**1.http请求头/响应头**

**1.1General Headers**

**Request URL：请求地址**

**Request Method：请求方法**

**Status Code：状态码**

**Remote Address：远程服务器地址**

**Referrer Policy 来源政策**

**No Referrer：任何情况下都不发送Referrer信息**

**默认策略：No Referrer When Downgrade：仅当协议降级（如HTTPS页面引入HTTP资源）时不发送Referrer信息。是大部分浏览器默认策略**

**Origin Only：发送只包含host部分的referrer**

**Origin When Cross-origin：仅在发生跨域访问时发送只包含host的Referer，同域下还是完整的。与Origin Only的区别是多判断了是否Cross-origin。协议、域名和端口都一致，浏览器才认为是同域**

**Unsafe URL：全部都发送Referrer信息。最宽松最不安全的策略**

**1.2Response Headers**

**Access-Control-Allow-Origin：标识允许哪个域的请求，设置为\*，服务器不会发送验证头和接受cookie，浏览器也不会发送cookie**

**Access-Control-Allow-Credentials：是否允许后续请求携带cookie**

**Access-Control-Max-Age: 预检结果缓存时间,也就是上面说到的缓存**

**Access-Control-Allow-Methods：允许的请求类型**

**Access-Control-Allow-Headers：允许的请求头字段**

**Cache-Control：是否进行缓存（public，private，no-cache）与浏览器协商**

**Pragma：同上**

**Expires ：失效时间，无效的日期表示已经过期，max-age设置该字段无效**

**Connection：keep-alive，保持tcp连接**

**Content-type：返回数据是什么类型**

**Content-encoding：内容编码**

**date：服务器发出该响应的时间**

**set-cookie：设置cookie**

**Last-Modified：最后被修改的时间**

**Etag：资源标识**

**Server：服务器软件名**

**Location：用来重定向接收方到非请求URL的位置来完成请求或标识新的资源**

**refresh：应用于重定向或一个新的资源被创造，在5秒之后重定向（由网景提出，被大部分浏览器支持）**

**1.3Request headers**

**Accept：可以接受的文件格式**

**Accept-Encoding：浏览器可以支持的压缩编码类型**

**Accept-Language：浏览器支持语言**

**Cache-Control：缓存策略**

**connection：keep-alive 保持Tcp连接**

**cookie：4k**

**origin：来源，发送请求的主机**

**Content-Length : 请求头的长度**

**host：请求的服务器域名和端口**

**referer：来源，用户在哪个url下发送的请求**

**user-agent：用户代理**

**2.AJAX，在不刷新页面的情况下，更新数据，利用的就是XMLHttpRequest低版本的IE是用ActiveXObject对象**

**3.XMLHttpRequest，浏览器为了能让js代码进行http(s)通信提供的对象**

**version1：只支持文本数据发送，无法读取和上传二进制文件**

**传送和接收数据时没有进度显示**

**收到同源策略影响，只能相同一域名下发送请求**

**version2：可获取服务端二进制数据/上传文件**

**FromData对象管理表单数据**

**可以设置请求时限/显示进度**

**跨域请求**

**3.1属性**

**XMLHttpRequest.readyState：请求状态码**

**XMLHttpRequest.status：响应状态**

**XMLHttpRequest.response：响应体**

**XMLHttpRequest.responseType：响应类型，枚举值**

**XMLHttpRequest.responseText：DOMString包含请求的响应**

**XMLHttpRequest.responseURL：响应URL**

**XMLHttpRequest.responseXML：响应xml/html**

**XMLHttpRequest.statusText：DOMString完整的响应文本**

**XMLHttpRequest.timeout：timeout**

**XMLHttpRequest.upload：上传进度**

**XMLHttpRequest.withCredentials：指定是否带cookie**

**3.2方法**

**XMLHttpRequest.abort()：如果请求已被发出，则立刻中止请求。**

**XMLHttpRequest.getAllResponseHeaders()：以字符串的形式返回所有用 CRLF 分隔的响应头，如果没有收到响应，则返回 null。**

**XMLHttpRequest.getResponseHeader()：返回包含指定响应头的字符串，如果响应尚未收到或响应中不存在该报头，则返回 null。**

**XMLHttpRequest.open()：初始化一个请求。该方法只能在 JavaScript 代码中使用，若要在 native code 中初始化请求，请使用 openRequest()。**

**XMLHttpRequest.overrideMimeType()：覆写由服务器返回的 MIME 类型。**

**XMLHttpRequest.send()：发送请求。如果请求是异步的（默认），那么该方法将在请求发送后立即返回。**

**XMLHttpRequest.setRequestHeader()：设置 HTTP 请求头的值。必须在 open() 之后、send() 之前调用 setRequestHeader() 方法。**

**3.3事件**

**Onreadystatechange：状态变化**

**Onloadstart：获取开始**

**Onprogress：数据传输进行中**

**Onabort：获取操作终止**

**Onerror：获取失败**

**Onload：获取成功**

**ontimeout：获取操作在用户规定的时间内未完成**

**onloadend：获取完成（不论成功与否）**

**4.axios**

**将XMLHttpRequest封装成Promise的http库**

**4.1使用方法：**

**axios.create({config})**

**axios.request(config)**

**axios.get(url[, config])**

**axios.delete(url[, config])**

**axios.head(url[, config])**

**axios.post(url[, data[, config]])**

**axios.put(url[, data[, config]])**

**axios.patch(url[, data[, config]])**

**4.1.1并发：**

**axios.all(iterable)**

**axios.spread(callback)**

**4.2config属性选择：**

**baseURL: 'https://some-domain.com/api/'： `baseURL` 将自动加在 `url` 前面，除非 `url` 是一个绝对 URL**

**headers: {'X-Requested-With': 'XMLHttpRequest'}： `headers` 是即将被发送的自定义请求头**

**data: 发送的数据**

**timeout: 1000**

**withCredentials: false, // 默认的，是否带cookit**

**4.3全局配置/实例默认配置：axios.defaults/axios.create.defaults**

**响应数据格式：**

**{ data: {}, status: 200, statusText: 'OK', headers: {},config: {} }**

**4.4拦截器：**

**在请求响应的then，catch之前触发的钩子函数**

**axios.interceptors.request.use(function1(config)请求之前,function2（error)请求失败)**

**axios.interceptors.response.use(function1(config)响应之前,function2（error)响应失败)**

**移除拦截器：axios.interceptors.request/response.eject**