1. axios原理
2. axios是什么？

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中。

1. axios有什么特性？（不得不说面试被问到几次）

* 从浏览器中创建 XMLHttpRequests从 node.js
* 创建 http 请求
* 支持 Promise API
* 拦截请求和响应转换请求数据和响应数据
* 取消请求
* 自动转换JSON 数据
* 客户端支持防御 XSRF

实际上，axios可以用在浏览器和 node.js 中是因为，它会自动判断当前环境是什么，如果是浏览器，就会基于XMLHttpRequests实现axios。如果是node.js环境，就会基于node内置核心模块http实现axios

2、**axios有哪些常用方法？**  
答：  
一、axios.get(url[, config])   //get请求用于列表和信息查询  
二、axios.delete(url[, config])  //删除  
三、axios.post(url[, data[, config]])  //post请求用于信息的添加  
四、axios.put(url[, data[, config]])  //更新操作  
  
**3、说下你了解的axios相关配置属性？**答：  
`url`是用于请求的服务器URL  
`method`是创建请求时使用的方法,默认是get  
`baseURL`将自动加在`url`前面，除非`url`是一个绝对URL。它可以通过设置一个`baseURL`便于为axios实例的方法传递相对URL  
`transformRequest`允许在向服务器发送前，修改请求数据，只能用在'PUT','POST'和'PATCH'这几个请求方法  
**`headers`是即将被发送的自定义请求头**

headers:{'X-Requested-With':'XMLHttpRequest'},  
**`params`是即将与请求一起发送的URL参数，必须是一个无格式对象(plainobject)或**URLSearchParams对象

params:{  
ID:12345  
},

1.**为什么 axios 既可以当函数调用，也可以当对象使用，比如axios({})、axios.get。**

答：axios本质是函数，赋值了一些别名方法，比如get、post方法，可被调用，最终调用的还是Axios.prototype.request函数。

**2.简述 axios 调用流程。**

答：实际是调用的Axios.prototype.request方法，最终返回的是promise链式调用，实际请求是在dispatchRequest中派发的。

**3.有用过拦截器吗？原理是怎样的？**

答：用过，用axios.interceptors.request.use添加请求成功和失败拦截器函数，用axios.interceptors.response.use添加响应成功和失败拦截器函数。在Axios.prototype.request函数组成promise链式调用时，Interceptors.protype.forEach遍历请求和响应拦截器添加到真正发送请求dispatchRequest的两端，从而做到请求前拦截和响应后拦截。拦截器也支持用Interceptors.protype.eject方法移除。

**4.有使用axios的取消功能吗？是怎么实现的？**

答：用过，通过传递config配置cancelToken的形式，来取消的。判断有传cancelToken，在promise链式调用的dispatchRequest抛出错误，在adapter中request.abort()取消请求，使promise走向rejected，被用户捕获取消信息。

