css提供的语法规则，只有导入样式表的作用
        link是html提供的标签，不仅可以加载css文件，还可以定义rss、rel连接属性等
    2. 加载顺序：
        加载页面时，link标签引入的css被同时加载；
        @import引入的css将在页面加载完毕后被加载
    3. 兼容性：
        @import是css2.1才有的语法，只可在ie5+才能识别；
        link标签作为html元素，不存在兼容性问题
    4. DOM可控性：
        可以通过js操作DOM，插入link标签来改变样式；
        无法使用@import的方式插入样式；
    5. 权重：link引入的样式权重大于@import引入的样式
```

11. div[foo]、div[foo='bar']、div[foo*='bar'] 分别代表什么？

```
    div[foo]：所有带 foo 属性的元素
    div[foo='bar']：所有带 foo 属性并且属性值为 bar 的元素；
    div[foo*='bar']：所有带 foo 属性并且属性值中带有 bar 的元素；
```

12. css 中伪类和伪元素有哪些？

```
    伪类：
        :active   ：将样式添加到被激活的元素
        :focus    ：将样式添加到被激活的元素
        :hover    ：当鼠标悬浮在元素上方是，向元素添加样式
        :link     ：当鼠标悬浮在元素上方是，向元素添加样式
        :visited      ：将特殊的样式添加到被访问的链接
        :first-child      ：将特殊的样式添加到元素的第一个子元素
        :lang     ：允许创作者来定义指定的元素中使用的语言
    伪元素:
        :first-letter  ：将特殊的样式添加到文本的首字母
```

```
:first-line ：将特殊的样式添加到文本的首行
:before ：在某元素之前插入某些内容
:after ：在某元素之后插入某些内容
```

Javascript

1. ☆ Js 的数据类型有哪些？☆

```
基本类型： String、number、Boolean、Null、Undefined、symbol(ES6)、BigInt(ES10)；
复制类型： Array、Function、Object(包含： Array、Function)；
```

2. ☆ Js 的数据类型检测有哪些方案？☆

```
1. typeof:
    优点：能够快速区分基本数据类型;
    缺点：不能将Object、Array和Null区分，都返回object;
    PS: JS中会将二进制前三位为'0'的判断为 object，而 Null 二进制表示全都是0。
    # typeof "test" //string
    # typeof [] //object

2. instanceof:
    优点：能够区分Array、Object和Function，适合用于判断自定义的类实例对象;
    缺点：Number，Boolean，String等基本数据类型不能判断;
    # [] instanceof Array  // true
    # "test" instanceof String // false

3. Object.prototype.toString.call():
    优点：精准判断数据类型;
    缺点：写法繁琐不容易记，推荐进行封装后使用;
    # Object.prototype.toString.call('test') === '[object String]' // true
```

3. ☆ 闭包 ☆

```
闭包： 有权访问另一个函数作用域中的变量的函数；
用途：
    1. 模拟块级作用域；
    2. 私有化变量；
    3. 创建模块；
    4. 能够访问函数中私有的变量；
优点： 延长变量的生命周期；
缺点： 会导致函数变量一直保存在内存中， 过多的闭包会导致内存的泄露；
常见的表现形式：
/*
    function fn(){
        let a = 1;
```

```
            return function(){
                a++;
                console.log(a);
            }
        }
        var f = fn();  // 闭包的产生，将fn返回的函数赋值到变量上，且不再调用fn函数；
        f();
    */
```

PS： 总结：局部变量无法共享和长期的保存，而全局变量可能造成变量污染，当我们希望有一种机制既可以长久保存又不会造成全局变量污染

3. ☆ 原型、原型链 ☆

原型：对象中固有的 __proto__（隐式原型） 属性，该属性指向构造函数的 prototype（显示原型） 原型属性。

原型链：当访问一个对象的属性时，会先在这个对象本身上找，如果没有会在这个对象的__proto__上找，对象的__proto__ 指向构造函数prototype，还是没有的话，会到在Object身上的__proto__上查找，直到最顶层为null。

PS:现在是 [[prototype]];

4. ☆ 继承 ☆

1. 原型链继承
   优点：父类的方法可以共享；
   缺点： 1. 子类实例无法向父类传参
          2. 父类上的引用类型（数组，对象）的数据会被子类修改
2. 借用构造函数继承
   优点：父类的引用类型的数据不会被子类修改
   缺点：子类不能访问父类原型上的属性、方法
3. 组合继承
   优点：父类可以复用，并且可以访问父类原型上的属性、方法
   缺点：会调用两次父类的构造函数，子类上的原型会有两份相同的属性和方法，影响性能
4. 寄生式组合继承
   优点：解决了上面三种方式的缺点
   缺点：实现较为复杂
7. ☆ ES6 extends ☆
   最优

5. ☆ this指向、new关键字 ☆

this指向：
1. 普通函数调用， this 指向 window；
2. 对象方法调用， this 指向 这个对象；
3. 构造函数调用， this 指向 用 new 创建的新对象；
4. 通过 call、apply、bind 调用， 指定调用函数的 this 指向；

new 关键字：
1. 首先创建了一个空对象；
2. 将对象的原型指向构造函数的 prototype；
3. 构造函数的 this 指向对象；
4. 返回新创建的对象

6. ☆ 防抖、节流 ☆

防抖：触发高频事件后在n秒内函数只会执行一次，如果在n秒内再次触发，则需要重新计算时间。
适用：
1. 搜索框搜索输入；
2. 手机号、邮箱验证输入检测 onchange oninput事件；
3. 窗口大小Resize；

节流：当持续触发事件时，保证隔间时间触发一次事件。
适用：
1. 懒加载、滚动加载、加载更多或监听滚动条位置；
2. 百度搜索框，搜索联想功能；
3. 防止高频点击提交，防止表单重复提交；

7. ☆ 浅拷贝、深拷贝 ☆

浅拷贝：只会复制某个对象的地址，不会复制对象的本身，新旧对象还是共享一块内存；
形式：
1. Object.assign()；
2. 扩展运算符；

深拷贝：会创建一个相同的对象，新旧对象不会共享一块内存，修改新对象的值不会改变就对象；
形式：
1. JSON.parse(JSON.stringify(obj)) // 函数、原型链上属性和方法无法被拷贝；
2. 递归函数

8. ☆ 作用域、作用域链 ☆

作用域：作用域就是变量的可用性的代码范围，就叫做这个变量的作用域。
作用域分为： 全局作用域、局部作用域、块级作用域；

作用域链：当你要访问一个变量时，首先会在当前作用域下查找，如果没有查找到，则回去上一级作用域进行查找，直到找到全局作用域，这个查找过程形成的链条叫做作用域链

9. ☆ EventLoop 事件循环 ☆

