基本使用

- axios是一个函数对象
- axios调用的返回值是Promise实例
- 成功的值叫response，失败的值叫error
- axios成功的值是一个axios封装的response对象，服务器返回的真正数据在response.data中
- 携带query参数时，编写的配置项叫做params
- 携带params参数时，就需要自己手动拼在url中

案例

- ## 获取所有人---发送GET请求---不携带参数

```
btn1.onclick = ()=>{
    //完整版
    axios({
        url:'http://localhost:5000/persons', //请求地址
        method:'GET',//请求方式
    }).then(
        response => {console.log('请求成功了',response.data);},
        error => {console.log('请求失败了',error);}
    )

    //精简版
    axios.get('http://localhost:5000/persons').then(
        response => {console.log('请求成功了',response.data);},
        error => {console.log('请求失败了',error);}
    )
}
```

- 获取所某个人---发送GET请求---携带query参数

```
btn2.onclick = ()=>{
    //完整版
    axios({
        url:'http://localhost:5000/person',
        method:'GET',
        params:{id:personId.value} //此处写的是params，但携带的是query参数，要携带params参数，直接在url后面拼/
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )

    //精简版
    axios.get('http://localhost:5000/person',{params:{id:personId.value}}).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
```

```
    )
}
```

- 添加一个人---发送POST请求---携带json编码参数 或 urlencoded编码

```
btn3.onclick = ()=>{
    //完整版
    axios({
        url:'http://localhost:5000/person',
        method:'POST',
        data:{name:personName.value,age:personAge.value}//携带请求体参数（json编码）
        //data:`name=${personName.value}&age=${personAge.value}`//携带请求体参数（urlencoded编
码)
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )

    //精简版
    axios.post('http://localhost:5000/person'
               {name:personName.value,age:personAge.value}).then(   // (json编码)
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
    axios.post('http://localhost:5000/person',
               `name=${personName.value}&age=${personAge.value}`).then(// (urlencoded编码)
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
}
```

- 更新一个人---发送PUT请求---携带json编码参数 或 urlencoded编码

```
btn4.onclick = ()=>{
    //完整版
    axios({
        url:'http://localhost:5000/person',
        method:'PUT',
        data:{
            id:personUpdateId.value,
            name:personUpdateName.value,
            age:personUpdateAge.value
        }
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )

    //精简版
    axios.put('http://localhost:5000/person',{
        id:personUpdateId.value,
        name:personUpdateName.value,
```

```
        age:personUpdateAge.value
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
}
```

- 删除一个人---发送DELETE请求---携带params参数

```
btn5.onclick = ()=>{
    axios({
        url:`http://localhost:5000/person/${personDeleteId.value}`,
        method:'DELETE',
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
}
```

常用配置项

- url:'/persons', ——请求地址
- method:'GET',——请求方式
- params:{delay:3000},——配置query参数
- data:{c:3,d:3},——配置请求体参数(json编码)
- data:'e=5&f=6',——配置请求体参数(urlencoded编码)
- timeout:2000,——配置超时时间
- headers:{school:'atguigu'} ——配置请求头
- responseType:'json'——配置响应数据的格式(默认值)

```
//给axios配置默认属性
axios.defaults.timeout = 2000
axios.defaults.headers = {school:'atguigu'}
axios.defaults.baseURL = 'http://localhost:5000'

btn.onclick = ()=>{
    axios({
        url:'/persons', //请求地址
        method:'GET',//请求方式
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
}

btn2.onclick = ()=>{
    axios({
        url:'/test1', //请求地址
        method:'GET',//请求方式
    }).then(
        response => {console.log('成功了',response.data);},
```

```
        error => {console.log('失败了',error);}
    )
}
```

create

- 根据指定配置创建一个新的axios, 也就是每个新axios都有自己的配置
- 新axios只是没有取消请求和批量发请求的方法, 其它所有语法都是一致的
- 为什么要设计这个语法?
    - 需求：项目中有部分接口需要的配置与另一部分接口需要的配置不太一样

```
const btn = document.getElementById('btn')
const btn2 = document.getElementById('btn2')
const btn3 = document.getElementById('btn3')

const axios2 = axios.create({
    timeout:3000,
    //headers:{name:'tom'},
    baseURL:'https://api.apiopen.top'
})

//给axios配置默认属性
axios.defaults.timeout = 2000
axios.defaults.headers = {school:'atguigu'}
axios.defaults.baseURL = 'http://localhost:5000'

btn.onclick = ()=>{
    axios({
        url:'/persons', //请求地址
        method:'GET',//请求方式
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
}

btn2.onclick = ()=>{
    axios({
        url:'/test1', //请求地址
        method:'GET',//请求方式
    }).then(
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
}

btn3.onclick = ()=>{
    axios2({
        url:'/getJoke',
        method:'GET'
    }).then(
```

```
        response => {console.log('成功了',response.data);},
        error => {console.log('失败了',error);}
    )
}
```

