axios: 基于promise的HTTP第三方库, 可用于客户端和服务端node.js

axios特性

1.支持promiseAPI

2.拦截请求和响应（可以在请求前，响应前做一些操作，如在请求头中加入信息）

3.转换请求数据和响应数据（在请求时，对敏感信息加密；响应时解密）

4.取消请求

5.自动转换JSON数据（一般在http请求时，传输的为字符串）

6.客户端支持防御XSRF

axios浏览器支持

火狐，Opera，谷歌，IE(8以上)

axios请求方式

axios请求方法：get,post,put,patch,delete

get:获取数据

post:提交数据（表单提交+文件上传）

put:更新数据——将所有数据推送到后端（服务端）——一般用于新建

patch:更新数据——只将修改的数据送到后端（服务端）——一般用于更新

delete:删除数据

参数:

1. post,put,patch-三个参数：url路径，请求数据，config配置

请求数据data格式两种：

1.form-data：表单提交（图片文件上传）

2.applicition/json

2.get,delete-两个参数：url路径，config配置

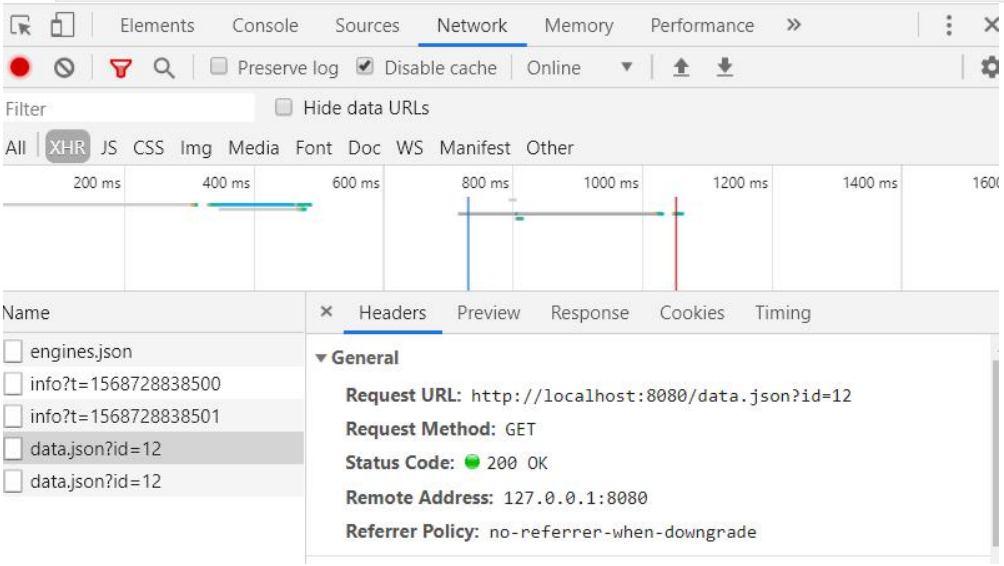
axios使用

1.get（Get.vue）

```
//在created生命周期
  created(){
    // 执行请求
    // 第一种写法
    // 传参请求—http://localhost:8080/data.json?id
    axios.get('/data.json',{
      params:{id:12}}).then((res)=>{console.log(res)})
    // 另一种写法
    // 传参
     axios({
```

```
        method:'get',
        url:'/data.json',
        params:{id:12}
    }).then((res)=>{console.log(res)})
}
```