```
PS: 宏任务和微任务
微任务：Promise.then()，.then中的逻辑是微任务
宏任务：setTimeout、setInterval、xhr、callback
```

10. ☆ call、apply、bind ☆

```
call、apply、bind都是改变this指向的方法
区别：
    call： 参数1：this， 参数2：多个参数使用逗号隔开；
    apply：参数1：this， 参数2：数组的形式；
    bind：不会立即执行，而是返回一个函数，在传参；
```

11. ☆ 事件冒泡、事件委托（捕获） ☆

```
    事件冒泡：对象上绑定事件，如果该对象上有这个事件会触发，如果没有这个事件就会逐级上找；

    事件委托：把原本需要绑定在子元素的响应事件，委托给父元素，让父元素担当事件监听的职务；
            事件委托的原理是DOM元素的事件冒泡。

PS： 阻止事件冒泡和阻止默认事件
    阻止事件冒泡： e.stopPropagation() || e.cancelBubble = true;
    阻止默认事件： e.preventDefault()
    阻止事件冒泡并阻止默认事件： return false;
```

12. 普通函数 和 箭头函数的区别？

```
箭头函数：
    1. 不能做构造函数，没有prototype原型对象，不能做 new 操作；
    2. 没有this， 在声明时可以捕获其所在上下文的this供自己使用；
    3. 不能通过 apply，call，bind 改变 this 指向；
    4. 没有 arguments 对象，通过使用 ...形参 的方式代替；
    5. 一个参数可省略(), 一行逻辑可省略{} 和 return；
```

13. 什么是跨域问题，如何解决跨域问题?

跨域：是指浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的。
同源策略：是浏览器对 JS 实施的安全限制，只要 协议、域名、端口 有任何一个不同，都被当作是不同的域。
跨域原理：即是通过各种方式，避开浏览器的安全限制。
方式：
1. JSONP 的缺点: 使用get请求不安全，携带数据较小；
2. CORS : 跨域资源共享 服务器设置对CORS的支持原理；
3. proxy代理: 在开发中使用proxy，在服务器上使用nginx代理；
4. window.postMessage() 利用h5新特性window.postMessage()

14. 双等号的规则

#1. '对象 == 字符': 会把对象隐式转换为字符串。
2. 'null == Undefined': 为 true ，除了这种情况null/undefined和其他任何值都不会相等；
3. 'NaN == NaN': 为 false ;
4. 'Symbol() == Symbol()': 为 false;

除了上面的情况，不同类型间对比的时候都要先转换成数字，再进行比较。

PS: 转换都是通过valueOf,(valueOf()方法通常由 Javascript 在后台自动调用)；

15. Object.is()、 == 和 === 的区别？

Object.is(): 用来判断两个值是否严格相等，处理一些特殊的情况；
// 1. +0 和 -0 不再相等;
// 2. 两个 NaN 相等；

'==': 只比较值，不比较值的类型； 不同的类型会进行隐式的转换；

'===': 会比较值和类型；如果类型不同直接返回 false；

16. ☆ localStorage、sessionStorage、cookies 有什么区别? ☆

相同点：
将数据存储在客户端
不同点：
1. 存储大小：
cookie为4K；localStorage、sessionStorage为5M或更大；
2. 有效时间：
localStorage： 需要用户主动删除
sessionStorage： 关闭浏览器窗口就会删除
cookie：可以设置时间， 没有时间关闭浏览器窗口就会删除
3. 交互方式
cookie：cookie数据可以自动传递到服务端， 服务端也可以写cookie到客户端

```
            localStorage、sessionStorage： 不会传递到服务端，仅在客户端保存
        4. 操作方法：
            cookie：需要自己封装方法
            localStorage、sessionStorage： 提供了API， setItem、getItem、
removeItem...
```

17. for...in 和 for...of的区别?

```
    1. for...in 是ES5， for...of 是 ES6；
    2. for...in 循环出的是 key， for...of 循环出的是 value；
    3. for...in 会遍历所有的可枚举属性，包括原型；
    4. for...of 遍历对象，需要配合 Object.keys() 使用；

    PS: for...in 更适合遍历对象， for...of 更适合遍历数组；
```

18. ☆ undefined 和 null 的区别？☆

```
    undefined 未定义； null： 为空；
    undefined: 代表 '无' 的原始值(未初始化的变量)，转化数值时为 NaN；
    null: 代表 '无' 的对象(空对象的指针)， 转化数值时为 0 ；

    出现场景：
    undefined
        1. 变量命名但未赋值的时候 为 undefined；
        2. 函数没有返回值的时候 为 undefined；
        3. 对象中属性没有没有赋值的时侯 为 undefined；
        4. 调用函数时，应该提供参数但未提供时 为 undefined；
    null：
        1. 作为对象原型链的终点；
        2. 作为函数的参数，表示参数不是对象（函数不需要参数是可以传入null）
        3. 如果定义的变量准备将来用于保存对象，可以将变量的初始值设置为 null
        4. 内存回收-对象、数组销毁
        5. 删除时间监听
```

19. break 和 continue 的区别？

```
    break：结束整个循环，不在判断执行循环条件是否成立；
    continue： 只结束本次循环，而不是整个循环，不能再switch中使用；
```

20. 哪些情况会导致内存泄漏