**5.为什么支持浏览器中发送请求也支持node发送请求？**

答：axios.defaults.adapter默认配置中根据环境判断是浏览器还是node环境，使用对应的适配器。适配器支持自定义。

1. http状态码

|  |  |  |
| --- | --- | --- |
| 1\*\* | 信息，服务器收到请求，需要请求者继续执行操作 | |
| 2\*\* | 成功，操作被成功接收并处理 | |
| 3\*\* | 重定向，需要进一步的操作以完成请求 | |
| 4\*\* | 客户端错误，请求包含语法错误或无法完成请求 | |
| 5\*\* | 服务器错误，服务器在处理请求的过程中发生了错误 | |
| 100 | Continue | 继续。[客户端](http://www.dreamdu.com/webbuild/client_vs_server/)应继续其请求 |
| 101 | Switching Protocols | 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议 |
|  | | |
| 200 | OK | 请求成功。一般用于GET与POST请求 |
| 201 | Created | 已创建。成功请求并创建了新的资源 |
| 202 | Accepted | 已接受。已经接受请求，但未处理完成 |
| 203 | Non-Authoritative Information | 非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本 |
| 204 | No Content | 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档 |
| 205 | Reset Content | 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域 |
| 206 | Partial Content | 部分内容。服务器成功处理了部分GET请求 |
|  | | |
| 300 | Multiple Choices | 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择 |
| 301 | Moved Permanently | 永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替 |
| 302 | Found | 临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI |
| 303 | See Other | 查看其它地址。与301类似。使用GET和POST请求查看 |
| 304 | Not Modified | 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源 |
| 305 | Use Proxy | 使用代理。所请求的资源必须通过代理访问 |
| 306 | Unused | 已经被废弃的HTTP状态码 |
| 307 | Temporary Redirect | 临时重定向。与302类似。使用GET请求重定向 |
|  | | |
| 400 | Bad Request | 客户端请求的语法错误，服务器无法理解 |
| 401 | Unauthorized | 请求要求用户的身份认证 |
| 402 | Payment Required | 保留，将来使用 |
| 403 | Forbidden | 服务器理解请求客户端的请求，但是拒绝执行此请求 |
| 404 | Not Found | 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面 |
| 405 | Method Not Allowed | 客户端请求中的方法被禁止 |
| 406 | Not Acceptable | 服务器无法根据客户端请求的内容特性完成请求 |
| 407 | Proxy Authentication Required | 请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权 |
| 408 | Request Time-out | 服务器等待客户端发送的请求时间过长，超时 |
| 409 | Conflict | 服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突 |
| 410 | Gone | 客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置 |
| 411 | Length Required | 服务器无法处理客户端发送的不带Content-Length的请求信息 |
| 412 | Precondition Failed | 客户端请求信息的先决条件错误 |
| 413 | Request Entity Too Large | 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息 |
| 414 | Request-URI Too Large | 请求的URI过长（URI通常为网址），服务器无法处理 |
| 415 | Unsupported Media Type | 服务器无法处理请求附带的媒体格式 |
| 416 | Requested range not satisfiable | 客户端请求的范围无效 |
| 417 | Expectation Failed | 服务器无法满足Expect的请求头信息 |
|  | | |
| 500 | Internal Server Error | 服务器内部错误，无法完成请求 |
| 501 | Not Implemented | 服务器不支持请求的功能，无法完成请求 |
| 502 | Bad Gateway | 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应 |
| 503 | Service Unavailable | 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中 |
| 504 | Gateway Time-out | 充当网关或代理的服务器，未及时从远端服务器获取请求 |
| 505 | HTTP Version not supported | 服务器不支持请求的HTTP协议的版本，无法完成处理 |

1. js原型与继承
2. http与https区别
3. http2
4. cors

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。

整个CORS通信过程，都是浏览器自动完成，不需要用户参与。对于开发者来说，CORS通信与同源的AJAX通信没有差别，代码完全一样。浏览器一旦发现AJAX请求跨源，就会自动添加一些附加的头信息，有时还会多出一次附加的请求，但用户不会有感觉。

因此，实现CORS通信的关键是服务器。只要服务器实现了CORS接口，就可以跨源通信。

浏览器将CORS请求分成两类：简单请求（simple request）和非简单请求（not-so-simple request）。

**对于简单请求**，浏览器直接发出CORS请求。具体来说，就是在头信息之中，增加一个Origin字段。上面的头信息中，Origin字段用来说明，本次请求来自哪个源（协议 + 域名 + 端口）。服务器根据这个值，决定是否同意这次请求。

如果Origin指定的源，不在许可范围内，服务器会返回一个正常的HTTP回应。浏览器发现，这个回应的头信息没有包含Access-Control-Allow-Origin字段（详见下文），就知道出错了，从而抛出一个错误，被XMLHttpRequest的onerror回调函数捕获。注意，这种错误无法通过状态码识别，因为HTTP回应的状态码有可能是200。

