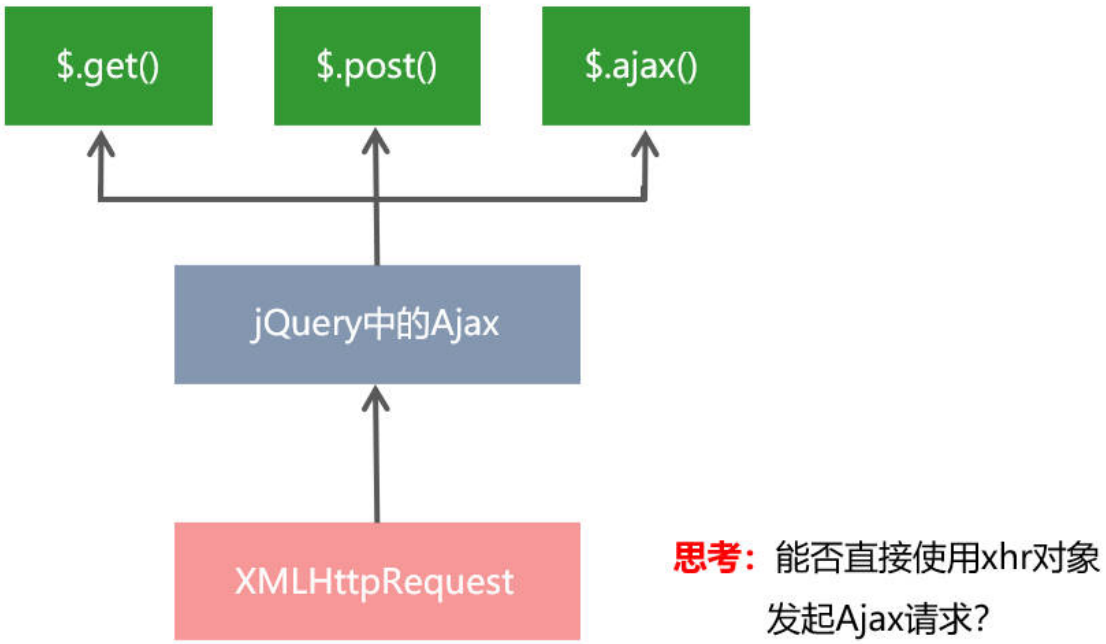
m 1.XMLHttpRequest的基本使用

1.1 什么是 XMLHttpRequest

XMLHttpRequest(简称 xhr) 是浏览器提供的 JavaScript 对象，通过它，可以 请求服务器上的数据资源 。之前所学的jQuery中的 Ajax 函数，就是基于 xhr 对象封装出来的。



1.2使用xhr发起GET请求

```
// 1. 创建 XHR 对象
const xhr = new XMLHttpRequest()
// 2. 调用 open 函数 指定 请求方式 与URl 地址
xhr.open('GET','http://www.liulongbin.top:3006/api/getbooks')
// 3. 调用 send 函数，发起Ajax请求
xhr.send()
// 4. 监听 onreadystatechange 事件
xhr.onreadystatechange = function() {
    // 4.1 监听 xhr 对象的 请求状态 readystate; 与服务器 响应的状态 status
    if(xhr.readystate === 4 && xhr.status === 200)
        // 4.2 打印服务器响应回来的数据
        console.log(xhr.responseText)
}
```

1.3 了解xhr对象的readyState属性

XHRHttpRequest 对象的 readystate 属性，用来表示==当前Ajax请求所处的状态。每个Ajax请求必然处于以下状态中的一个：

值	状态	描述
0	UNSENT	XMLHttpRequest 对象已被创建，但尚未调用 open方法。
1	OPENED	open() 方法已经被调用。
2	HEADERS_RECEIVED	send() 方法已经被调用，响应头也已经被接收。
3	LOADING	数据接收中，此时 response 属性中已经包含部分数据。
4	DONE	Ajax 请求完成，这意味着数据传输已经彻底完成或失败。

1.4 使用xhr发起带参数的GET请求

使用 xhr 对象发起带参数的 GET 请求时，只需在调用 xhr.open() 期间。为URL 地址指定参数即可：

```
// ...省略不必要的代码
xhr.open('GET','http://www.liulongbin.top:3006/api/getbooks?id=1')
// ...省略不必要的代码
```