```
    1. 意外的全局变量：由于使用为声明变量，而意外的创建了一个全局变量而这个变量一直保存在
内存中无法回收。
```

2. 被遗忘的计时器或回调函数：设置了 setInterval 定时器，而忘记取消它，循环函数中引用外部变量，这个变量将保存在内存中无法回收。
3. 脱离 DOM 的引用：获取一个DOM元素，然后将其删除，由于一直保留了对DOM的引用，所以他也无法被回收。
4. 闭包：不合理的使用闭包，从而导致某些变量一直保存在内存中。

21. NaN

NaN（not a number）：不是一个数字，表示是否属于 number 类型的一种状态；是或否，不是确切的值；
PS: JS中number数据类型除了浮点型和整数型，还有一个特殊值 NaN；
console.log(typeof NaN); // number；
console.log(NaN == NaN); // false, 不是确切的值；
console.log(Number("abc")) // NaN, 表达式中存在不可转化的变量，返回了无效的结果，不是返回确切的值；

22. 工厂函数和构造函数的区别？

工厂函数：就是一个函数，用来创造对象的，对象拥有相同的属性和方法；
构造函数：当一个普通函数作为创造一类对象时，称为构造函数；需要通过 new 这个函数创建。
区别：
1. 工厂函数创造的对象是Object实例， 构造函数创造的对象是构造函数的实例。
2. 工厂函数创造需要在内部将对象返回， 构造函数创造对象是需要通过new关键字。（不使用new，构造函数内部的属性和方法将直接绑定到window上）
3. 工厂函数创造的对象中的方法属性都是独立的， 构造函数可以将方法放在原型上减少对内存的消耗。

23. 如何判断一个空对象

```
let obj = {};
1. 使用JSON.stringify()
    JSON.stringify(obj) == '{}';
2. 使用 ES6 的Object.keys();
    Object.keys(obj).length !== 0
```

24. document.load 和 $(document).ready 的区别

document.load：load是当页面所有资源全部加载完成后（包括DOM文档树，css文件，js文件，图片资源等），执行一个函数;
$(document).ready： 是当DOM文档树加载完成后执行一个函数 （不包含图片，css等）所以会比load较快执行；

25. ☆JS垃圾回收机构☆

JS垃圾回收机构： 是为了防止内存泄露，垃圾回收机构会不断寻找那些不在使用的的变量，并且释放掉它所指向的内存。
变量的生命周期：
全局变量：会一直存在在内存中，直到关闭浏览器为止
局部变量：是在当前的函数中产生作用，当该函数结束之后，该变量内存会被释放。
JS垃圾回收的方式：
标记清除：大部分浏览使用标记清除，当变量生命时进行标识，当变量离开变量时在进行标记，随之进行清除
优点：能够解决循环引用的问题；
缺点：内存碎片化；
引用计数：如果变量指向了一块内存空间，计数器+1，否则-1， 当计数器为0时，就会被清除。
优点：
1. 时刻监控内存的变化； 2. 能够降低程序卡顿的情况；
缺点：
1. 造成比较大的性能开销； 3. 造成循环引用的问题

26. 逐进增强和优雅降级

逐进增强：针对低版本的浏览器进行构建页面，保证最基本的功能，然后在针对高版本的浏览器进行效果、交互等的改进和追加功能达到更好的用户体验。
优雅降级：一开始就构建完整的页面，然后在针对低版本的浏览器进行兼容。

ES6、ES7、ES8、ES9、ES10

ES6

1. ES6 新增

1. 新增symbol类型 表示独一无二的值，用来定义独一无二的对象属性名;
2. const/let 都是用来声明变量,不可重复声明，具有块级作用域。存在暂时性死区，也就是不存在变量提升。(const一般用于声明常量);
3. 变量的解构赋值(包含数组、对象、字符串、数字及布尔值,函数参数),剩余运算符(...rest);
4. 模板字符串(${data});
5. 扩展运算符(数组、对象);;
6. 箭头函数;
7. Set和Map数据结构;
8. Proxy/Reflect;
9. Promise;
10. async函数;
11. Class;
12. Module语法(import/export);
13. Object.is();

2. ☆ Promise ☆

```
    Promise 是异步编程的一种解决方式，主要解决的是回调地狱（回调金字塔）；

    Promise 的三种状态：
        1. pending： 等待中
        2. fulfilled：已成功
        3. rejected： 已失败
        PS: 一旦状态发生改变，就不会在变。 pending -> fulfilled、 pending ->
rejected，只要处于 fulfilled、rejected 状态就不再变

    Promise 对象的方法：
        1. Promise.then：
            参数1： 成功的回调函数
            参数2： 失败的回调函数
        2. Promise.catch： 用来捕获异常的；
        3. Promise.finally：无论返回的结果是成功还是失败，都会执行finally方法；
        3. Promise.all：作用是接收一组异步任务，然后并行执行异步任务，并且在所有异步操作
执行完后才执行回调。
            使用：Promise.all([p,p1,p2,p3...]).then();
        4. Promise.race：作用是接收一组异步任务，然后并行执行异步任务，只保留取第一个执
行完成的异步操作的结果，其他的方法仍在执行，不过执行结果会被抛弃。
            使用：Promise.race([p,p1,p2,p3...]).then();
```

3. ☆ const、let、var ☆

```
    1. var 存在变量提升， const、let 没有变量提升；
    2. var 可以重复声明， const、let 不可以重复声明；
    3. var 不会构成块级作用域， const、let会构成块级作用域；
    4. const 声明后必须赋值且不能被修改， let 声明后可以不赋值且可以被修改；
```

4. ☆ forEach、map、filter、some、every、reduce、find、findIndex☆

```
    相同点：
        都不会主动改变原数组中的基本类型的值；可以通过回调函数中原数组进行修改；
    不同点：
        1. forEach 和 其他相比： forEach 没有返回值， 其他方法是有返回值的 需要
return；
        2. filter 是过滤数组中的数据；
        3. some 是数组中有一个符合条件就返回 true；
        4. every 是数组中数据都符合条件就返回 true；
        5. reduce 是 参数1：回调函数：
                                参数1：初始值，参数2：值，参数3：下标，参数4：原数组，
                        参数2： 初始值；
        6. find 是判断数组中的元素是否有符合的值；(返回符合条件的第一个元素)；
                如果有，则返回该元素; 如果没有，则返回 undefined;
        7. findIndex 是判断数组中的元素是否有符合的值;(返回符合条件的第一个元素)；
```

```
如果有，则返回该元素的下标; 如果没有，则返回 -1;
```

4. ☆ Set 和 Map 有什么区别？ ☆

```
相同：
    1. 都能通过迭代器进行for...of 遍历；
    2. 键和值可以是任何得值；

不同：
Set：是值集合，Set的值是唯一的，可以做数组去重，
    1. 使用add(值)添加数据；
    2. 初始化 new Set([1,2,3]);

