HTML

1、语义化标签：标签带有一定的语义，让阅读代码的人第一眼就大致知道这个标签内部的代码内容。例如h1-h6的标题标签，P文本标签等

2、移动端适配：移动端适配，meta标签设置视口name=‘viewport’，content=‘width=device-witch’，initial-scale=1.0、设置根字体大小、使用flex + rem + gird 进行布局

HTML5

1、H5视频、音频的使用：使用radio、audio标签，并结合js的dom操作

CSS

1、将一个文本种的英文转为大写，（不可回答JS，因为js效率低）CSS里面text-transform属性uppercase

2、选择器：ID选择器 属性选择器 类选择器（class和伪类） 标签选择器 通配符\*

3、选择器优先级： ！import（最高） id（100） class、属性、伪类（10） 标签（1） 通配符（0/无优先级）

3、属性选择器8种：

4、Css表示颜色的几种方式：1、颜色关键字如red；2、rgb函数；3、rgba函数（较rgb多个a透明度）；4、hsl色调；5、hsla（较hsl多个a透明度）

CSS3

1、flex和grid：flex和grid都是弹性布局方式，flex是一维布局 grid是二维布局

2、动画animation属性：

**name：**动画名称，搭配@keyframes

**@keyframes：**定义动画方向

**iteration-count：**动画循环次数

**Duration：**动画完成时间，单位为秒

**timing-function：**动画的速度曲线（贝塞尔曲线）

**Delay：**延迟执行时间

JAVA SCRIPT(ES5)

1、JS中8种数据类型： **基本类型：**数字、bool、字符串 ；**引用类型：**数组、对象 ；null 、 undefind 、

Es6新增symbol

2、什么是闭包，闭包是一个函数嵌套一个函数，将内部函数return出去，将其的作用域暴露给外部使用。闭包并没有固定格式，只要符合这种思想的代码就是闭包

3、递归：函数直接、间接调用自身；三个阶段：展开阶段、到达边界、返回阶段，存在问题：递归展开阶段会申请内存，大幅度占用内存资源，到达边界后回归则释放内存。

4、检查对象是否包含这个元素 1、in关键字（会检测继承来的属性和方法），2、has own propthy

5、逻辑与&&逻辑或|| or 短路与&&短路或||：逻辑与/短路与：两者为真则为真，一假全假，使用时逻辑与将假放左侧；逻辑或/短路或：一者为真则全为真，使用时逻辑或将真放左侧

6、Type of：检测并返回一个变量的数据类型，

7、创建一个对象：New关键字 new obj 或者直接={}

8、对象使用时注意事项：JS中对象可以任意添加成员，访问不存在成员则返回undefined

9、回调函数：把一个函数作为参数传递给另一函数，传递的是函数的引用，在使用者那里定义该函数

10、运算符和优先级：算数运算符：一维：+ - \* / % 二维：+ + - - ；优先级：没有严格的规律，在优先级明确的情况下使用圆括号包裹，先进行运算

11、转义符：在使用时需要输入特殊字符，因为特殊字符不能直接输入，就需要\（反斜杠）对特殊字符进行转义

12、如何遍历一个数组：普通for循环、加强型for （for in遍历下边、for of遍历元素）、数组中的forEach方法

13、如何遍历一个对象：使用加强型for in，对象实现了iterable接口的可以使用 for of遍历获取对象

14、Delete关键字的使用：删除，对象中使用则删除对象成员（根据面向对象思想，因对象属性是对象的特征，不建议直接删除对象成员，可在类定义的时候就不进行定义），数组中则删除对应下标的元素（长度不变，占位undefined）

15、自执行函数：程序加载时自己执行一次，是一种匿名函数，用圆括号包裹，建议在自执行函数前后都加上分号‘；’