如果Origin指定的域名在许可范围内，服务器返回的响应，会多出几个头信息字段。

**非简单请求**是那种对服务器有特殊要求的请求，比如请求方法是PUT或DELETE，或者Content-Type字段的类型是application/json。

非简单请求的CORS请求，会在正式通信之前，增加一次HTTP查询请求，称为"预检"请求（preflight）。

浏览器先询问服务器，当前网页所在的域名是否在服务器的许可名单之中，以及可以使用哪些HTTP动词和头信息字段。只有得到肯定答复，浏览器才会发出正式的XMLHttpRequest请求，否则就报错。服务器收到"预检"请求以后，检查了Origin、Access-Control-Request-Method和Access-Control-Request-Headers字段以后，确认允许跨源请求，就可以做出回应。如**果浏览器否定了"预检"**请求，会返回一个正常的HTTP回应，但是没有任何CORS相关的头信息字段。这时，浏览器就会认定，服务器不同意预检请求，因此触发一个错误，被XMLHttpRequest对象的onerror回调函数捕获。控制台会打印出如下的报错信息。

**一旦服务器通过了"预检"请求，**以后每次浏览器正常的CORS请求，就都跟简单请求一样，会有一个Origin头信息字段。服务器的回应，也都会有一个Access-Control-Allow-Origin头信息字段。

1. jsonp
2. css垂直居中
3. 作用域问题
4. 闭包
5. this指针
6. 异步问题
7. 原型以及原型链
8. 浏览器的渲染原理
9. http tcp报文格式
10. 302 304区别
11. this理解
12. tagcanvas和webgl区别
13. 项目难点
14. 常用ES6语法  
    一、var 变量提升，即脚本开始运行 变量已经存在 但是没有值 所以会输出undefined  
    let 不会发生变量提升，不允许同一代码块重复声明同一变量  
    const 声明一个只读常量，不可重复声明

二、数组，对象，函数解构赋值

三、set 和Map

四、箭头函数和this

1 箭头函数只能用赋值式写法，不能用声明式，箭头函数对于常规函数最大的优化之处在于this。

2 this

* 1. 普通函数里默认为window对象
  2. 在箭头函数内部是词法作用域，由上下文决定，this是使用call方法调用函数时传递的第一个参数，它可以在函数调用时修改，在函数没有调用的时候，this的值是无法确定。
  3. 构造函数中的this，构造函数new之后会返回一个对象，这个对象就是this
  4. window.setTimeout()和window.setInterval()的函数中的this有些特殊，里面的this默认是window对象。

【总结】**简单总结一下：函数完整的调用方法是使用call方法，包括test.call(context, name)和obj.greet.call(context,name)，这里的context就是函数调用时的上下文，也就是this，只不过这个this是可以通过call方法来修改的；构造函数稍微特殊一点，它的this直接指向new之后返回的对象；window.setTimeout()和window.setInterval()默认的是this是window对象。**

3 箭头函数中的this

1. 箭头函数的特性一：默认绑定外层this
2. 箭头函数的特性二：不能用call方法修改里面的this
3. 箭头函数不会创建自己的this,它只会从自己的作用域链的上一层继承this

五、Promise使用  
 两个特点：对象的状不受外界 影响。一旦状态改变，就不会再变，任何时候都可以得到这个结果

21. href与src的区别

（1）请求资源类型不同：

href是超文本引用的简写，用来为当前元素和文档之间建立连接，常用的是link、a标签。

src会将指向的资源下载并引用到当前文档中，常用的标签有script，img，iframe标签。

（2）作用的结果不同：

href是为当前文档和引用资源建立联系；

而src是替换当前的元素。

（3）浏览器的解析方式不同

herf引用的资源，浏览器会将其识别为CSS文档，并行下载资源并且不会停止对当前文档的处理。

当浏览器解析到src时，会暂停其他资源的下载和处理，直接将该资源下载，编译，执行完毕，图片和框架也是如此，类似于将所指资源应用到当前内容。

1. **HTTP 1.0 & 1.1 & 2.0**
2. HTTP1.0 和HTTP1.1的区别

1.1 长连接(Persistent Connection)

  HTTP1.1支持长连接和请求的流水线处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，在HTTP1.1中默认开启长连接keep-alive，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点。HTTP1.0需要使用keep-alive参数来告知服务器端要建立一个长连接。

