axios 基础知识点总结

介绍

ajax 是封装 XMLHttpRequests
axios 是基于ES6 Promise,也是从浏览器创建XMLHttpRequest，可以用在浏览器和nodejs中

特征

从浏览器创建XMLHttpRequest　//get post put patch
从node创建http请求
拦截请求和响应　　//axios.interceptors.request　axios.interceptors.response
取消请求　　// cancelToken
自动转换json数据
转换请求数据和响应数据　　//transformRequest transformResponse
客户端支持防御XSRF

get请求两种方式

axios.get(url[,config]) 参数 params:{}
axios.get('/user?id=12345')===>axios.get('/user',params{id:12345})

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

post

axios.post(url[,data,config]) data是带的参数，可以是字符串或者对象
axios.post('128,3,4,5','name="zhangsan"')
axios.post('128,3,4,5',{name:"zhangsan"})

并发请求

axios.all(iterable)

axios.spread(callback)

axios.all(iterable)回调函数接收的参数axios.spread(callback)

```
axios.all([req1,req2]).then(axios.spread(res1,res2)=>{
//全部成功
}).catch(err=>{
//有一个报错
})
```

配置信息

可以通过传递相关配置来创建请求
axios([config])

```
{
    url:'',
    method:'',
    baseURL:'',
    transformRquest:'',
    //在传递给then/catch前，允许修改响应数据
    transformResponse:[function(data){}],
    headers:[],
    params:{},
    paramsSerializer:function(params){},
    data:{},
    timeout:'',//请求超时
    withCtrdentials:false, //跨域cookie的使用，如果需要cookie就涉及这个，建议使用token
    adapter:function(config){},
    auth:{},
    responseType:'json',
    responseEncoding:'utf8',
    //是用作xsrf token的值的cookie的名称
    xsrfCookieName:'XSRF-TOKNE',
    //是承载 xsrf token的值的HTTP头名称
    xsrfHeaderName:'XSRF-TOKEN',
    onUploadProgress: function(progressEvent){},
    onDownloadProgress: function(progressEvent){},
   cancelToken: new CancelToken(function (cancel) {
     }),    validateStatus:function(status){}
}
```

withCtrdentials

表示跨域请求时是否需要使用凭证，就是是否允许浏览器携带cookie信息到服务器，默认时false，如果设置为true就允许携带cookie

onUploadProgress与onDownloadProgress

允许为上传处理进度事件

```
onUploadProgress:function(progressEvent){
//对原生进度事件的处理
progressEvent.loaded/progressEvent.total*100//上传的百分比
}
```

允许为下载处理进度事件

```
onDownloadProgress:function(progressEvent){}
文件的传输就是流，formdata处理
upload(e){
var file=e.target.files[0];
var fd=new FormData();
fd.append('file',file);
axios.post('/upload',file,{
onprogressUpload:function(progressEvent){
var progress=progressEvent.loaded/progressEvent.total*100
}
}).then(res=>{}).cathch(err=>{})
}
```

一般这样的进度都是结合elementui iview组件结合使用的

demo:

```
download(){
axios.post('/download',{
onprogressDownload:function(progressEvent){
var progress=progressEvent.loaded/progressEvent.total*100
}
}).then(res=>{}).cathch(err=>{})
}
```

xsrfCookieName xsrfHeaderName

跨站请求伪造，防注射，后端响应的token值，是为了cookie安全

cancelToken

'cancelToken'指定用于取消请求的cancel token

```
cancelToken:new CancelToken(function(cancel){})
```

使用方法

使用取消令牌取消请求====》使用CancelToken.source工厂方法创建取消令牌

```
var CancelToken=axios.CancelToken;
var source=CancelToken.source();
axios.get('/user',{
cancelToken:source.token//携带取消标识
}).then(res=>{
})catch(err=>{
if(axios.isCancel(err)){
console.log(err.message);
```

```
}else{
//处理错误
}
});
//message参数是可选的
source.cancel('cancel message');
```

可以通过一个executor函数到 CancelToken的构造函数来创建取消令牌：

```
var CancelToken=axios.CancelToken;
var cancel;
axios.get('/user',{
cancelToken:new CancelToken(function executor(c){
cancel=c;

})
}).then(res=>{
}).catch(err=>{
});
//取消请求
cancel();
```

tips:可以使用一个cancelToken取消多个请求

断点续传

在文件上传的过程中实现断点续传，
在onUploadProgress中获取已经上传的文件的loaded信息并保存，
取消请求，点击继续上传，分割文件已上传的部分

```
var fileData=this.file.slice(loaded,file.size);//获取文件未上传的流，
var formData=new FormDate();
formData.append('file'fileData);
```

transformRequest与transformResponse

transformResponse

```
axios.get({
url:'/user',
params:{id:123},
transfromResponse:[function(data){
//对data进行任意转换处理,默认是字符串，需要用JSON.parse(data)进行转换
return data
}]
}).then(res=>{

}).catch(err=>{

})
```

允许在向服务器发送前，修改请求数据，只能用'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

```
axios.post(
{userId:123},
{transformRequest:[function(data){
//对data进行任意转换处理
return data
}]}
).then(res=>{}).catch(err=>{});
或者：
axios.post('userid=123',
{transformRequest:[function(data){
//对data进行任意转换处理
return data
}]}
).then(res=>{}).catch(err=>{});
```

请求方法的别名

```
request(配置)
get(url[,config])
delete(url[,config])
head(url[,config])
post(url[,data[,config]])
put(url[,data[,config]])
patch(url[,data[,config]])
```

默认配置

```
axios.default.baseURL
axios.default.headers.common['header-name']
axios.default.headers.post['Content-Type']
```

创建实例

```
var instance=axios.create({
});
```

拦截器

在请求或响应被then或catch处理前拦截它们。
use:中间件 任何请求都会通过这里

```
axios.interceptors.request.use(function(config){
//config类似axios的defaults可以在这里配置axios的各种配置信息
return config;
},function(err){
return Promise.reject(error);
})
axios.interceptors.response.use(function(response){
return response;
},fnnction(error){
return Promise.reject(error);
});
```

实现一个类似cookie的机制
//客户端发送请求。服务器响应，并把cookie放在header中发送到客户端，
客户端再发送请求就带着cookie到服务器

思路：

第一次发送请求，获取服务器端返回的token,存放在localstorage中

在axios.interceptors.request.use中判断localstorage中是否存在token，存在就放在请求实例的headers中