这种在 URL 地址后面拼接的参数，叫做 `查询字符串` 。

1.5 查询字符串

1. 什么是查询字符串

定义：查询字符串（URL 参数）是指在URL的末尾加上用于向服务器发送信息的字符串(变量)。

格式：将英文的 `？` 放在URL的末尾，然后再加上 `参数=值` ，想加上多个参数的话，使用 `&` 符号进行分隔。以这个形式，可以将想要发给服务器的数据添加到 URL中。

```
// 不带参数的 URL 地址
http://www.liulongbin.top:3006/api/getbooks
// 带一个参数的 URL 地址
http://www.liulongbin.top:3006/api/getbooks?id=1
// 带两个参数的 URL 地址
http://www.liulongbin.top:3006/api/getbooks?id=1 & bookname=西游记
```

2.GET请求携带参数的本质

无论使用$.ajax()，

还是使用$.get()，又或者直接使用xhr对象发起GET请求，当需要携带参数的时候，本质上，是直接将参数以查询字符串的形式，追加到URL地址的后面，发送到服务器的。

```
$.get('url', {name: 'zs', age: 20}, function() {})
// 等价于
$.get('url?name=zs&age=20', function() {})

$.ajax({ method: 'GET', url: 'url', data: {name: 'zs', age: 20}, success: function() {} })
// 等价于
$.ajax({ method: 'GET', url: 'url?name=zs&age=20', success: function() {} })
```

1.6URL编码与解码

1. 什么是URL编码

URL 地址中，只允许出现英文相关的字母、标点符号、数字，因此，在URL地址中，不允许出现中文字符。

如果URL中需要包含中文这样的字符，则必须对中文字符进行编码（转义）。

URL编码的原则：使用安全的字符（没有特殊用途或者特殊意义的可打印字符）去表示那些不安全的字符。

URL编码原则的通俗理解：使用**英文字符**去表示**非英文字符**。

```
http://www.liulongbin.top:3006/api/getbooks?id=1&bookname=西 游 记
// 经过 URL 编码之后，URL地址变成了如下格式：
http://www.liulongbin.top:3006/api/getbooks?id=1&bookname=%E8%A5%BF %E6%B8%B8 %E8%AE%B0
```

2. 如何对URL进行编码与解码

浏览器提供了URL编码与解码的API，分别是：

- encodeURL() 编码的函数
- decodeURL() 解码的函数

```
encodeURI('黑马程序员')
// 输出字符串  %E9%BB%91 %E9%A9%AC %E7%A8%8B %E5%BA%8F %E5%91%98
decodeURI('%E9%BB%91 %E9%A9%AC')
// 输出字符串  黑马
```

3.URL编码的注意事项

由于浏览器会自动对 URL 地址进行编码操作，因此，大多数情况下，程序员不需要关心 URL地址的编码与解码操作。

更多关于URL编码的知识，请参考如下博客：

https://blog.csdn.net/Lxd_0111/article/details/78028889

1.7 使用xhr发起POST请求

步骤：

1. 创建 xhr 对象
2. 调用 xhr.open() 函数
3. 设置 Content-Type 属性 （固定写法）
4. 调用 xhr.send() 函数， 同时指定要发送的数据
5. 监听 xhr.onreadystatechange 事件

```
// 1. 创建 xhr 对象
const xhr = XMLHttpRequest()
// 2. 调用 open()
xhr.open('POST','http://www.liulongbin.top:3006/api/addbook')
// 3. 设置 Content-Type 属性 （固定写法）
xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded')
// 4. 调用 send() 同时将数据以查询字符串的形式，提交给服务器
xhr.send('bookname:水浒传&author=施耐庵&publisher=天津图书出版社')
// 5. 监听 onreadystatechange 事件
xhr.onreadystatechange = function() {
    if (xhr.readystate === 4 && xhr.status === 200){
            console.log(xhr.responseText)
    }
}
```

2. 数据交换格式

2.1 什么是数据交换格式

数据交换格式，就是 服务器 与 客户端 之间进行 **数据传输与交换的格式** 。

前端领域，经常提及的两种数据交换格式分别是 XML 和 JSON 。其中 XML 用的非常少，所以，我们重点要学习的数据交换格式就是JSON。

2.2 XML

1.什么是XML

XML的英文全称是EXtensible Markup Language， 即可扩展标记语言 。因此，XML和HTML类似，也是一种标记语言。

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
  </head>
  <body></body>
</html>
```

HTML