1.2 节约带宽

  HTTP1.1支持只发送header信息（不带任何body信息），如果服务器认为客户端有权限请求服务器，则返回100，客户端接收到100才开始把请求body发送到服务器；如果返回401，客户端就可以不用发送请求body了节约了带宽。

1.3 HOST域

  在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname），HTTP1.0没有host域。随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都支持host域，且请求消息中如果没有host域会报告一个错误（400 Bad Request）。

1.4缓存处理

  在HTTP1.0中主要使用header里的If-Modified-Since, Expires来做为缓存判断的标准，HTTP1.1则引入了更多的缓存控制策略例如Entity tag，If-Unmodified-Since, If-Match, If-None-Match等更多可供选择的缓存头来控制缓存策略。

1.5错误通知的管理

   在HTTP1.1中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

1. HTTP1.1 和HTTP2.0的区别

2.1 多路复用

HTTP2.0使用了多路复用的技术，做到同一个连接并发处理多个请求，而且并发请求的数量比HTTP1.1大了好几个数量级。HTTP1.1也可以多建立几个TCP连接，来支持处理更多并发的请求，但是创建TCP连接本身也是有开销的

2.2 头部数据压缩

  在HTTP1.1中，HTTP请求和响应都是由状态行、请求/响应头部、消息主体三部分组成。一般而言，消息主体都会经过gzip压缩，或者本身传输的就是压缩过后的二进制文件，但状态行和头部却没有经过任何压缩，直接以纯文本传输。随着Web功能越来越复杂，每个页面产生的请求数也越来越多，导致消耗在头部的流量越来越多，尤其是每次都要传输UserAgent、Cookie这类不会频繁变动的内容，完全是一种浪费。

  HTTP1.1不支持header数据的压缩，HTTP2.0使用HPACK（**HPACK，一种新型压缩器，可消除冗余报头字段，限制已知安全攻击的漏洞，并且在受限环境中使用有限的内存要求。**）算法对header的数据进行压缩，这样数据体积小了，在网络上传输就会更快。

2.3 服务器推送

  服务端推送是一种在客户端请求之前发送数据的机制。网页使用了许多资源：HTML、样式表、脚本、图片等等。在HTTP1.1中这些资源每一个都必须明确地请求。这是一个很慢的过程。浏览器从获取HTML开始，然后在它解析和评估页面的时候，增量地获取更多的资源。因为服务器必须等待浏览器做每一个请求，网络经常是空闲的和未充分使用的。

为了改善延迟，HTTP2.0引入了server push，它允许服务端推送资源给浏览器，在浏览器明确地请求之前，免得客户端再次创建连接发送请求到服务器端获取。这样客户端可以直接从本地加载这些资源，不用再通过网络。

1. session和cookie的作用与区别

**Cookie通过在客户端记录信息确定用户身份**，**Session通过在服务器端记录信息确定用户身份**。

Cookie的maxAge决定着Cookie的有效期，单位为秒（Second）。Cookie中通过getMaxAge()方法与setMaxAge(int maxAge)方法来读写maxAge属性。

如果maxAge属性为正数，则表示该Cookie会在maxAge秒之后自动失效。浏览器会将maxAge为正数的Cookie持久化，即写到对应的Cookie文件中。无论客户关闭了浏览器还是电脑，只要还在maxAge秒之前，登录网站时该Cookie仍然有效。

如果maxAge为负数，则表示该Cookie仅在本浏览器窗口以及本窗口打开的子窗口内有效，关闭窗口后该Cookie即失效。maxAge为负数的Cookie，为临时性Cookie，不会被持久化，不会被写到Cookie文件中。Cookie信息保存在浏览器内存中，因此关闭浏览器该Cookie就消失了。Cookie默认的maxAge值为–1。

如果maxAge为0，则表示删除该Cookie。Cookie机制没有提供删除Cookie的方法，因此通过设置该Cookie即时失效实现删除Cookie的效果。失效的Cookie会被浏览器从Cookie文件或者内存中删除，