Map：是键值对，而Map由于没有格式限制，可以做数据存储
    1. 使用set(键，值)存储数据，使用get(键)获取数据；
    2. 初始化 new Map([[1, 999],[true, 888],['aa', 777]]);
```

5. Object.is();

```
Object.is(): // 用来判断两个值是否严格相等，处理一些特殊的情况；
//如: +0 和 -0 不在相等， 两个 NaN 相等
console.log(Object.is(+0, -0)); // false
console.log(Object.is(NaN, NaN)); // true
```

Vue、vuex、axios、vue-router

Vue

1. vue 渐进式框架

Vue是一套构建用户界面的 渐进式框架，Vue被设计为 自底向上逐级应用，Vue的核心库只关注视图层，不仅容易上手，还可以与第三方的库或已有的项目进行整合。

2. 什么是MVVM？

MVVM 是 Model-View-ViewModel 的缩写， M：代表数据模型，定义操作数据的业务逻辑，V：代表视图层，负责渲染数据到页面上， VM：通过双向绑定View和Model进行同步交互，不需要手动操作 DOM 的一种设计思想。

3. Vue 的两个核心？

1. 数据驱动： 数据双向绑定，用于保证数据和视图的一致性。
2. 组件系统： 把页面抽象成对个相对独立的组件，实现代码的重用，提高开发效率和代码的质量，便于代码维护。

4. v-if 和 v-show 的区别？

1. 相同性：
   都可以控制 DOM 的 显示和隐藏
2. 不同性：
   1. 方式： v-if：对 DOM 是删除和创建的过程。 v-show： 只是通过控制css中的display，对 DOM 显示和隐藏。
   2. 编译： v-if：值为真的时候才会进行渲染和编译。 v-show：不管值为真假都会进行渲染和编译。
   3. 消耗： v-if： 切换时消耗大， v-show： 初始渲染的时候消耗大。
   4. 适用： v-if： 适用于不频繁切换的业务。 v-show： 适用于频繁切换的业务

5. v-if 和 v-for 优先级（vue2的问题， vue3中已经修复）

v-for 的优先级高于 v-if， 这就意味着， 不管 v-if 值为真假都会先循环，后判断。
PS： vue3 已经 将 v-if 的优先级高于 v-for

6. ☆ vue中 key 值的作用 ☆

key的概念：
1. key 主要用在 Vue 的虚拟DOM算法中，在新旧nodes对比时识别VNodes。 如果不使用key，就会vue没有办法精确的更新元素，而是采用就地复用（旧的标签中插入新的内容）在元素尾部添加新旧内容。
2. 当 vue正在使用v-for是，没有key，默认使用就地复用（（旧的标签中插入新的内容））， 如果数据项顺序发生变化，vue不会移动DOM元素来匹配数据项的顺序，而是就地更新每一个元素。

key的作用
1. Vue 在 patch 过程中通过key可以判断两个虚拟节点是否相同。（可以复用旧节点）
2. 无key会导致更新的时候出现问题。（标签相同就会复用旧的节点）
3.尽量不要采用索引作为key

7. 什么是虚拟DOM？

虚拟DOM就是普通的js对象。是一个用来描述真实dom结构的js对象，因为它不是真实的dom，所以才叫做虚拟dom。

8. vue.directive 自定义指令

指令本质上是装饰符，是 vue 对HTML 元素的的扩展，给HTML元素增加自定义功能。
当 vue 编译DOM时，会找到指令对象，执行指令的相关方法。
#vue2：
自定义指令有五个生命周期：
1. bind：只调用一次，指令第一次绑定元素时调用。
2. inserted：被绑定元素插入父节点时调用；
3. update：被绑定元素所在的模板更新时调用，无论绑定值是否变化。
   私有：oldVnode：上一个虚拟节点
4. componentUpdate：被绑定元素所在模板完成一次更新周期是调用；
   私有：oldVnode：上一个虚拟节点
5. unbind：只调用一次，指令与元素解绑时调用；

#vue3
自定义指令有七个生命周期：
created：
beforeMount：在元素被插入到 DOM 前调用
mounted：在绑定元素的父组件，及他自己的所有子节点都挂载完成后调用
beforeUpdate：绑定元素的父组件更新前调用
私有：之前的渲染中代表指令所绑定元素的 VNode
updated：在绑定元素的父组件，及他自己的所有子节点都更新后调用
私有：之前的渲染中代表指令所绑定元素的 VNode
beforeUnmount：绑定元素的父组件卸载前调用
unmounted：绑定元素的父组件卸载后调用

生命周期共有参数：
el：指令所绑定的元素，可以用来直接操作 DOM；
binding：一个对象，传递过来的数据；
vnode：Vue 编译生成的虚拟节点；

9. Vue 为什么时候异步渲染？

当数据发生变化时就要重新渲染数据，这样性能不高， 为了防止数据一更新就更新组件，所以做了异步渲染。

10. ☆ $nextTick? ☆

vue中的nextTick主要用于处理数据动态变化后，DOM还未及时更新的问题，用nextTick就可以获取数据更新后最新DOM的变化
应该场景：
1. 获取数据更新后的DOM
2. 使用第三方插件 如：better-scroll
3. 数组更新后获取焦点
源码|原理：
Promise ==> mutationObserver ==> setImmediate ==> setTImeout

11. 异步渲染的原理？

只要侦听到数据变化，Vue将开启1个队列，并缓存在同一事件循环中发生的所有数据变更。如果同一个 watcher 被多次触发，只会被推入到队列中-次。这种在缓冲时去除重复数据对于避免不必要的计算和DOM操作是非常重要的。nextTick方法会在队列中加入一个回调函数，确保该函数在前面的dom操作完成后才调用；

12. ☆ Vue 中的 keep-alive ☆

作用：实现组件缓存;

生命钩子函数：
activated: 组件渲染后调用
deactivated: 组件销毁后调用

配置属性:
1. include: 只有名称匹配的组件会被缓存
2. exclude: 任何名称匹配的组件都不会被缓存
3. max: 最多可以缓存多少组件实例

13. 什么是Vue的生命周期？

每个Vue实例在创建是都会经过一系列的初始化过程。
Vue 的生命周期钩子，就是说在某一个阶段或条件去触发的函数，目的就是为了完成一些动作过着事件。

14. ☆ Vue的生命钩子有哪些？（8 + 2）☆

beforeCreate: 创建前：data和methods中的数据没有初始化
created： 创建后：此时 data 和 method 已经初始化完成， DOM 未挂载

beforeMount： 数据挂载前：模板已获取到，但是数据未挂载到模板上
mounted： 数据挂载后： 数据已挂载到模板中

beforeUpdate： 模板更新前：data 改变后，更新数据模板前调用
updated： 模板更新后：将 data 渲染到数据模板中

beforeDestroy： 实例被销毁前：此时可以手动销毁一些方法
destroyed： 销毁后

keep-alive:
activated: 组件渲染后调用