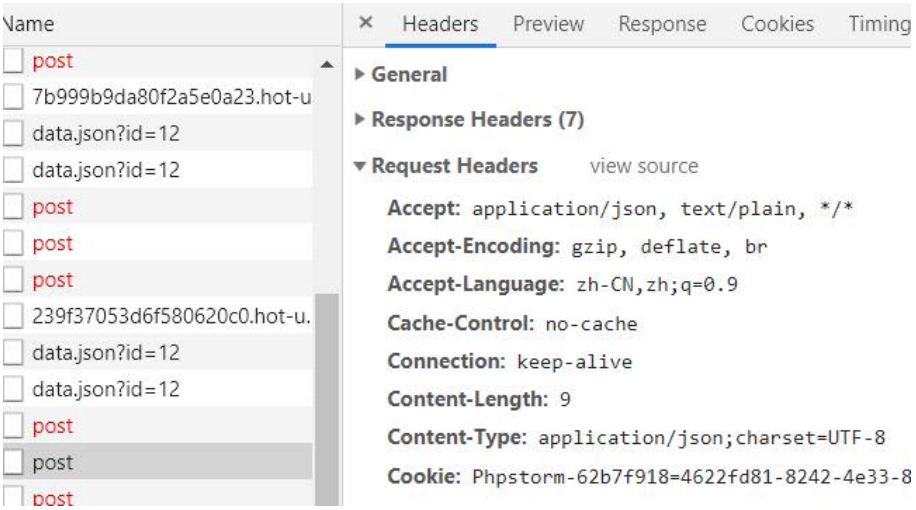
访问：http://localhost:8080/get



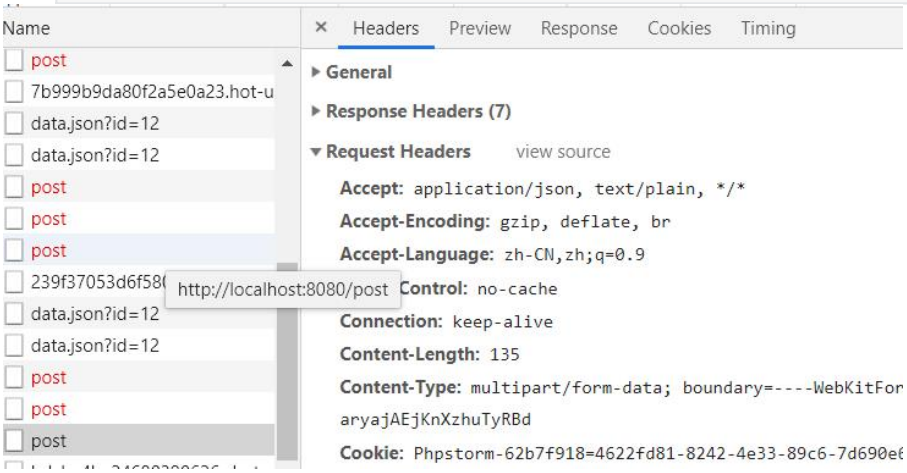
2.post请求

```
post-三个参数：url路径，{}请求数据，config配置
请求数据data格式两种：
 1.form-data：表单提交（图片文件上传）
 2.applicition/json
```

```
//applicition/json
let data = {id:12}
    axios.post('/post',data).then(res=>{
      console.log(res)
    })
    // 另一种写法
    // 传参
    axios({
        method:'post',
        url:'/post',
        data:data
    }).then((res)=>{
        console.log(res)
    })
```




```
// form-data请求
    // 创建一个FormData格式的对象
    let formData = new FormData()
    let data = {id:12}
    // 对data遍历（key对应id）
    for(let key in data){
      // 将键值添加到formData中
      formData.append(key,data[key])
    }
     axios.post('/post',formData).then(res=>{
       console.log(res)
     })
```



3.put请求

```
// put请求
    axios.put('/put',data).then(res=>{
      console.log(res)
    })
```

4.patch请求

```
// patch请求
    axios.patch('/patch',data).then(res=>{
      console.log(res)
    })
```

5.delete请求

```
// delete请求-两个参数

    // 若接口需要在url后拼接参数
    axios.delete('/delete',{
      params:{
        id:12
      }
    }).then(res=>{
      console.log(res)
     })

     // 不在url传输
      axios.delete('/delete',{
       data:{
         id:12
       }
     }).then(res=>{
       console.log(res)
     })

     // 不用别名
     axios({
       method:'delete',
       url:'/delete',
       params:{},
       data:{}
     }).then(res=>{
       console.log(res)
     })