```
<note>
  <to>ls</to>
  <from>zs</from>
  <heading>通知</heading>
  <body>晚上开会</body>
</note>
```

XML

2.2 XML 和 HTML的区别

XML 和 HTML 虽然都是标记语言，但是，他们两者之间没有任何的关系。

- HTML 被设计用来描述网页上的 **内容** ，是网页内容的载体
- XML 被设计用来 **传输和存储数据** ，是数据的载体

3.XML的缺点

```
<note>
  <to>ls</to>
  <from>zs</from>
  <heading>通知</heading>
  <body>晚上开会</body>
</note>
```

1. XML 格式臃肿，和数据无关的代码多，体积大，传输效率低。
2. 在JavaScript 中解析 XML 比较麻烦

2.3 JSON

1.什么是JSON

概念：JSON的英文全称是JavaScript Object Notation，即“JavaScript对象表示法”。

简单来讲，**JSON 就是 Javascript 对象和数组的字符串表示法**，它使用文本表示一个JS对象或数组的信息，

因此，JSON 的本质是字符串。

作用：JSON是一种**轻量级的文本数据交换格式**，在作用上类似于 XML，专门用于存储和传输数据，但是JSON比XML**更小、更快、更易解析**。

现状：JSON是在2001年开始被推广和使用的数据格式，到现今为止，**JSON 已经成为了主流的数据交换格式**。

2.JSON的两种结构

JSON就是用字符串来表示Javascript的对象和数组。所以，JSON中包含**对象**和**数组**两种结构，通过这两种结构的相互嵌套，可以表示各种复杂的数据结构。

对象结构：对象结构在JOSN中表示为{ } 括起来的内容。数据结构为{ key：value,key:value, ...}的键值对结构。其中，key**必须**是使用**英文的双引号包裹的字符串**，value的数据类型可以是**数字**、**字符串**、**布尔值**、**null**、**数组**、**对象**6种类型

```
{
    name: "zs",
    'age': 20,
    "gender": '男',
    "address": undefined,
    "hobby": ["吃饭", "睡觉", '打豆豆']
    say: function() {}
}
```

```
{
    "name": "zs",
    "age": 20,
    "gender": "男",
    "address": null,
    "hobby": ["吃饭", "睡觉", "打豆豆"]
}
```

数组结构：数组结构在JSON中表示为[ ] 括起来的内容。数据结构为["java","javascript",30, true …]。数组中数据的类型可以是**数字**、**字符串**、**布尔值**、**null**、**数组**、**对象**6种类型。

```
[ "java", "python", "php" ]
[ 100, 200, 300.5 ]
[ true, false, null ]
[ { "name": "zs", "age": 20}, { "name": "ls", "age": 30} ]
[ [ "苹果", "榴莲", "椰子" ], [ 4, 50, 5 ] ]
```

3. JSON语法注意事项：

1. 属性名必须使用`双引号`包裹

2. 字符串类型的值 必须使用双引号 包裹
3. JSON 中 不允许使用单引号表示字符串
4. JSON 中不能写注释
5. JSON 的最外层必须是对象或数组格式
6. 不能使用 undefined 或函数作为 JSON 的值

JSON的作用 ：在计算机与网络之间存储和传输数据。

JSON的本质 ：用字符串来表示JavaScript 对象数据或数组数据。

4.JSON和JS对象的关系：

JSON是JS 对象的字符串表示法，它使用文本表示一个 JS 对象的信息，本质是一个字符串。例如：

```
// 这是一个对象
const obj = {a: 'Hellow',b:'world'}

// 这是一个 JSON 字符串， 本质上是一个字符串
const JSON = '{"a":  "Hello" , "b": "world"}'
```

5.JSON和JS对象的互转

要实现 JOSN 字符串转换为 JS 对象，使用 JSON.parse() **方法：**

```
const obj = JOSN.parse('{"a":  "Hello" , "b": "world"}')
// 结果是 {a: 'Hellow',b:'world'}
```

要实现 JS 对象转换为 JSON 字符串，使用 JSON.stringify() **方法：**