16、几种定时器：settimeout 只执行一次， setinterva 循环执行，清除定时器，在声明定时器时接受定时器的ID值，通过clear清楚

17、任务队列：一种执行机制，由于js是单线程，导致执行异步任务时.......，事件队列，渲染队列，只有定时器的任务队列可操作（在主程序执行完毕后执行的队列），

18、如何获取DOM子节点：父节点.childNode（会获取其他节点：文本、注释、属性节点） .children（只获取元素节点）

19、获取元素距顶部、左侧的距离：offset top offsetleft ，在使用时会受定位父级的影响，通过递归调用方法获取上一级定位父级距离左侧和顶部的距离，此时存在缺失定位父级的边框厚度距离，需要加上定位父级的client Left 、 clientTop

20、事件冒泡和事件捕获：①事件冒泡：当事件触发时，会将事件层层向父级传递，触发父级同类型事件的一种现象；②事件捕获：当事件触发时，事件会从外层父级向内层子集逐层触发相同类型的事件

21、所有事件都有冒泡和捕获吗？ 不是，例如onmouselive onmouseenter

22、事件委托：利用事件冒泡的现象，将事件绑定给父元素，所有子元素都可触发该事件，后续所有新增的子元素都无需重新添加事件，提高代码简洁性，提高效率。

23、有几种绑定事件的方式？1、普通赋值，onclick=function 2、addeventlisener（事件类型，事件处理函数，options/usercapture）

options选项：

①、capture：一个布尔值，会在该类型的事件捕获阶段传播到该 事件对象时触发

②、once：一个布尔值，表示事件处理函数在添加之后最多只调用一次。如果为 true，listener 会在其被 调用之后自动移除。

③、passive：设置为 true 时，表示 事件处理函数 永远不会调用 preventDefault()，调用则报警告。

④、signal ：[AbortSignal](https://developer.mozilla.org/zh-CN/docs/Web/API/AbortSignal)，该 AbortSignal 的 [abort()](https://developer.mozilla.org/zh-CN/docs/Web/API/AbortController/abort" \o "abort()) 方法被调用时，该事件的事件处理函数会被移除。

usercapture选项：

①、一个布尔值,为true时捕获，false时冒泡，默认是false冒泡

24、让一个函数执行指定次数：①定义全局变量指定次数，②在函数内添加计数器，函数每执行一次，计数器内改变全局变量的值（每运行一次自减1），④添加if判断，全局变量次数为0时，执行this=空，销毁函数。

25、元素克隆： node.cloneNode(ture’拷贝元素的子集’) node：要被克隆的节点；

26、任务队列：解决JS中单线程（需求决定，网页展示数据需求），因需求改变，需要进行异步操作，推出任务队列，渲染队列，事件队列，定时器队列（仅有可操作），使用定时器时，自动入列，在主进程完成后执行。

27、数组去重：

第一种方法：使用for循环，当前值与后面的所有值对比，相同的不进行处理，不同的则通过push方法添加到新数组中；

第二种方法：使用ES6中Set方法去重（set方法是类似数组的集合，值唯一）；

第三种方法：使用ES6中Map方法去重（Map方法类似对象‘键值对’的集合，键唯一）Map方法中的参数是二维数组需要对原一维数组进行处理，再通过加强for遍历，将二维数组的第0号下标展开到新数组中

28、防抖和节流：

防抖节流主要是应对连续触发的事件触发频率太高的问题  
防抖：给目标事件方法添加一个时间限定，setTimeout延迟执行时机  
节流：在防抖的基础上，添加一个控制器，让函数执行一次后，在某个时间段内暂时失效，  
过了这段时间后再重新激活效果；如果短时间内大量触发同一事件，那么在函数执行一次之后，该函数在  
指定的时间期限内不再工作，直至过了这段时间才重新生效