```

并发请求：同时进行多个请求，并统一处理返回值

```
两个方法：
axios.all()-参数：数组，数组内的值是一个一个axios请求
axios.spread()-作用是:在axios.all()多个请求完成的时候，将返回数据进行分割处理
然后对每一个返回值统一进行处理
```

```
//在created生命周期
  created(){
    axios.all(
      [
        axios.get('/data.json'),
        axios.get('/city.json'),
      ]
      // then()内的参数为axios.spread()
    ).then(
       // axios.spread()内的参数为回调函数
       // axios.all()有几个方法，就需要有几个对应的返回值
       axios.spread((dataRes,cityRes)=>{
         console.log(dataRes,cityRes)
       })
     )
   }
```

创建axios实例

```
//在created生命周期
  created(){
    // 当项目中只有一种后端语言时,可直接:
    axios.get('/data.json')

    // 当后端接口地址有多个，且超时时长不一样
    // 创建实例
    // axios.create(axios配置信息)方法
    let axios1 = axios.create({
       // 域名
       baseURL:'http://localhost:8080',
       // 超时时长（默认是1000毫秒），超时报401
       timeout:1000
     })
     let axios2 = axios.create({
```

```
      // 域名
      baseURL:'http://localhost:9090',
      // 超时时长（默认是1000毫秒），超时报401
      timeout:5000
    })

    // 使用实例
    axios1.get('/data.json').then(res=>{
      console.log(res)
    })
    axios2.get('/city.json').then(res=>{
      console.log(res)
    })
  }
```

axios配置参数

```
//在created生命周期
  created(){
    axios.create({
      // 请求域名，基本地址
      baseURL:'http://localhost:8080',
      // 请求超时时长（毫秒）
      timeout:1000,
      // 请求路径url
      url:'/data.json',
       // 请求方法
       method:'get,post,patch,put,delete',
       // 设置请求头
       headers:{},
       // 比如登录后一些接口需要限权
       //  headers:{
       //    //  识别当前登陆人信息
       //    token:''
       // }
       //请求参数拼接在url上
       params:{},
       // 请求参数放在请求体中
       data:{}
    })
```

```
    // get()后为相对路径,请求时会和baseURL拼接——http://localhost:8080/data.json
    axios.get('/data.json',{config:上述的配置参数})
  }
```

axios配置方法

```
// 1.axios全局配置(常修改的全局配置timeout和baseURL)
      axios.defaults.timeout = 1000
      axios.defaults.baseURL = 'http://localhost:8080'
   // 2.axios实例配置
      // 若不添加参数，则默认全局配置
      let instance = axios.create()
      // 修改配置
      instance.defaults.timeout = 3000
   // 3.axios请求配置
       instance.get('/data.json',{
         timeout:3000
       })
    // 4.优先级-高--->低
    // 请求配置>实例配置>全局配置
```

axios配置具体使用

```
//在created生命周期内
// 实际开发中，全局配置很少用到
    // 如:有两种接口
    // http://localhost:8080
    // http://localhost:9090

    // 创建实例
    let instance1 = axios.create({
      baseURL:'http://localhost:8080',
       timeout:1000
     })
     let instance2 = axios.create({
       baseURL:'http://localhost:9090',
       timeout:3000
    })
    // 使用实例请求
    // instance1用到的参数：baseURL，timeout,url,method,params
```

```
    instance1.get('/data.json',{
        params:{}
    }).then(res=>{
      console.log(res)
    })
    // instance2用到的参数：baseURL，timeout：5000,url,method
     instance2.get('/city.json',{
      //当实例的配置不符合时
        timeout:5000,
    }).then(res=>{
      console.log(res)
    })