```
        deactivated: 组件销毁后调用

    补充：
    1. created 和 mounted 的区别？
        1. 阶段： 有没有 $data、 $el
            created：有 $data ，没有 $el； PS： 在 created 中获取DOM只能在异步方法中，或者 Vue 自带的 nextTick 中；
            mounted：有 $data, 有$el； PS：可以同步获取DOM；
        2. 执行顺序：
            created：先执行，如有有子组件：也先执行
            mounted：后执行，如有有子组件：在子组件执行完后，在执行
    2. 生命周期的使用场景？
        created：单组件请求；
        mounted: 1. 同步获取DOM， 2. 子组件优先展示内容；
        activated: 判断 ID 是否相同，如果不同则发起请求；
        deactivated: 关闭页面时记录信息，如：页面滚动位置、视频播放时长等等；
    3. 生命周期的执行顺序？
        父：beforeCreate、created、beforeMount；
        子：beforeCreate、created、beforeMount、mounted；
            ...
        父：mounted；
```

15. 什么是scoped？ 其原理是什么？

```
    scoped 是将本组件中的 style 样式，只作于本组件。
    原理：给 HTML 中的 DOM 添加一个不重复 data 属性（形如，data-v-2311c06a）， 来表示他的唯一性。
```

16. Vue组件中 data 为什么是函数？

```
    为了组件独立性和复用性，data需要是函数。
    如果data是对象的话，每个组件实例的data 都是同一个引用数据，在使用组件的时候，一个组件改变了，其他组件也跟着改变。
    data要使用函数的形式，每次使用组件的时候创造一个新的地址，这样就不会相互影响了

PS: 数据定义在data中return内和return外的区别？
    return外：单纯的修改这个数据是没有办法修改的，因为没被数据劫持（get/set）；
    return内：是可以被修改的；
```

17. ☆ Vue的响应式原理？☆

```
    当一 Vue 在实例创建是， Vue 都会遍历 data 选项的属性， 用 Object.defineProperty 对这些属性进行监听，当访问和修改属性通知变化。
    每个组件实例都有相应的 watcher 程序实例，它在组件渲染的过程中会把属性记录作为依赖，当依赖变化时，会通知 watcher重新计算，从而致使它关联的组件得以更新
```

18. Vue 常用的修饰符

```
1. v-model 修饰符
    1. .lazy: 输入框改变，光标离开 input 框才会更新数据；
    2. .trim: 输入框过滤首尾空格；
    3. .number：自动将用户的输入值转化为数值类型；

2. 事件修饰符
    1. .once: 绑定的事件只触发一次；
    2. .self: 触发当前元素自身的函数；
    3. .stop: 阻止事件冒泡， 同理： e.stopPropagation()
    4. .prevent: 阻止默认事件， 同理： e.preventDefault();
    5. .native： 触发子组件的的事件

3. 键盘事件修饰符  @keyup 事件
    1. .enter 回车
    2. .delete 捕获“删除”和“退格”键
    ...
```

19. ☆ computed 为什么有缓存？实现原理？☆

```
缓存：
原理：
```

20. ☆ watch ☆

```
deep 的实现原理:
```

21. ☆ computed 和 watch ☆

```
computed:
    1. 使用和 data 中的数据一样，
    2. 在调用时不需要加 ( )
    3. 必须使用 return 返回
    4. 依赖项的属性没有变化时，从缓存中读取

watch:
    1. 函数名称必须和data中的数据名一致
    2. watch 中的函数有两个参数，前者是newVal，后者是oldVal
    3. watch 中的函数是不需要调用的
    4. 只会监听数据的值是否发生改变，而不会去监听数据的地址是否发生改变，要深度监听需
要配合deep：true属性使用
    5. immediate:true 页面首次加载的时候做一次监听
```

```
    区别：
        1. 功能：  computed是计算属性；
                   watch是监听一个值的变化执行对应的回调；
        2. 是否调用缓存：  computed函数所依赖的属性不变的时候会调用缓存；
                          watch每次监听的值发生变化时候都会调用回调；
        3. 是否调用return：  computed必须有；
                            watch可以没有；
        4. 使用场景：  computed当一个属性受多个属性影响的时候；例如购物车商品结算；
                      watch当一条数据影响多条数据的时候，例如搜索框；
        5. 是否支持异步：  computed函数不能有异步；
                          watch可以；
```

21. ☆ v-model语法糖是怎么实现的? ☆

```
    v-model：  本质上不过是语法糖，它负责监听用户的输入事件以及更新数据，并对一些极端场景进行一些特殊的处理；
    通过给元素绑定input或change事件，监听用户输入的内容；  然后将内容通过  v-bind  渲染到元素上；

    PS:
        v-model：双向绑定
        v-bind：  单向绑定
```

22. ☆ 说一下SPA单页面有什么优缺点？ ☆

```
    优点：
        1. 体验好，不刷新，减少请求，数据通过ajax获取；
        2. 前后端分离，开发分工明确；
        3. 减少服务端压力；
    缺点：
        1. 首屏加载慢；（解决 vue的项目优化）；
        2. SEO 不利于搜索引擎抓取；（解决 vue 单页面应用怎么做的SEO）
```

23. ☆ 组件之间的传值有几种方式？ ☆

传值详情