29、**作用域和作用域链，**

作用域在es5的时候分为全局和函数作用域，es6引入了代码块的概念和let关键字，作用域就是全局作用域和局部作用域

作用域链：子集中使用了一个未在该子集声明的变量时，会向父级查找，逐级向上直至最顶层，没有则报undefined

30、**js的执行顺序：**在页面onload后执行src引入的js代码，在js代码内先执行自执行函数

31、This指向问题：全局函数中this指向window，事件中的this指向发生事件的dom对象，箭头函数指向箭头函数所在上下文，定时器指向window

JAVA SCRIPT (ES6)

1、Promis

Promise修饰的函数是异步函数，promise有3中状态，进行中、成功、失败，可通过.then和.catch获取成功状态时的回调函数和失败状态时的回调函数；

Promise提供的2个方法：

①、promise.all方法（一个集合中全成则成，一败全败，返回失败的函数回调），

②、promise.race方法（一个异步集合中第一个确认状态的，返回第一个的函数回调，无论成功还是失败，“赛跑”）

2、Async 、await

Async修饰的函数是异步函数，默认返回一个pormise状态，await只在async修饰的函数内使用，用于

等待一个异步操作的完成

3、作用域：es6以前：函数作用域和全局作用域 es6：函数作用域和局部作用域

4、可变参数（ES6：...args）：除去形参列表中指定的形参以外的参数，形参中有且只有一个，放在形参最末尾。

5、深拷贝一个对象：使用object对象提供的assign方法 object . Assign（{ }，被拷贝的对象）

6、解构语法：用于数组和对象，let[n1,n2,n3 ]=[1,2,3]依次解构 对象：let { key }={key：value}，解构对象的 key必须相等

7、Set、map：是一个对象、集合，set不允许值重复，map不允许key重复

8、函数默认值：函数形参给定默认值，在函数使用形参是无对应实参传入时则使用默认值

9、模块导入（import）、导出（export / export default）：

导入（import）：

①导入的是export 所导出的值时，在导入时需要添加花括号：import { xxx } from ‘.....’;

②导入的时export default所导出的对象时，导入时不需要添加花括号：import xxx from ‘.....’

导出（export / export default）：

①一个文件或模块中，可以有多个export，但只允许有一个export default

①export 导出的是跟着的值；

②export default导出的是一个对象，其中的default指的是对应的导出接口变量。

10、ES6新特性

新增了let、const、箭头函数、class类语法以及constructor构造方法

11、箭头函数和普通函数的区别

①、This指向：全局函数中this指向window，箭头函数指向箭头函数所在上下文，

②、箭头函数不能new，无法作为构造函数使用

③、箭头函数没有prototype（原型继承）

④、箭头函数没有arguments（可变参数）

浏览器存储

1、Cookie：理解为文件存储的一种方式，存储在客户端浏览器，通多DOM操作获取和设置cookie，是挂载在浏览器地址中，一般是不超过4Kb的文本，cookie需要设置有效期，如不设置则为会话有效期，cookie会往返于服务端与客户端之间，常用于实现记住登录状态的功能以及保存用户名和密码，可以进行关闭，bom中cookie enable

2、Local storereage seasson storereage 的区别：都是储存客户端，一个长久储存，一个会话有效期储存

3、强缓存cache-control与协商缓存Last-Modified

①、浏览器在请求某一资源时，会先获取该资源缓存的header信息，判断是否命中强缓存（cache-control和expires信息），若命中直接从缓存中获取资源信息，包括缓存header信息；本次请求根本就不会与服务器进行通信；

②、如果没有命中强缓存，浏览器会发送请求到服务器，请求会携带第一次请求返回的有关缓存的header字段信息（Last-Modified/If-Modified-Since和Etag/If-None-Match），由服务器根据请求中的相关header信息来比对结果是否协商缓存命中；若命中，则服务器返回新的响应header信息更新缓存中的对应header信息，但是并不返回资源内容，它会告知浏览器可以直接从缓存获取；否则返回最新的资源内容