```

axios拦截器（请求拦截器+响应拦截器）

```
  //在created生命周期
  created(){
    // 拦截器：在请求或响应被处理前拦截它们
    //（在发起请求前做些处理，在响应回来后做些处理）
    // 请求拦截器+响应拦截器
//(use()方法有两个参数：1，请求前的回调；2.请求错误的回调)

    // 请求拦截器
    axios.interceptors.requeset.use
     (config=>{
       // 在发送请求前做些什莫,处理后需要return出去
       return config
     },err=>{
       // 在请求错误时做些什么
       return Promise.reject(err)
     })

    // 响应拦截器
     axios.interceptors.response.use
     (res=>{
       // 请求成功，对响应数据做些处理，处理后需要return出去
       return res

     },err=>{
       // 响应错误做些什莫
```

```
        return Promise.reject(err)
    })
    // 请求成功的数据，会到res;请求失败的数据，会到err
    axios.get('').then(res=>{
 //成功
    }).catch(err=>{
 //失败
    })
}
```

```
请求错误和响应错误的区别：
请求错误：发送请求，但没有到达后端，报错404
响应错误：发送请求，有到达后端，返回的错误
```

拦截器使用（举例）：登录状态

```
// 举例：登录状态（登录，后端会返回一个参数（一个编码的字符串）token:''）
    // token内包含了加密的个人信息，通过加密信息可以识别身份
    // 一般token，放在请求头headers内

    // 使用请求拦截器

    // 访问-需要登录的接口
    //声明实例(在开发中，给实例添加拦截器和属性，不会给axios设置内容。若设置则可能会造成全局污染)
    let instance = axios.creste({})
     instance.interceptors.request.use
     (config=>{
       // 给token赋值
       config.headers.token = ''

       // return 回去
       return config
     })
     // 若访问-不需要登录的接口
     let instance2 = axios.creste({})
```

拦截器使用—移动端开发-弹窗（使用请求拦截器+响应拦截器）

```
    // 移动端开发-使用请求拦截器
    let instance3 = axios.creste({})
    instance3.interceptors.request.use
```

```
    (config=>{
      // 有一弹窗，在请求前将弹窗弹出
      $('#model').show()
      // return
      return config
    })
     // 请求后，将弹窗隐藏
     // 使用响应拦截器
     instance3.interceptors.response.use
     (res=>{
       // 弹窗隐藏
       $('#model').hide()
       return res
     }
```

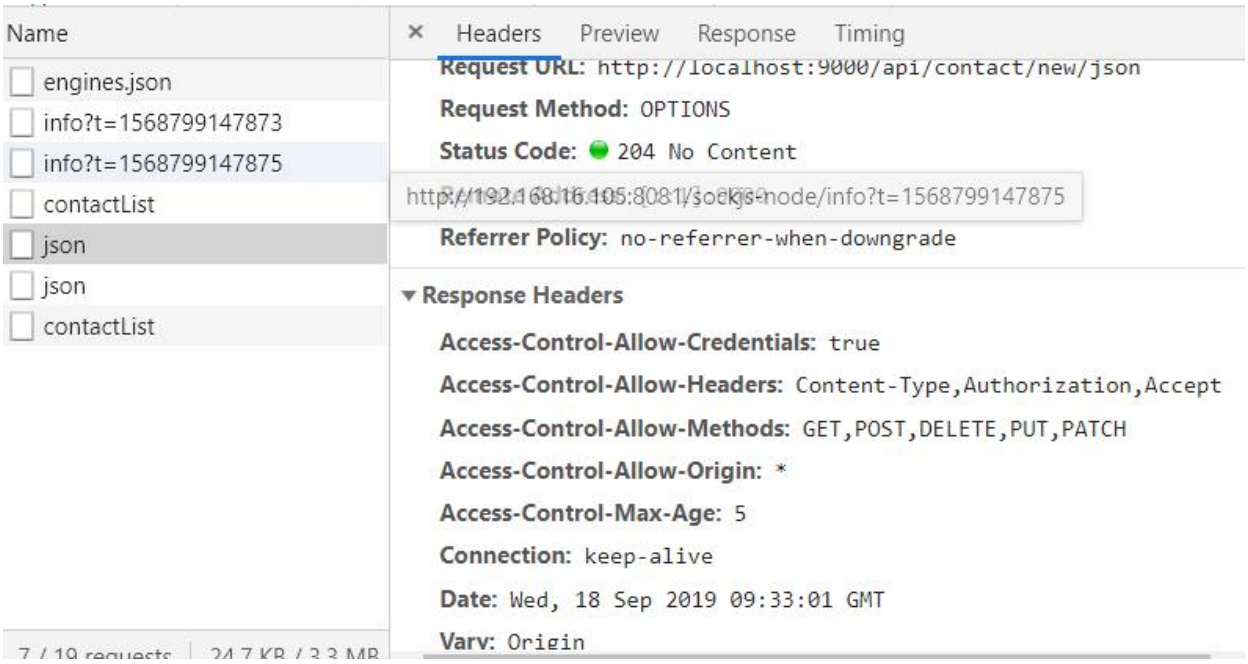
axios错误处理

```
//在created生命周期
  created(){

    // 错误处理:请求错误时进行的处理
    // 请求错误和响应错误一般都会汇聚到catch中

    // 执行请求
    axios.get('/data.json').then((res)=>{
      // 成功
       console.log(res)
     }).catch(err=>{
       // 失败
     })

     // 举例：实际开发中，一般添加统一的错误处理
     // 创建实例
     let instance = axios.create({})
     // 请求拦截器
     instance.interceptors.request.use
     (config=>{
         console.log(config)
     },err=>{
       // 请求错误-一般http状态码以4开头，常见：401超时；404not found
       // 添加2s弹窗提示
```