拦截器

- axios请求拦截器：
  - 在真正发请求前执行的一个回调函数
  - 作用：对所有的请求做统一的处理：追加请求头、追加参数、界面loading提示等等
- axios响应拦截器
  - 得到响应之后执行的一组回调函数
  - 作用：
    - 若请求成功，对成功的数据进行处理
    - 若请求失败，对失败进行统一的操作

```
//请求拦截器
axios.interceptors.request.use((config)=>{
    console.log('请求拦截器1执行了');
    if(Date.now() % 2 === 0){
        config.headers.token = 'atguigu'
    }
    return config
})

//响应拦截器
axios.interceptors.response.use(
    response => {
        console.log('响应拦截器成功的回调执行了',response);
        if(Date.now() % 2 === 0) return response.data
        else return '时间戳不是偶数，不能给你数据'
    },
    error => {
        console.log('响应拦截器失败的回调执行了');
        alert(error);
        return new Promise(()=>{})
    }
)

btn.onclick = async()=>{
    const result = await axios.get('http://localhost:5000/persons21')
    console.log(result);
}
```

取消请求

```
<button id="btn">点我获取测试数据</button><br/><br/>
<button id="btn2">取消请求</button><br/><br/>
```

```
<script type="text/javascript" >
        const btn = document.getElementById('btn')
        const btn2 = document.getElementById('btn2')
        const {CancelToken} = axios //CancelToken能为一次请求“打标识”
        let cancel

        btn.onclick = async()=>{
                axios({
                        url:'http://localhost:5000/test1?delay=3000',
                        cancelToken:new CancelToken((c)=>{ //c是一个函数，调用c就可以关闭本次请求
                                cancel = c
                        })
                }).then(
                        response => {console.log('成功了',response.data);},
                        error => {console.log('失败了',error);}
                )
        }

        btn2.onclick = ()=>{
                cancel()
        }
</script>
```

- 多次请求取消之前的

```
<button id="btn">点我获取测试数据</button><br/><br/>
<button id="btn2">取消请求</button><br/><br/>
<script type="text/javascript" >
        const btn = document.getElementById('btn')
        const btn2 = document.getElementById('btn2')
        const {CancelToken,isCancel} = axios //CancelToken能为一次请求“打标识”
        let cancel

        btn.onclick = async()=>{
                if(cancel) cancel()     //多次调用 取消之前的（如果cancel有值 调用cancel）
                axios({
                        url:'http://localhost:5000/test1?delay=3000',
                        cancelToken:new CancelToken((c)=>{ //c是一个函数，调用c就可以关闭本次请求
                                cancel = c
                        })
                }).then(
                        response => {console.log('成功了',response.data);},
                        error => {
                                if(isCancel(error)){
                                        //如果进入判断，证明：是用户取消了请求
                                        console.log('用户取消了请求，原因是：',error.message);
                                }else{
                                        console.log('失败了',error);
                                }
                        }
                )
        }
```

```
        btn2.onclick = ()=>{
                cancel('任性，就是不要了')
        }
</script>
```

- 与拦截器配合使用

```
<button id="btn">点我获取测试数据</button><br/><br/>
<button id="btn2">取消请求</button><br/><br/>
<script type="text/javascript" >
        const btn = document.getElementById('btn')
        const btn2 = document.getElementById('btn2')
        const {CancelToken,isCancel} = axios //CancelToken能为一次请求“打标识”
        let cancel

        axios.interceptors.request.use((config)=>{
                if(cancel) cancel('取消了')
                config.cancelToken = new CancelToken((c)=> cancel= c)
                return config
        })

        axios.interceptors.response.use(
                response => {return response.data},
                error => {
                        if(isCancel(error)){
                                //如果进入判断，证明：是用户取消了请求
                                console.log('用户取消了请求，原因是：',error.message);
                        }else{
                                console.log('失败了',error);
                        }
                        return new Promise(()=>{})
                }
        )

        btn.onclick = async()=>{
                const result = await axios.get('http://localhost:5000/test1?delay=3000')
                console.log(result);
        }

        btn2.onclick = ()=>{
                cancel('任性，就是不要了')
        }
</script>
```

批量发送请求

```
btn.onclick = async()=>{
        axios.all([
                axios.get('http://localhost:5000/test1'),
                axios.get('http://localhost:5000/test2?delay=3000'),
                axios.get('http://localhost:5000/test3'),
        ]).then(
                response => {console.log(response);},
                error => {console.log(error);}
        )
}
```