VUE

1、vue中没有参数的指令：[v-cloak](https://cn.vuejs.org/api/built-in-directives.html" \l "v-cloak)（解决网络延迟双花括号） [v-memo](https://cn.vuejs.org/api/built-in-directives.html" \l "v-memo)（缓存） [v-once](https://cn.vuejs.org/api/built-in-directives.html" \l "v-once)（只执行一次） [v-pre](https://cn.vuejs.org/api/built-in-directives.html" \l "v-pre)（预文本格式） [v-slot](https://cn.vuejs.org/api/built-in-directives.html" \l "v-slot)（声明具名插槽）

2、动态组件：component 标签，属性值就是组件名，通过改变属性名，从而动态改变展示的组件页面解构不改变，只改变数据时，最大程度服用路由组件

3、v-for使用时注意：不建议v-if同时使用，指定一个KEY值，可以遍历数字、字符串、数组、对 象

4、计算属性跟普通函数的区别：计算函数 无参，有返回值，有缓存，当依赖的data数据发生改变时就会 调用，计算属性的使用：{{计算属性的名字}}

Vue - Router

Vue-router 的使用：1、安装vue-router路由；2、在跟组件中使用.use方法安装插件；3、创建路由实例，在路由实例中配置路由规则（{ path：’ / ’ ,【懒加载】component :  () => import('../views/AboutView.vue')

, 【子路由】children:[ { path: ’ home ’ , component : ‘ ‘ } ]}）；4、设置路由跳转连接（router-link标签）渲染出口（router-view标签）

Vue的路由守卫：

全局前置（before Each），

路由独享（before Enter），

全局解析（before Resolve），

路由后置守卫（after Each），

组件内守卫（before Route Enter、before Route Update、before Route Leave）

声明式导航和编程式导航的区别：声明式是标签router-link ，编程式是调用router.push（）

动态路由：在编写路由配置中path中第二段以后的路径（即第二段‘/’），通过’：’表示，当路由被匹配时，他的params 值将在每个组件中暴露出来，通过$router.params获取动态路由，

**面试题：**

项目中常用的路由守卫有哪些？

全局的守卫（全局前置守卫、全局解析守卫、全局后置守卫）

单个路由独享的

组件级的

全局前置守卫：只要路由有变化，全局前置守卫都会进行拦截

运用场景：每次路由跳转时，检查页面是否**初次访问**和**用户是否登录**（是否持有token），没有则通过return一个name或者url进行跳转回登录页。

全局解析守卫：

运用场景：是获取数据或执行任何其他操作（如果用户无法进入页面时你希望执行的操作）的理想位置，因为它是导航被确认前，且组件内守卫和异步路由组件被解析后才执行的。

全局后置守卫：

运用场景：路由跳转完成后触发。在全局前置和全局解析之后、组件内守卫开始之前执行。不接受next函数，不改变导航本身，对于分析、更改页面标题、声明页面等辅助功能以及许多其他事情都很有用

路由独享守卫：针对单个路由设置的守卫

运用场景：可进行判断用户的访问权限，有权限则正常访问，无权限则return false取消跳转，或者指向新的页面

组件内守卫：

before Route Enter：在渲染组件对应的路由时调用

before Route Update：当前路由改变，但该组件被复用时调用，组件已挂载，可访问this

before Route Leave：导航离开该组件时调用

路由守卫能做什么/作用？

答：在路由跳转时，通过逻辑控制，决定是否跳转，或者重新指向新的路由地址

Pinia

什么是Pinia:用于vue各组件之间数据共享的插件，

有3个主要内容：变量：state；计算属性：getter；函数：action

安装pinia插件：在根组件中，从pinia中解构出createPinia方法，通过.use(createPinia())

使用：

1、从pinia中解构出defineStore定义一个存储库；

2、在组件中导入存储库函数（存储库中export const跟着的那个就是导入函数名）；

3、const ‘Name’接收函数执行的返回的存储库；

4、在组件中使用‘Name’.xxx（xxx代表存储库中定义的变量、常量）

VueX

主要内容：store仓库（state数据/状态（store中包含的数据））；getter（store的计算属性）；mutation（内部定义修改state的函数）；action（批量提交mutation）；module（模块）

