**Axios可以设置很多请求配置 网上索文档**

这些是创建请求时可以用的配置选项。只有 url 是必需的。如果没有指定 method，请求将默认使用 get 方法。

|  |
| --- |
| {  // `url` 是用于请求的服务器 URL  url: '/user',   // `method` 是创建请求时使用的方法  method: 'get', // default   // `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  baseURL: 'https://some-domain.com/api/',   // `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  // 后面数组中的函数必须返回一个字符串，或 ArrayBuffer，或 Stream  transformRequest: [function (data, headers) {  // 对 data 进行任意转换处理  return data;  }],   // `transformResponse` 在传递给 then/catch 前，允许修改响应数据  transformResponse: [function (data) {  // 对 data 进行任意转换处理  return data;  }],   // `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(plain object)或 URLSearchParams 对象  params: {  ID: 12345  },   // `paramsSerializer` 是一个负责 `params` 序列化的函数  // (e.g. https://www.npmjs.com/package/qs, http://api.jquery.com/jquery.param/)  paramsSerializer: function(params) {  return Qs.stringify(params, {arrayFormat: 'brackets'})  },   // `data` 是作为请求主体被发送的数据  // 只适用于这些请求方法 'PUT', 'POST', 和 'PATCH'  // 在没有设置 `transformRequest` 时，必须是以下类型之一：  // - string, plain object, ArrayBuffer, ArrayBufferView, URLSearchParams  // - 浏览器专属：FormData, File, Blob  // - Node 专属： Stream  data: {  firstName: 'Fred'  },   // `timeout` 指定请求超时的毫秒数(0 表示无超时时间)  // 如果请求话费了超过 `timeout` 的时间，请求将被中断  timeout: 1000,   // `withCredentials` 表示跨域请求时是否需要使用凭证  withCredentials: false, // default   // `adapter` 允许自定义处理请求，以使测试更轻松  // 返回一个 promise 并应用一个有效的响应 (查阅 [response docs](#response-api)).  adapter: function (config) {  /\* ... \*/  },   // `auth` 表示应该使用 HTTP 基础验证，并提供凭据  // 这将设置一个 `Authorization` 头，覆写掉现有的任意使用 `headers` 设置的自定义 `Authorization`头  auth: {  username: 'janedoe',  password: 's00pers3cret'  },   // `responseType` 表示服务器响应的数据类型，可以是 'arraybuffer', 'blob', 'document', 'json', 'text', 'stream'  responseType: 'json', // default   // `responseEncoding` indicates encoding to use for decoding responses  // Note: Ignored for `responseType` of 'stream' or client-side requests  responseEncoding: 'utf8', // default   // `xsrfCookieName` 是用作 xsrf token 的值的cookie的名称  xsrfCookieName: 'XSRF-TOKEN', // default   // `xsrfHeaderName` is the name of the http header that carries the xsrf token value  xsrfHeaderName: 'X-XSRF-TOKEN', // default   // `onUploadProgress` 允许为上传处理进度事件  onUploadProgress: function (progressEvent) {  // Do whatever you want with the native progress event  },   // `onDownloadProgress` 允许为下载处理进度事件  onDownloadProgress: function (progressEvent) {  // 对原生进度事件的处理  },   // `maxContentLength` 定义允许的响应内容的最大尺寸  maxContentLength: 2000,   // `validateStatus` 定义对于给定的HTTP 响应状态码是 resolve 或 reject promise 。如果 `validateStatus` 返回 `true` (或者设置为 `null` 或 `undefined`)，promise 将被 resolve; 否则，promise 将被 rejecte  validateStatus: function (status) {  return status >= 200 && status < 300; // default  },   // `maxRedirects` 定义在 node.js 中 follow 的最大重定向数目  // 如果设置为0，将不会 follow 任何重定向  maxRedirects: 5, // default   // `socketPath` defines a UNIX Socket to be used in node.js.  // e.g. '/var/run/docker.sock' to send requests to the docker daemon.  // Only either `socketPath` or `proxy` can be specified.  // If both are specified, `socketPath` is used.  socketPath: null, // default   // `httpAgent` 和 `httpsAgent` 分别在 node.js 中用于定义在执行 http 和 https 时使用的自定义代理。允许像这样配置选项：  // `keepAlive` 默认没有启用  httpAgent: new http.Agent({ keepAlive: true }),  httpsAgent: new https.Agent({ keepAlive: true }),   // 'proxy' 定义代理服务器的主机名称和端口  // `auth` 表示 HTTP 基础验证应当用于连接代理，并提供凭据  // 这将会设置一个 `Proxy-Authorization` 头，覆写掉已有的通过使用 `header` 设置的自定义 `Proxy-Authorization` 头。  proxy: {  host: '127.0.0.1',  port: 9000,  auth: {  username: 'mikeymike',  password: 'rapunz3l'  }  },   // `cancelToken` 指定用于取消请求的 cancel token  // （查看后面的 Cancellation 这节了解更多）  cancelToken: new CancelToken(function (cancel) {  }) } |

**响应结构**

某个请求的响应包含以下信息

|  |
| --- |
| {  // `data` 由服务器提供的响应  data: {},   // `status` 来自服务器响应的 HTTP 状态码  status: 200,   // `statusText` 来自服务器响应的 HTTP 状态信息  statusText: 'OK',   // `headers` 服务器响应的头  headers: {},   // `config` 是为请求提供的配置信息  config: {},  // 'request'  // `request` is the request that generated this response  // It is the last ClientRequest instance in node.js (in redirects)  // and an XMLHttpRequest instance the browser  request: {} } |

使用 then 时，你将接收下面这样的响应 :

|  |
| --- |
| axios.get('/user/12345')  .then(function(response) {  console.log(response.data);  console.log(response.status) |

**等等 。。。。。。。。。。。。。。**