```
/**
 * 1、返回的时promise
 * 2、可以请求并发
 * 3、入参：
 * {
 *      method：请求方式
 *      baseURL：基础路由
 *      url：路由
 *      params：接在路由上的参数【get、head、option、delete请求方式时】
 *      data：在发送体里的参数【post、put、patch请求方式时】
 *      header：额外发送的请求头
 *      dataType：数据接受的格式
 *      timeout：请求超时时间
 *      withCredentials：发送请求时是否携带cookie
 *      transformRequest：允许在请求数据发送到服务器之前对其进行更改【只适用于请求方法‘PUT’，‘POST’和‘PATCH’】
 *                        数组中的最后一个函数必须返回一个字符串，一个 ArrayBuffer或一个 Stream
 *                          transformRequest：[function (data) {
 *                              做任何你想要的数据转换 然后  return data;
 *                          }],
 *      transformResponse：允许在 then / catch之前对响应数据进行更改
 *                          transformResponse：[function (data) {
 *                              做任何你想要的数据转换 然后  return data;
 *                          }],
 * }
 * 4、API：
 *      axios.get(url, configs)
 *      axios.post(url, data, configs)
 *      axios({
 *          method：'get'
 *          ……
 *      })
 *      axios.all()：并发请求
 *      axios.create：创建一个axios实例
 *      axios.interceptors.request.use(sucessCB, errCB)
 *      axios.interceptors.response.use(sucessCB, errCB)
 */
class InterceptorManage {
    constructor() {
        this.callbackList = []
    }

    use = (onFullfiled, onRejected) => {
        this.callbackList.push({onFullfiled, onRejected})
    }
}

class Axios {
    constructor() {
        this.interceptors = {
            request: new InterceptorManage(),
            response: new InterceptorManage()
        }
    }

    // 将{a:1, b:2}形式转换成a=1&b=2格式
    _transData(params) {
        if (params === null) return null;
        return Object.keys(params).map(key => (key + '=' + encodeURIComponent(params[key]))).join('&')
    }

    // 制造axios(configs)形式的请求方法
    request = (configs) => {
        // 拦截器和请求的执行队列
        let chain = [this._sendAjax.bind(this), undefined]
        // 加入请求拦截器：该拦截器会对请求前的config数据做处理
        this.interceptors.request.callbackList.forEach(item => {
            chain.unshift(item.onFullfiled, item.onRejected)
        })
        // 加入响应拦截器：该拦截器会对请求后的response做处理
        this.interceptors.response.callbackList.forEach(item => {
            chain.push(item.onFullfiled, item.onRejected)
        })

        let promise = Promise.resolve(configs)
        while(chain.length) {
            promise = promise.then(chain.shift(), chain.shift())
        }
        return promise
    }

    // 发送ajax请求的函数
    _sendAjax = (configs) => {
        return new Promise((resolve, reject) => {
            let {method='get', baseURL='', url='', params={}, data=null, header={}, timeout=null } = configs

            // 如果有超时设定就设置定时器
            let timer = null
            if (timeout) {
                timer = setTimeout(() => {
                    xhr.abort()
                    clearTimeout(timer)
                }, timeout)
            }

            let xhr = null
            try {
                xhr = new XMLHttpRequest()
            } catch(e) {
                xhr = new ActiveXObject('Micorsoft.XMLHttp')
            }
            if (/^(get|head|option|delete)$/i.test(method)) {  // 需要将params接在url后面
                xhr.open(method, `${baseURL}/${url}?${this._transData(params)}${new Date().getTime()}`, true)
            } else {    // 将data以请求体的形式发送
                xhr.open(method, `${baseURL}/${url}`, true)
            }

            xhr.onreadystatechange = function() {
                if (xhr.readyState === 4) {
                    try {
                        if (xhr.status >= 200 && xhr.status < 300 || xhr.status === 304) {
                            timer && clearTimeout(timer)
                            resolve(xhr)
                        }
                    } catch (e) {
                        reject(e)
                    }
                }
            }

            // 设置请求头
            Object.keys(header).length && Object.keys(header).forEach(key => {
                xhr.setRequestHeader(key, header[key])
            })

            if (/^(post|put)$/i.test(method)) {  // 需要发送Content-type：application/x-www-form-urlencoded
                xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded')
            }

            // 此处的data初始值是null,如果是get类的请求，可以直接send(null)
            xhr.send(this._transData(data))
        })
    }
}

// axios.get(url, config) axios.post(url, data, config)
['get', 'head', 'option', 'delete', 'post', 'put'].forEach(method => {
    /**
     * 这些get/head/option/delete方法是挂载在Axios原型上的，可以被axios实例调用
     * 注意!!!这里不可以用箭头函数，因为在函数中需要this(axios实例)去获取request函数
     */
    if (/^(get|head|option|delete)$/i.test(method)) {
        Axios.prototype[method] = function(url, args={}){
            return this.request({
                method,
                url,
                ...args
            })
        }
    } else {
        Axios.prototype[method] = function(url, data, args){
            return this.request({
                method,
                url,
                data,
                ...args
            })
        }
    }

})

let utils = {
    copyAttr: (source, target, context) => {
        Object.keys(source).forEach(attr => {
            if (typeof source[attr] === 'function') {

                target[attr] = source[attr].bind(context)
            } else {
                target[attr] = source[attr]
            }
        })
```

```
    }
}

function createAxios() {
    let axios = new Axios()
    let request = axios.request.bind(axios) // 这里bind(axios)是为了返回一个函数
    // 将axios实例能调用get/post等方法挂载在request方法上
     /**
     * 这里用传context是因为，虽然将Axios.prototype上的get/post属性挂在了request函数上，
     * 但是!!!执行这些get,post的还是axios实例，
     * 如果不绑定this值，执行get/post函数时：就是axios.request.get(url,config),
     * 而get函数中用到了this.request,在axios.request函数中是找不到request函数的【细想一下！】
     */
    utils.copyAttr(Axios.prototype, request, axios)
    // 将绑在axios实例上的拦截器复制到axios.request函数上
    utils.copyAttr(axios, request)
    return request
}

// 暴露出去的实际上是axios实例的request方法
let axios = createAxios()




// 使用
axios({
    method: 'get',
    baseURL: 'http://localhost:1124',
    url: 'getInfo',
    params: {name: 'lidan', age: 27},
    // 由于设置简单请求以外的请求头，所以此次请求会先发送一次预检请求
    header: {'x-csrf-token': 'lllddd aaannn'},
}).then(res => {
    console.log(res, '====axios')
})

axios.interceptors.request.use(configs => {
    console.log('请求拦截')
    return { ...configs, params: {}}
}, err => {
    console.log(err)
})

axios.interceptors.response.use(response => {
    console.log('响应拦截')
    return response
}, err => {
    console.log(err)
})

// ----------或者--------------
axios.get('getInfo', {
    baseURL: 'http://localhost:1124',
    params: {name: 'lidan', age: 27},
    // 由于设置简单请求以外的请求头，所以此次请求会先发送一次预检请求
    header: {'x-csrf-token': 'lllddd aaannn'},
}).then(res => {})

axios.post('getInfoPost', {name: 'lidan', age: 27}, {
    baseURL: 'http://localhost:1124',
    // 由于设置简单请求以外的请求头，所以此次请求会先发送一次预检请求
    header: {'x-csrf-token': 'lllddd aaannn'},
}).then(res => {})
```