使用：

1、创建store文件，从vuex中解构createStore，const创建store实例（将5个成员写入）；

2、在main.js中导入store实例，并在根组件中.use注册store；

3、在组件中使用：

①通过store.state.xxx访问state中的数据成员；

②通过store.commit（‘mutation中定义的函数名’字符串表示）来触发mutation；

③通过store.dispatch触发action中定义的函数（函数内使用【形参.commit(mutation中定义的函数名)】）；

④getter中定义的函数，传入state，在组件中通过【store.getter.函数名】使用

⑤store允许分割成模块module，每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块，最后在store中的module对象内汇总，key即store中module内对应子模块key，value为const/let声明的子模块名（对象语法{key ：value}），在根组件中，通过store.registerModule（‘组件名’，组件本身）进行注册，在组件内通过store.state.‘组件名’进行使用

React

一、JSX语法，是一个 JavaScript 的语法扩展，能够在JS中编写类似HTML的标签，需要babel编译。

二、react是mvc，专注视图层，数据一旦渲染，再更改数据则不会重新渲染，如需要重新渲染，必须调用 this.setState / 重新调用render / 强制刷新 forceUpDate

三、React通过对DOM的模拟，最大限度地减少与DOM的交互，大幅度提高效率。

四、使用：

①在JSX中，通过单花括号包裹，插入JS表达式 <h1>{ JS表达式 }</h1>

②在JSX中，bool、null、undefined 不能直接渲染到DOM，数组则直接展开所有内部元素

③在DOM中直接插值是安全的，因为ReactDOM在渲染时会将所有的内容转换成字符串进行渲染，有效防止XSS跨站脚本攻击

④使用花括号 {} 为属性插入一个JS表达式 <h1 className=’box’ title={ text } >{ JS表达式 }</h1>

⑤在JSX中使用style来设置样式时,style的值必须是一个对象,各个样式属性作为key(驼峰命名),样式值为value⑥事件：

1、使用 JSX 语法时你需要传入一个函数作为事件处理函数，而不是一个字符串。<button onClick={fun1}>按钮</button>

2、JSX 可以通过箭头函数的方式向事件回调函数中传递数据<button onClick={ ev => fun1( n1,n2,n3 ) }>按钮</button>

3、通过bind函数向传递数据<button onClick={ ev => fun1.bind( n1,n2,n3 ) }>按钮</button>

五、组件：

1、函数式组件（无生命周期、无状态）

function Name（props）{

return ( jsx标签内容 )

}

2、createClass ES5语法定义组件（过渡版本、15.5后废弃）

const Name = React . createClass( {

render:function(){

return （ JSX标签 ）

}

} )

3、ES6 Class语法定义组件

class Name extends React . component{

constructor(props){

super(props)

}

render( ){

Return( JSX语法)

}

}

六、props

1、函数式组件通过props.value使用value数据

2、ES5\ES6声明的组件都通过this.props.value使用value数据

3、类型验证（15.5移入prop-types库）

①createClass组件：propTypes：{ value:React.PropTypes.string }

②函数式组件和类组件：引入prop-types库

非脚手架环境：script标签内 crossorigin src="https://unpkg.com/prop-types@15.7.2/prop-types.js"

4、通过在父组件内子组件标签中设置 属性名=属性值 的方式，实现父组件向子组件传递数据，通过传递在父组件中定义的函数引用，在子组件内调用该引用，传值，实现子组件向父组件传值