```
    1. 父传子：  props
    2. 子传父：  $emit
    3. 兄弟组件：  EventBus
    4. 全局组件：  Vuex、cookie、localStorage、sessionStorage
    5. 不常使用：  provide、inject，  $parent、$children，$attrs、$listeners，$ref、$refs;
```

24. v-html 存在的问题？

1. 易导致XSS攻击；
2. v-html 会覆盖当前标签内的子元素；
3. scoped 的样式不会应用在 v-html 内部；
4. v-html 的内容不会经过vue的模板编译器处理；

25. vue 怎么做数组的响应式?

vue2中： vue 重新写了数组的七个方法
1. push：在数组尾部添加一个元素；
2. pop：删除数组尾部的一个元素；
3. unshift：在数组头部添加一个元素；
4. shift：删除数组头部的一个元素；
5. splice：指定索引 删除、插入、替换 元素；
6. sort：对数组进行排序；
7. reverse：对数组内的元素进行反转；

vue3中： 使用 Proxy 是可以监听数组中的元素的；

26. 说一下vue开发环境和线上环境如何切换

27. ☆ Vue 项目的优化 ☆

1. 循环的时候使用正确的key 。
2. 封装高度复用的模块。
3. 路由懒加载，首屏加载快。
4. keep-alive 缓存不活跃的组件。
5. 尽量减少data中的数据，data中的数据都会增加getter和setter，会收集对应的watcher。
6. 图片放到cdn上 ，icon字体文件。
7. 组件按需导入 ，需要什么导入什么。
8. 防抖、节流
9. 第三方模块按需导入
10. 在更多的情况下，使用v-if替代v-show

28. vue.$set()、vue.$delete()

问题： 属性必须在 data 对象上存在，才是响应式的数据。如果在给对象添加新的属性，新加的属性不是响应式的， 需要通过 Vue.$set();
使用：
Array： this.$set(Array, index, newValue);
Object： this.$set(Object, key, newValue);

PS: 除了 $set 外 this.$forceUpdate() 也可以更新视图； #不建议使用
使用：this.$forceUpdate();
使用场景：
vue多层循环，动态改变数据后渲染的很慢或者不渲染。
比如v-for里面数据层次太多， 修改过数据变了，页面没有重新渲染，需手动强制刷新。

29. vue 单页面应用怎么做的SEO？

1. 使用 Nuxt.js
    1. Nuxt 服务端渲染应用部署;
    2. Nuxt 静态应用部署;
2. 使用 插件
    1. 'vue-meta-info';
    2. 'prerender-spa-plugin';

30. vue2 页面 data 中的数据放在 return 和 不放return 的区别？

放在return：数据是响应式的，数据改别视图也跟着改别； PS：大量的数据放到return中会导致页面的性能差。

不放return：数据不是响应式的，但数据是可以改别的，
使用场景： 一个变量没有在html显示，而在内部用到，最好定义在return外面
更新视图：
1. this.$forceUpdate(); 强制更新视图；
2. 使用 DOM 的方法；

32. 如何找到根组件和父组件

```
this.$root.$el // 根组件
this.$parent.$el // 找到当前组件的父组件， 如果找不到返回自身
```

34. ☆封装组件☆

35. Vue组件中name的作用？

1. 当项目使用 keep-alive 时， 可搭配组件的 name 进行缓存过滤。
2. 组件进行递归时，需要调用自身的 name。
3. vue 调试工具中显示的 name， 是由 vue 组件中的 name 决定的。

37. vue mixins？

定义：将组件的公共逻辑或者配置提取出来，哪个组件需要用到时，直接将提取的这部分混入到组件内部即可。这样既可以减少代码冗余度，也可以让后期维护起来更加容易。

使用：

局部使用：mixins：[mixins]

全局使用：Vue.mixin(mixins);

抽离代码和组件代码冲突：

（1）生命周期函数：先执行mixin中生命周期函数中的代码，然后在执行组件内部的代码

（2）data数据冲突：组件中的data数据会覆盖mixin中数据；

（3）方法冲突：组件方法调用；

优缺点：

优点：1.提高代码复用性 2.无需传递状态 3.维护方便，只需要修改一个地方即可

缺点：1.命名冲突 2.滥用的话后期很难维护 3.不好追溯源，排查问题稍显麻烦 4.不能轻易的重复代码

PS：Mixin和Vuex的区别？

Vuex公共状态管理，主要是以数据状态为主，在一个组件中修改数据，其他组件中也会跟着变化。

mixins中的数据和方法都是独立的，组件之间使用后是互相不影响的。

38. options API 和 composition API 有什么区别？

Composition API是一组API，包括：Reactivity APl、生命周期钩子、依赖注入，使用户可以通过导入函数方式编写vue组件，使得组件的可复用性、可维护性和可读性更强；而Options API则通过声明组件选项的对象形式编写组件；

Composition API最主要作用是能够简洁、高效的复用逻辑；解决了过去options API中mixins的各种缺点（命名冲突、来源不明）；另外Composition API具有更加敏捷的代码组织能力，很多用户喜欢options API，认为所有东西都有固定位置的选项放置代码，但是单个组件增长过大之后这反而成为限制，一个逻辑关注点分散在组件各处，形成代码碎片，维护时需要反复横跳，而Composition API则可以将它们有效组织在一起；最后composition API拥有更好的类型推断，对ts支持更友好，options API在设计之初并未考虑类型推断因素，虽然官方为此做了很多复杂的类型体操，确保用户可以在使用options API时获得类型推断，然而还是没办法用在mixins和provide/inject上；

vue3首推Composition API，但是这会让我们在代码组织上多花点心思，因此在选择上，如果我们项目中只有一些低复杂度的场景， options API仍是一个好选择；而对于那些大型，高扩展，强维护的项目上，composstion API 会获得更大收益；

39. assets 和 static 的区别

相同点：assets 和 static 两个都是存放静态资源文件。
不同点：
assets：assets 中存放的静态资源文件在项目打包过程中，会被压缩体积、代码格式化。打包完成后会被存放到static中。
static：static 中放置的静态资源文件就不会要走打包压缩格式化等流程，从而体积会比 assets 要大；
PS： 建议：项目中自己编写的css、js、image 存放到 assets 中。
像第三方的文件如 iconfoont.css。存放到 static， 避免再次压缩时出错。

40. delete 和 Vue.delete 删除数组的区别

delete 只是被删除的元素变成了 empty/undefined 其他的元素的键值还是不变。（数组的长度没有变化）
Vue.delete 直接删除了数组 改变了数组的键值。（数组的长度发生变化）

Vue2和Vue3的区别？

1. 双向数据绑定：
vue2：使用 ES5 的Object.defineProperty();
vue3: 使用 ES6 的 proxy; （懒惰的递归、用户不调用就不递归）

2. 生命周期：
vue2:
beforeCreate、created、beforeMount、mounted、beforeUpdate、updated、beforeDestroy、destroyed、activated、deactivated
vue3:
setup、onBeforeMount、onMounted、onBeforeUpdate、onUpdate、onUnmount、onUnmounted、onActivated、onDeactivated

3. vue2（Options API）和 vue3（Composition API）；
vue2 使用选项型API（Options API）
vue3 使用合成型API（Composition API）

4. 是否支持多个根节点
vue2 只支持一个根节点
vue3 支持多个根节点

5. 自定义指令生命周期不同：
vue2：（五个） bind、inserted、update、componentUpdate、unbind；
vue3：（七个）created、beforeMount、mounted、beforeUpdate、updated、beforeUnmount、mounted

Vuex

1. 什么是Vuex？

Vuex 是 vue.js 应用程序开发的 状态管理模式 。它采用集中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

2. Vuex 解决了什么问题？

中大型单页面应用程序，项目开发中频繁的使用组件传参的方式来同步 data 中的值，管理和维护这些值是非常难的， vuex 是 vue 为了这些被多个组件频繁使用的值提供的一个统一管理的工具。

3. Vuex 有哪几种属性？

state: 存储所有共享的状态信息；
getters: 计算 state 中的值，在数据展示前进行一些变化处理，具有缓存功能，能够提高运行效率，不可以直接修改数据；
mutations: 更改状态的逻辑，同步操作；理论上修改 state 的唯一路径；
actions: 提交mutation，可以包含异步操作；
modules: 将vuex进行分模块；

4. mapState, mapGetters, mapActions, mapMutations

当一个组件需要获取多个状态时候，将这些状态都声明为计算属性会有些重复和冗余。
mapState：
mapGetters：
mapActions：
mapMutations：
PS: 使用辅助函数不可以直接修改数据；

5. store 需要注意的问题？

1. store 存储的状态是响应式的，当组件读取状态时，如果 store 中的状态发生改变，那么相应的组件也会发生改变。
2. 不能直接改变 store 中的状态。改变 store 的唯一途径是 （commit）mutations，这样使得我们可以方便地跟踪每一个状态的变化。

6. mutation 和 action 有什么区别？

1. 流程顺序：视图触发 -> action 触发 -> mutations 触发 -> state；

2. 定位：基于流程顺序，二者扮演不同的角色。
    1. mutation：专注于修改State，理论上是修改State的唯一途径。