```
const json = JSON.stringify({a: 'Hello', b: 'World'})
//结果是 '{"a": "Hello", "b": "World"}'
```

6. 序列化和反序列化

把**数据对象** **转换为** **字符串**的过程，叫做 **序列化** ，例如：调用 **JSON.stringify()** 函数的操作，叫JSON序列化。

把**字符串** **转换为** **数据对象**的过程，叫做 **反序列化** ，例如：调用 **JSON.parser()** 函数的操作，叫JSON反序列化。

3. 封装自己的Ajax函数

3.1要实现的效果

```
<!-- 1. 导入自定义的ajax函数库 -->
<script src="./itheima.js"></script>


<script>
```

```
    // 2. 调用自定义的 itheima 函数，发起 Ajax 数据请求
    itheima({
        method: '请求类型',
        url: '请求地址',
        data: { /* 请求参数对象 */ },
        success: function(res) { // 成功的回调函数
            console.log(res)     // 打印数据
        }
    })
</script>
```

3.2 定义options参数选项

itheima() 函数是我们自定义的 Ajax 函数，他接收一个配置对象作为参数，配置对象中可以配置如下属性：

- method：请求的类型
- url：　请求的地址
- data　请求携带的数据
- success 请求成功之后的回调函数

3.3 处理data参数

需要把 data对象，转化成查询字符串的格式，从而提交给服务器，因此提前定义 resolveData 函数如下：【

```
/**
 * 处理 data 参数
 * @param {data} 需要发送到服务器的数据
 * @returns {string} 返回拼接好的查询字符串 name=zs&age=10
 */
function resolveData(data) {
  var arr = []
  for (var k in data) {
    arr.push(k + '=' + data[k])
  }
  return arr.join('&')
}
```

3.4 定义itheima函数

在itheima()函数中，需要创建xhr对象，并监听onreadystatechange事件：

```
function itheima(options) {
  var xhr = new XMLHttpRequest()
  // 拼接查询字符串
  var qs = resolveData(options.data)

  // 监听请求状态改变的事件
  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
      var result = JSON.parse(xhr.responseText)
      options.success(result)
    }
  }
}
```

3.5判断请求的类型

不同的请求类型，对应xhr对象的不同操作，因此需要对请求类型进行if… else … 的判断：}

```
if (options.method.toUpperCase() === 'GET') {
    // 发起 GET 请求
    xhr.open(options.method, options.url + '?' + qs)
    xhr.send()
  } else if (options.method.toUpperCase() === 'POST') {
    // 发起 POST 请求
    xhr.open(options.method, options.url)
    xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
    xhr.send(qs)
  }
```

4.XMLHttpRequest level2的新特性

4.1 认识 XMLHttpRequest Level2

1.旧版的XMLHttpRequest 的缺点

1. 只支持文本数据的传输，无法用来读取和上传文件
2. 传送和接收数据时，没有进度信息，只能提示有没有完成

2.XMLHttpRequest Level2 的新功能

1. 可以设置HTTP 请求的时限
2. 可以使用 FormData 对象管理表单数据
3. 可以上传文件
4. 可以获得数据传输的进度信息

4.2 设置HTTP 请求时限

有时，Ajax操作很耗时，而且无法预知要花多少时间。如果网速很慢，用户可能要等很久。新版本的 XMLHttpRequest 对象，增加了 timeout 属性，可以设置HTTP 请求的时限：

```
  xhr.timeout = 3000
```

上面的语句，将最长等待时间设置为3000毫秒，过了这个时限，就自动停止HTTP请求。与之配套的还有一个 timeout 事件，用来指定回调函数：

```
xhr.ontimeout = function() {
    alert('请求超时！')
}
```

4.3 FormData 对象管理表单数据

Ajax 操作往往用来提交表单数据。为了方便表单处理，HTML5 新增了一个FormData对象，可以模拟表单操作：

```
// 1. 新建 FormData 对象
let fd = new FormData()
// 2. 为 FormData 添加表单项
fd.append('uname','zs')
fd.append('upwd','123456')
// 3. 创建 xhr 对象
let xhr = new XMLHttpRequest()
// 4. 指定请求类型与URL地址
xhr.open('POST','http://www.liulongbin.top:3006/api/formdata')
// 5. 直接提交 FormData 对象，这与提交网页表单的效果，完全一样
xhr.send(fd)
```

FormData对象也可以用来获取网页表单的值，实例代码如下：