5、内容分发：标签中属性传递的是一个JSX格式内容

①默认内容分发：写在父组件中子组件标签对中的内容，在子组件内通过this.props.children

②指定内容分发：写在父组件中子组件标签属性中，属性名=属性值，在子组件中通过this.props.属性名

七、组件内事件问题：组件中事件回调函数使用this，它的指向为undefined。

处理方法：1、使用bind函数修正this指向：<button onClick={ this.btnClick.bind ( this ) }> 按钮 </button>

2、在构造方法中修改，在组件中直接调用该方法：this. btnClick = this.btnClick.bind( this )

3、使用箭头函数<button onClick={ ev=>this.btnClick( ev ) }> 按钮 </button>

八、组件内的数据state及setState（修改数据）：

1、state

①ES5：get Initial State : function( ){

return {

Value：xxx

}

}

②ES6: 在构造方法中初始化this . state={ value：xxx }

2、setState：更新数据是异步的，可以监听setState中的回调函数来获取新值，setState会触发视图更新

①this . setState ( { data : newData } )

//通过监听回调函数，实时拿到改变后的新值

②this . setState ( { val=>{

val . data=newData

return val

}

} )

九、Ref属性，16.3之前使用回调函数方式，ref={el =>{this. Xxx=el}}

十、安全模式StrictMode的作用：①识别不安全的生命周期；②关于使用过时字符串ref、API的警告；③使用废弃的findDOMNode方法的警告；④检测意外的副作用；⑤检测过时的context API；⑥确保可复用的状态

React - router - dom（路由）

1、类：

①、route（抽象类，路由对象，匹配浏览器地址栏url,所有路由记录都必须嵌套在该组件内）

2、reactRouter里的路由必须包含在根里（browserRouter、HashRouter、MemoryRouter....）

Routes标签里只能写路由的成员

3、设置路由规则：包含“路由匹配规则”和“路由渲染组件”，routes里面只能有route

包裹关系（代码展示）：

<browserRouter>

<Routes>

<Route path=’路径1’ element={<组件1/>} />

<Route path=’路径2’ element={<组件2/>} />

<Route path=’路径3’ element={<组件3/>} />

......

</Routes>

</browserRouter>

4、Link 和NavLink 的导航区别

①相同：在渲染时都渲染为a标签

②不同：NavLink 在渲染时会有一个active的类，他是动态的，只要url匹配，就会添加上

③自定义选中时active类：在标签中className={ ( { isActive} ) }=> isActive ? “类名” : “” ，给类名设置样式 就可以实现动态显示选中效果

5、路由重定向。因重定向是最低匹配优先级，在路由规则最后面加<Route path=’\*’ element={<组件1/>} />

6、动态路由。动态匹配路由规则中第二段的。可根据变量的改变，在不改变页面样式的情况下动态展示不同内容

7、嵌套路由渲染出口“<Outlet />”,如果存在嵌套路由，则需要在父路由的模板内容中使用<Outlet />标签指定子路由的渲染出口。注意：一定是在父路由的模板内容中指定渲染出口

Redux

一、功能：Redux：react共享数据 ，3个核心：store存储数据 reducer修改数据 action触发用户的操作

二、使用：

1、npm安装redux

React面试题

1、Jsx怎么实现列表渲染？基于数组在jsx中单花阔号{数组}是直接展开的特性，在渲染之前用数组.Map方法，将数组成员进行处理成jsx格式的数据成员，map方法会返回一个处理后的新数组，再在模板中使用花括号将新数组中的jsx元素展开

**原型和原型链，**

原型就要讲到继承，继承是子集使用父级定义的属性和方法，当子集使用了一个当前子集未定义的属性或者方法时，会向当前子集的原型中查找，再向父级中的普通属性和方法中查找，再向父级的原型中查找，这种查找过程类似链条相扣，我们将之称为原型链