```
        2. action：业务代码、异步请求。

    3. 限制： 定位不同，二者有着不同的限制。
        1. mutation：必须是同步执行。
        2. action：可以是异步，但不能直接修改state；

    4. 调用：
        mutation： 使用 this.$store.commit() 调用；
        action： 使用 this.$store.dispatch() 调用；
```

7. Vuex 存储的数据浏览器刷新数据丢失问题

```
    Vuex 本身不是持久化存储的数据，Vuex是一个状态管理仓库（state：全局属性）==》就是存放全局属性的地方
    实现持久化存储：1. 自己封装到localStorage
                  2.使用插件：'vuex-persistedstate' 插件
```

VueRouter

1. ☆ vue-router 有哪几种钩子函数？(导航守卫)☆

```
    1. 全局性的守卫：
        1. router.beforeEach :     全局前置守卫，路由跳转前触发，参数 to、from、next
        2. router.beforeResolve : 全局解析守卫，路由跳转前触发， 参数 to、from、next
        3. router.afterEach :      全局后置守卫，路由跳转完成后触发，参数包括to、from，没有 next

    2. 路由独享的守卫：
        1. beforeEnter :           路由跳转前触发，如有beforeEach，则在其后触发，参数to、from、next

    3. 组件内的守卫：
        1. beforeRouteEnter :     在组件 beforeCreate 前触发，参数 to、from、next。(不能使用this，可以使用next中的vm )
        2. beforeRouteUpdate :    (vue-router 2.2 新增) 在当前路由改变，但是依然渲染该组件是调用，参数 to、from、next
        3. beforeRouteLeave :     导航离开该组件的对应路由时被调用，参数 to、from、next
```

2. vue-router 如何响应 路由参数 的变化?

```
    1. 复用组件时，相对 路由参数的变化 作出响应式， 可以使用 watch 监听 $route 对象；
    2.使用 （vue-router 2.2） 中引入的 beforeRouteUpdate 守卫
```

3. ☆ vue-router 路由的两种模式？☆

```
    1. hash模式：监听 onhashChange 实现；
    2. history模式： 使用 h5的API pushState ( ) 和 replaceState ( ) 和 onpopstate 监听URL变化
    两者的区别：
        1. 原理： 同上
        2. url: hash模式 url 中带有 '#' 号， history模式不带有 '#';
        3. 兼容: hash模式 兼容 IE8， history 模式 兼容 IE10
        4. 跳转: hash模式 只能修改'#'后面的内容，history模式 只要是同源的就可以
    补充 区别：
        1. 关于找不到页面：
            history：找不到页面也会向后台发送一次GET请求；
            hash：不会向后台发送请求
        2. 关于项目打包自测：
            history：默认情况看不到内容
            hash：可以看到内容
```

4. VueRouter传递参数 query 和 params 的区别？

```
    1. 传参： query 传参会在url 中显示; params 传参不会在url中显示;
    2. 刷新： query 刷新后 数据不会丢失; params 刷新后数据会丢失;
    3. 引入： query传参要用path、name来引入; params传参要用name来引入 ( 如果是path， 过去的参数为undefined ) ;

    PS： params传参 不想刷新后数据丢失，需要在 router 中在该路径后面配置'/:id'
```

5. router 和 route 的区别？

```
    $router: 是 VueRouter 的实例，相当于一个全局的路由器对象，作用是进行路由跳转的；
    $route: 是跳转路由对象，每一个路由都会有一个route对象，是一个局部的对象，可以获取对应的值；
```

6. 导航故障