```
// 获取表单元素
const form = document.querySelector('#form1')
// 监听表单元素的 submit 事件
form.addEventListener('submit',function(e){
    e.preventDefault()
    // 根据 form 表单创建的 FormData 对象，会自动将表单数据填充到 FormData对象中
    const fd = new FormData(from)
    const xhr = new XMLHttpRequest()
    xhr.open('post','http://www.liulongbin.top:3006/api/formdata')
    xhr.send(fd)
    xhr.onreadystatechange = function() {
        if (xhr.readyState === 4 && xhr.status === 200) {
            console.log(JSON.parse(xhr.responstTEXT))
     }
   }
})
```

4.4上传文件

新版 XMLHttpRequest 对象，不仅可以发送文本信息，还可以上传文件

实现步骤：

1. 定义UI结构
2. 验证是否选择了文件
3. 向 FromData 中追加文件
4. 使用 xhr 发起上传文件的请求
5. 监听 onreadystatechenge 事件

1.定义 UI 结构

```
    <!-- 1. 文件选择框 -->
    <input type="file" id="file1" />
    <!-- 2. 上传按钮 -->
    <button id="btnUpload">上传文件</button>
<br />
    <!-- 3. 显示上传到服务器上的图片 -->
    <img src="" alt="" id="img" width="800" />
```

2.验证是否选择了文件

```
// 1. 获取上传文件的按钮
 var btnUpload = document.querySelector('#btnUpload')
 // 2. 为按钮添加 click 事件监听
 btnUpload.addEventListener('click', function() {
     // 3. 获取到选择的文件列表
     var files = document.querySelector('#file1').files
     if (files.length <= 0) {
         return alert('请选择要上传的文件！')
     }
     // ...后续业务逻辑
 })
```

3.向 FormData 中追加文件

```
// 1. 创建 FormData 对象
 var fd = new FormData()
 // 2. 向 FormData 中追加文件
 fd.append('avatar', files[0])
```

4. 使用 xhr 发起上传文件的请求

```
// 1. 创建 xhr 对象
 var xhr = new XMLHttpRequest()
 // 2. 调用 open 函数，指定请求类型与URL地址。其中，请求类型必须为 POST
 xhr.open('POST', 'http://www.liulongbin.top:3006/api/upload/avatar')
 // 3. 发起请求
 xhr.send(fd)
```

5.监听onreadystatechenge 事件

```
xhr.onreadystatechange = function() {
  if (xhr.readyState === 4 && xhr.status === 200) {
    var data = JSON.parse(xhr.responseText)
    if (data.status === 200) { // 上传文件成功
      // 将服务器返回的图片地址，设置为 <img> 标签的 src 属性
      document.querySelector('#img').src = 'http://www.liulongbin.top:3006' + data.url
    } else { // 上传文件失败
      console.log(data.message)
    }
  }
}
```

4.5 显示文件上传进度

新版本的 XMLHttpRequest 对象中，可以通过监听 xhr.upload.onprogress 事件，来获取到文件的上传进度。语法格式如下：

```
// 创建 XHR 对象
const xhr  = new XMLHttpRequest()
// 监听 xhr.upload 的 onprogress 事件
xhr.upload.onprogress = function(e) {
    // e.lengthComputable 是一个布尔值，表示当前上传的资源是否具有可计算的长度
    if(e.lengthComputable) {
        // e.loaded  已传输的字节
        // e.total   需传输的字节
        let percentcomplete = Math.ceil((e.loaded/e.total)) *100
    }
}
```

1.导入需要的库

```
 <link rel="stylesheet" href="./lib/bootstrap.css" />
 <script src="./lib/jquery.js"></script>
```

2.基于Bootstrap渲染进度条

```
<!-- 进度条 -->
    <div class="progress" style="width: 500px; margin: 10px 0;">
      <div class="progress-bar progress-bar-info progress-bar-
striped active" id="percent" style="width: 0%">
        0%
      </div>
    </div>
```

3.监听上传进度的事件

```
xhr.upload.onprogress = function(e) {
    if (e.lengthComputable) {
    // 1. 计算出当前上传进度的百分比
    var percentComplete = Math.ceil((e.loaded / e.total) * 100)
    $('#percent')
        // 2. 设置进度条的宽度
        .attr('style', 'width:' + percentComplete + '%')
        // 3. 显示当前的上传进度百分比
        .html(percentComplete + '%')
    }
 }
```

4. 监听上传完成的事件