**http协议，**

一个无状态、无连接的基于TCP协议演变的协议，客户端在获取/未获取到请求的数据都会视为一次链接，

每一次链接都是新链接，有几种常用的状态码：1XX信息，2XX成功，3XX重定向，4XX客户端错误，5XX服务器端错误

Dns-解析匹配ip地址，将数据转发给匹配的地址，匹配端口应用、路径，

**vue的原理，**

vue 基于 MVVM思想的框架,MVVM 起源于MVC（Model View Controller）架构,

(Model：数据模型,View：视图,Controller：控制器)

MVVM （Model View  View Model）双向数据绑定

明显体现就是vue中的v-model双向数据绑定，修改data，视图也会改变

单页面应用程序

整个应用只有一个页面：按需加载（一个页面就是一个JS模块）

Vue3中data为什么是一个函数而不是一个对象？

Js中对象/实例是有构造函数new出来的，而对象又是引用数据类型，在vue中，一个vue组件就是一个实例，如果data是对象的话，复用的组件中的data都是同样的引用，当某个组件的data值更改时则会影响其他组件的data。函数是返回一个data对象，当组件复用的时候，则相当于在内存中新开辟了一个data对象空间，确保各组件的data独立性，使组件自己维护自己的数据。

HTTP

一、http协议：

①是一个网络请求协议，是一种无状态的协议 每次请求都是新的请求，HTTP协议是为了网页而生

②一个无状态、无连接的基于TCP协议演变的协议，客户端在获取/未获取到请求的数据都会视为一次链接，

每一次链接都是新链接，有几种常用的状态码：1XX信息，2XX成功，3XX重定向，4XX客户端错误，5XX服务器端错误

二、Ajax：

Ajax的9种请求方式：

1. GET 方法请求一个指定资源的表示形式. 使用GET的请求应该只被用于获取数据.
2. HEAD 方法请求一个与GET请求的响应相同的响应，但没有响应体.
3. POST 方法用于将实体提交到指定的资源，通常导致在服务器上的状态变化或副作用.
4. PUT 方法用请求有效载荷替换目标资源的所有当前表示。
5. DELETE 方法删除指定的资源。
6. CONNECT 方法建立一个到由目标资源标识的服务器的隧道。
7. OPTIONS 方法用于描述目标资源的通信选项。
8. TRACE 方法沿着到目标资源的路径执行一个消息环回测试。
9. PATCH 方法用于对资源应用部分修改。

Ajax请求：

1、创建请求对象

Let xml = new XMLHttpRequest

2、设置请求参数

Xml.open( ‘请求方式’ , URL , bool值【true为异步，false为同步会堵塞主程序】 )

\*\*post请求需设置请求头： Xml.setRequestHeader("Content-type","application/x-www-form-urlencoded");

3、发送请求参数（post请求时需传入一个formData对象）

Xml.send( ‘formData’ )

4、等待响应

Xml.onReadyStateChange = function(){

if(4===this.state && 200===this.status ) {

Try{

resolve( JSON.parse( this.responseText) )

}

Catch{

resolve( this.responseText )

}

}

}

三、一个http请求的流程：

1、地址栏输入地址

2、DNS解析器解析域名匹配ip地址

3、将数据转发给匹配的地址

4、匹配应用端口、匹配文件路径

5、①匹配成功：成功获取到数据并渲染到页面中；②匹配失败：获取数据失败，返回5xx状态码

GIT的使用

1、创建仓库（新建文件夹）

2、在仓库内初始化仓库 ？待确认

3、git status 查看新增文件

4、git add . or git add xxx文件，将文件确认为待提交状态（必须）

5、git commit -m “提交文件的备注信息” 提交文件

6、git log 查看提交文件的id和备注信息

7、git reset --hard 文件ID 恢复到某次提交的文件状态

8、提交到GitHub仓库流程：

①创建分支：git branch -M 分支名

②git remote add origin 提交的仓库http地址

③git push -u origin 提交的分支名

9、从仓库拉取代码：git clone 仓库地址 .

10、添加上游代码库：git remote add upstream 上游代码库地址

11、创建分支git checkout -b 分支名（创建并进入分支）

Uni-app