```
    这个错误是 vue-router 内部错误，没有进行 catch 处理，导致的编程式导航跳转问题，向同一地址跳转时会报错的情况 ( push 和replace 都会导致这个情况的发生 ) 。
    解决：
        方案一：
            安装 vue-router 3.0 以下版本，先卸载 3.0 以上版本然后再安装旧版本 。
            npm install vue-router@2.8.0 -S
        方案二：
            针对于路由跳转相同的地址添加 catch 捕获一下异常。
            this.$router.push({path:'/register'}).catch(err => { console.log(err)
```

```
})
        方案三：
            在路由 router 里面加上以下这段代码，（重写push 和replace方法）
            // 解决编程式路由往同一地址跳转时会报错的情况
            const originalPush = VueRouter.prototype.push;
            const originalReplace = VueRouter.prototype.replace;

            // push
            VueRouter.prototype.push = function push(location, onResolve,
onReject) {
            if (onResolve || onReject)
                return originalPush.call(this, location, onResolve, onReject);
            return originalPush.call(this, location).catch(err => err);
            };

            //replace
            VueRouter.prototype.replace = function push(location, onResolve,
onReject) {
            if (onResolve || onReject)
                return originalReplace.call(this, location, onResolve, onReject);
            return originalReplace.call(this, location).catch(err => err);
            };
```

Axios

1. 说**一下你**对**axios**拦**截器的理解**

```

```

网络

1. ☆ **http** 状态码；☆

```

    /*
        200:  请求成功
        204：请求成功，但没有资源返回

        301：永久重定向
        302：临时重定向
        304：资源未修改，服务不会返回资源

        400：请求报文中出现语法错误
        401：未经授权，禁止访问
        403：没有权限，禁止访问
        404：没有找到资源

        500：服务器内部错误
```

```
        503： 服务器在维护或超负荷运行
    */
```

2. 原生ajax 、 fetch 、axios 的区别；

```

```

3. ☆ HTTP和HTTPS协议的区别? ☆

```
    1. https 协议需要 ca 证书，费用较高； 而 http 协议不需要；
    2. https 协议具有安全性的SSL加密传输协议； http协议是超文本传输协议，信息是明文传输；
    3. https 协议是 443 的端口； http 协议是 80 的端口；
    4. https 协议是具有SSL和HTTP协议构建的可进行加密传输、身份认证的网络协议，比HTTP更加安全；http 协议连接很简单,是无状态的;
```

4. ☆ GET和POST请求的区别? ☆

```
    1. 获取、提交数据：
        GET： 多用于获取数据；
        POST： 多用于提交数据；
    2. 传递数据大小：
        GET：请求在URL中传送的参数是有长度限制的；
        POST：理论上是不会受限制的，上传文件通常要使用post方式；
    3. 安全性：
        GET：请求的数据会显示在 URL 中。
        POST： 请求的数据放在 Request body中。
    4. 浏览器历史记录：
        GET：会被完整保留在浏览器历史记录中；
        POST：不会被记录
    5. 编码：
        GET：只能进行url编码
        POST：支持多种编码方式
    6. 数据类型：
        GET：只接受ASCII字符
        POST：没有限制
```

5. ☆ 输入 url 后，按下回车后到生成网页的过程 ☆

详情

```
    简易版：
        输入地址 ==> 查看 浏览器缓存、系统缓存、路由缓存（有缓存的话，显示页面 结束）
```

==> 域名解析 ==> 建立 TCP 链接、三次握手 ==> 浏览器发送http请求 ==> 服务器返回数据 ==> 浏览器接受数据 ==> 浏览器渲染，同时设置缓存、关闭TCP连接

详情版：

1. 浏览器开启处理请求：浏览器开启一个线程处理请求，对url进行判断如果是http协议就按照web方式处理。
2. 域名解析： 浏览器自身DNS -> 系统DNS、host -> 路由器缓存 -> 运营商DNS、根域 -> com域的顶级域的IP地址
3. 浏览器向域名对应的IP地址服务器发起TCP连接；
4. TCP/IP连接建立起来以后，浏览器就可以向服务器发送HTTP请求了；
5. 服务器端接受到了这个请求后，根据路径参数，经过后端的一些处理之后，把处理后的结果返回给浏览器；
6. 浏览器拿到了完整的HTML页面代码或者其他的数据结果后，浏览器开始下载HTML文档，同时设置缓存并关闭TCP连接
7. 浏览器根据拿到的资源对页面进行渲染，最终把一个完整的页面呈现给用户。

6. 协商缓存 和 强缓存

协商缓存（对比缓存）：

协商缓存（也叫对比缓存）：是由服务器来确定资源是否可用，所以客户端与服务器端要通过某种标识来进行通信；

原理：浏览器第一次发出请求时会带上字段（Last-Modified或者Etag），则后续请求则会带上对于的请求字段（if-modified-since或者if-none-Match），服务器对标识进行对比决定是否使用缓存

Last-Modified 和 Etag 的区别？

Last-Modified/if-modified-since 是用时间作为标识， 缺点：1.精确度回到秒级 2.文件会每隔一段时重新生成，内容没有改变，Last-Modified/if-modified-since 会被修改，从而会重新发送完整的请求。

Etag/if-none-Match：是用 字符型的唯一标识，会更加的精确，也解决了文件重新生成的问题。Etag 优先于 Last-Modified

强缓存（本地缓存）：

强缓存（本地缓存）：是由服务器决定是否将资源进行缓存，如果决定缓存，在设置的时间内就不会发起HTTP请求，分为 expires、cache-control

原理：完全是由服务器决定，前端不需要进行操作。

expires： 用于表示资源的过期时间的请求头字段，值是一个绝对时间，是由服务器端返回的。缺点：expires是根据本地的时间来判断的，假设客户端和服务器时间不同，会导致缓存命中误差

cache-control：是由服务器决定缓存到期时间， cache-control 优于 expires；

PS：强缓存和协商缓存同时存在时，先对比强缓存是否还再有效期，如果强缓存生效则对比协商缓存，即强缓存 > 协商缓存