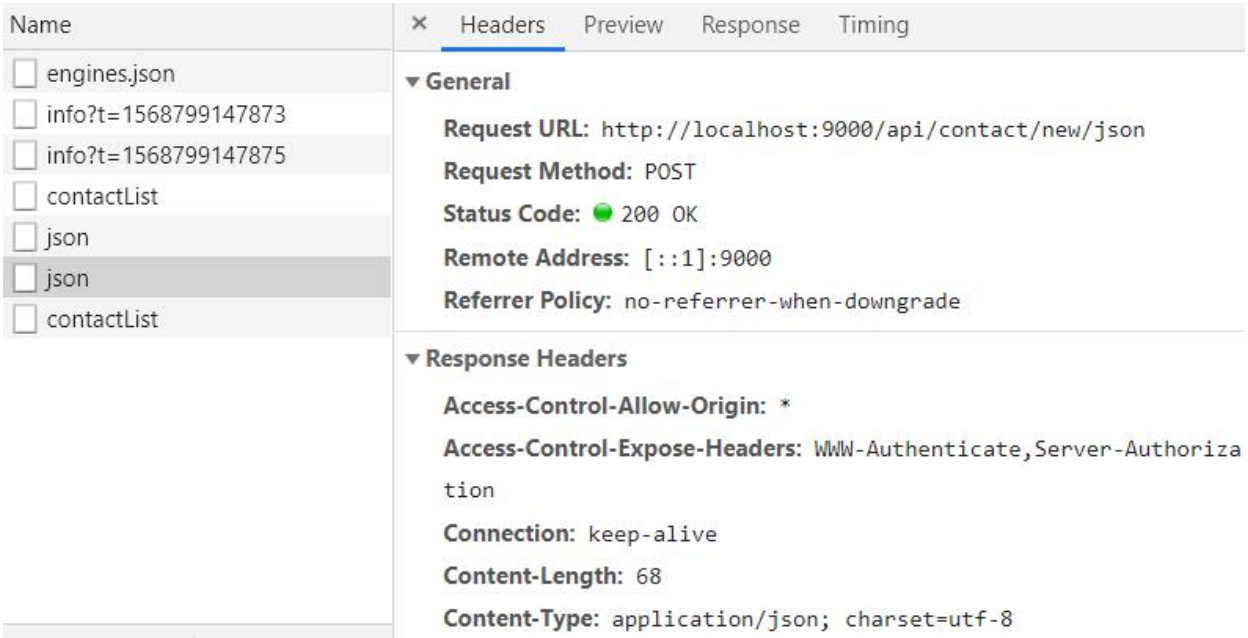
```
      $('#model').show()
      setTimeout(()=>{
        $('#model').hide()
      },2000)
      return Promise.reject(err)
    })
    // 响应拦截器
    instance.interceptors.response.use
    (res=>{
      console.log(res)
    },err=>{
      // 添加2s弹窗提示
      $('#model').show()
      setTimeout(()=>{
        $('#model').hide()
      },2000)
      // 响应错误处理-一般http状态码以5开头，500-系统错误；502-系统重启
      return Promise.reject(err)
    })

    // 请求
    instance.get('data.json').then(res=>{
      //成功
      console.log(res)
    }).catch(err=>{
      // 错误处理
      console.log(err)
    })
  }
```