```
 xhr.upload.onload = function() {
     $('#percent')
         // 移除上传中的类样式
         .removeClass()
         // 添加上传完成的类样式
         .addClass('progress-bar progress-bar-success')
 }
```

5.jQuery的高级用法

5.1 利用jQuery实现文件上传

1.定义UI结构

```
<!-- 导入 jQuery -->
<script src="./lib/jquery.js"></script>
<!-- 文件选择框 -->
<input type="file" id="file1" />
<!-- 上传文件按钮 -->
<button id="btnUpload">上传</button>
```

2.验证是否选择了文件

```
$('#btnUpload').on('click', function() {
    // 1. 将 jQuery 对象转化为 DOM 对象，并获取选中的文件列表
    var files = $('#file1')[0].files
    // 2. 判断是否选择了文件
    if (files.length <= 0) {
        return alert('请选择图片后再上传！')
    }
})
```

3.向FormData中追加文件

```
// 向 FormData 中追加文件
var fd = new FormData()
fd.append('avatar', files[0])
```

4.使用jQuery发起上传文件的请求

```
$.ajax({
    method: 'POST',
    url: 'http://www.liulongbin.top:3006/api/upload/avatar',
    data: fd,
    // 不修改 Content-Type 属性，使用 FormData 默认的 Content-Type 值
    contentType: false,
    // 不对 FormData 中的数据进行 url 编码，而是将 FormData 数据原样发送到服务器
    processData: false,
    success: function(res) {
        console.log(res)
    }
})
```

5.2jQuery实现loading效果

1.ajaxStart(callback)

Ajax请求 **开始** 时，执行ajaxStart 函数，可以在 ajaxStart 的 callback 中显示 loading效果，实例代码如下：

```
// 自 jQuery 版本 1.8 起，该方法只能被附加到文档
$(document).ajaxStart(function() {
    $('#loading').show()
})
```

注意： $(document).ajaxStart( ) 函数 **会监听当前文档内的所有Ajax请求**

2.ajaxStop(callback)

Ajax请求 **结束** 时，执行 ajaxStop 函数。 可以在 ajaxStop 的 callback 中隐藏 loading 效果，实例代码如下：

```
// 自 jQuery 版本 1.8 起，该方法只能被附加到文档
$(document).ajaxStop(function() {
    $('#loading').hide()
})
```

6.axios

6.1 什么是 axios

Axios 是专注于 **网络数据请求** 的库。

相比于原生的 XMLHttpRequest 对象，axios **简单易用** 。

相比于jQuery，axios 更加 **轻量化** ，只专注于网络数据请求。

6.2axios发起GET请求

axios发起get请求的语法：

```
axios.get('url', { params: { /*参数*/ } }).then(callback)
```

具体的请求示例如下：

```
// 请求的 URL 地址
const url = 'http://www.liulongbin.top:3006/api/get'
// 请求的参数对象
const paramsObj = { name: 'zs', age: 20 }
// 调用 axios.get() 发起 GET 请求
axios.get(url, { params: paramsObj }).then(function(res) {
    // res.data 是服务器返回的数据
    var result = res.data
    console.log(res)
})
```

6.3 axios发起POST请求

axios 发起 post 请求的语法：

```
axios.post('url',{ /*参数*/ }).then(callback)
```

具体的请求示例如下：

```
// 请求的url 地址
const url = 'http://www.liulongbin.top:3006//api/post'
// 要提交到服务器的数据
const dataObj = { location: '北京', address:'顺义' }
// 调用 axios.post()  发起POST请求
axios.post(url,dataObj).then(function(res) {
     // res.data 是服务器返回的数据
    const result = res.data
    console.log(result)
})
```

6.4 直接使用axios发起请求

axios 也提供了类似于jQuery 中 $.ajax()的函数，语法如下：

```
axios({
    method:'请求类型',
    url:'请求的URL地址',
    data:{ /* POST数据 */},
    params: { /* GET参数 */}
}).then(callback)
```

1. 直接使用axios发起GET请求

```
axios.get({
    method: 'GET',
    url : 'http://www.liulongbin.top:3006/api/get'
    params: {
        name: 'zs',
        age:20
   }
}).then(function(res) {
      console.log(res.data)
})
```

2.直接使用axios发起POST请求

```
axios({
    method : 'POST',
    url : 'http://www.liulongbin.top:3006/api/post'
    data : {
    bookname: '程序员的自我修养',
    price: 666
   }
}).then(function(res) {
          console.log(res.data)
})
```