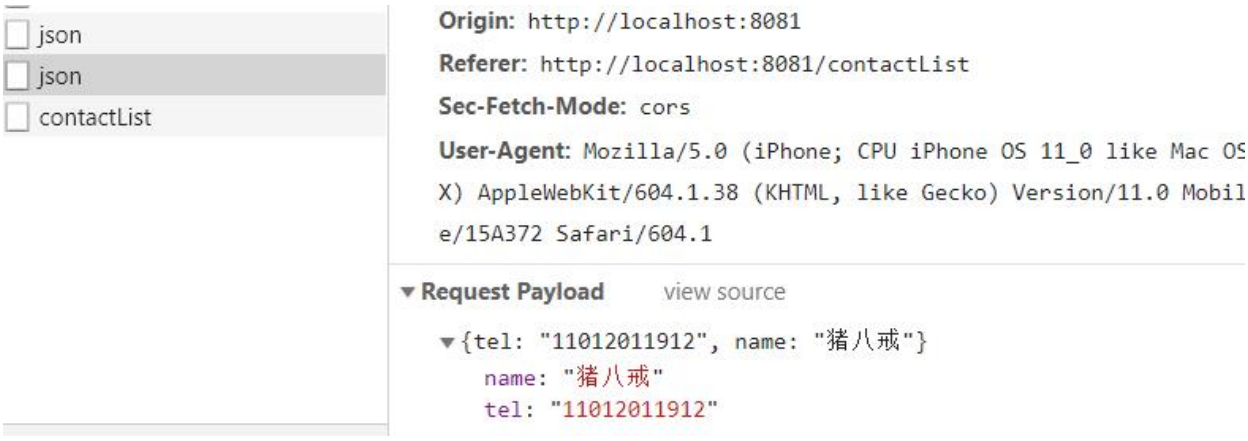
两个JSON+contactList:

第一个json:称为“不确定通信”，判断接口是否能请求到后端服务；通，则继而请求第二个json

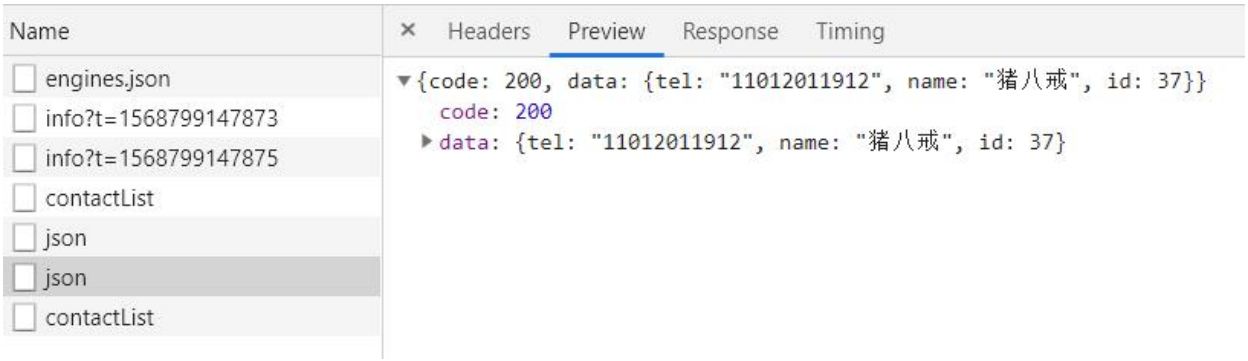


第二个json：真正的请求





返回值：



第三个contactList：联系人列表