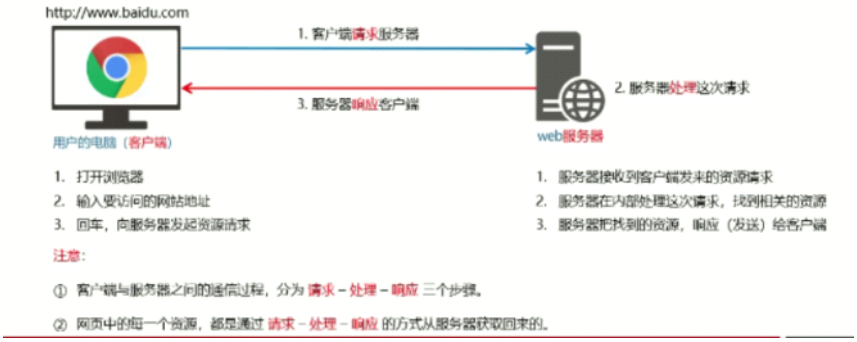
Ajax

2021年2月24日　15:11

服务器：负责存放和对外提供资源的电脑
客户端：负责消费和获取资源的电脑

URL（Uniform Resource Location）：统一资源定位符，用于表示互联网上每个资源的位置。只有通过正确的URL才能成功访问资源。
组成部分：

客户端与服务器的通信过程：



网页中请求数据：利用XMLHttpRequest对象，使用方法：

```
let xhrObj=new XMLHttpRequest()
```

资源请求的方式:

（1）get:向服务器请求数据，例如：通过URL网址获得html、css、js文件

（2）post：向服务器提交数据。例如：登录时向服务器提交用户信息

Ajax:实现网页与服务器之间的数据交互。

jQuery中的Ajax:

$.get() 获取、$.post()提交、$.ajax()既可以获取也可以提交

1.获取数据：$.get(url,[data],[callback])

参数名	参数类型	是否必选	说明
url	string	是	要请求的资源地址
data	object	否	请求资源时间要携带的参数
callback	function	否	请求成功时的回调函数

实例：

```
//不带参数的请求
$.get("http://www.liulongbin.top:3006/api/getbooks",function(res) {
  console.log(res);
})
//带参数的请求
$.get("http://www.liulongbin.top:3006/api/getbooks",{id:1},function(res) {
  console.log(res);
})
```

2.获取数据：$.post(url,[data],[callback])　参数含义与get一致

实例：

```
//post请求
$.post("http://www.liulongbin.top:3006/api/addbook",{
  bookname:"水浒传",
  author:"施耐庵",
  publisher:"上海图书出版社"
},function(res) {
  console.log(res);
})
```

3.$.ajax({

type:"", 请求类型，get或者psot

url:"", 请求的url地址

data:"", 本次请求携带的参数

success:function(){} 成功之后的回调函数

}

)

实例:

```
//$.ajax()的使用
//1.发起get请求
$.ajax({
  type:"get",
  url:"http://www.liulongbin.top:3006/api/getbooks",
  success:function (res) {
    console.log(res)
  }
})
//发起post请求
$.ajax({
  type:"post",
  url:"http://www.liulongbin.top:3006/api/addbook",
  data:{
    bookname:"水浒传",
    author:"施耐庵",
    publisher:"上海图书出版社"
  },
  success:function (res) {
    console.log(res)
  }
})
```

接口:ajax请求数据时的url，每个接口都有请求方式。

接口文档:

2. 接口文档的组成部分

接口文档可以包含很多信息，也可以按需进行精简，不过，一个合格的接口文档，应该包含以下6项内容，从而为接口的调用提供依据：

1. **接口名称**：用来标识各个接口的简单说明，如登录接口，获取图书列表接口等。
2. **接口URL**：接口的调用地址。
3. **调用方式**：接口的调用方式，如 GET 或 POST。
4. **参数格式**：接口需要传递的参数，每个参数必须包含参数名称、参数类型、是否必填、参数说明这4项内容。
5. **响应格式**：接口的返回值的详细描述，一般包含数据名称、数据类型、说明3项内容。
6. 返回示例（可选）：通过对象的形式，例举服务器返回数据的结构。

form表单:

属性:

属性	值	描述
action	URL地址	规定当提交表单时，向何处发送表单数据
method	get或post	规定以何种方式把表单数据提交到 action URL
enctype	application/x-www-form-urlencoded multipart/form-data text/plain	规定在发送表单数据之前如何对其进行编码
target	_blank _self _parent _top *framename*	规定在何处打开 action URL

action：默认值为当前的url地址，当提交数据后，会立即跳转到action指定的地址。

target：默认值时self，在相同的框架或窗口下打开。

method：默认为get。

enctype:默认值为application/x www-form-urlencoded 不编码。

multipart/form-data 不对字符编码，在使用包含文件文件上传控件的表单时，必须 使用该值。

text/plain 空格转换为“+”，但不对特殊字符编码。（很少用）

表单同步提交：submit提交表单时，页面会跳转到action指定的url处。

缺点：

1.页面会发生跳转

2.页面之前的状态和数据会丢失

解决方法：使用Ajax提交数据。使用方法：1.$(表单).submit(function(){})

2.$(表单).on("submit",function(){})

阻止表单的默认提交行为：e.preventDefault.

快速获取表单中的值：$(表单).serialize() 标签必须添加name属性，返回：name属性值=标签中的值

模板引擎：根据程序员指定的模板结构和数据，生成完整的html页面。

优点：1.减少了字符串的拼接操作。

3.使代码更清晰。

2. 使代码更易于阅读和维护。

使用步骤：

1. 导入art-template；
2. 定义数据
3. 定义模板
4. 调用template函数
5. 渲染html结构。

```
<!--  1.引入模板引擎-->
<!--  此时window全局多了一个template函数，template("模板的id",数据对象)-->
<script src="../../template-web.js"></script>
<script src="../../jquery-3.5.1/jquery-3.5.1.min.js"></script>
</head>
<body>
<div id="content"></div>
<!--3.定义模板，type的类型必须是text/htm，必须添加上id属性-->
<script type="text/html" id="tmp">
    <h1>{{name}}</h1>
</script>
<script>
    //2.定义数据
    let data={
        name:"lx"
    }
    //4.调用函数
    let temp=template("tmp",data);  //不需要写#

    //5.渲染html结构
    $("#content").html(temp)
</script>
```

art-template的输出语法：{{}}除了能做一些逻辑运算、三目运算符，还能做其他的输出。

1.原文输出：{{@ value}}

如果要输出的value值中，包含了HTML标签结构，则需要使用原文输出语法，才能保证HTML标签被正常渲染。

```
<body>
    <div id="content"></div>
<!--3.定义模板，type的类型必须是text/htm，必须添加上id属性-->
<script type="text/html" id="tmp">
    <h1>{{name}}</h1>
    <div>{{@ age}}</div>
</script>
<script>
    //2.定义数据
    let data={
        name:"lx",
        age:"<h3>hello</h3>"
    }
```

2.条件输出：

{{if value}}按需要的内容输出{{/if}}

{{if v1}}按需要的内容输出{{else if v2}}按需要的内容输出{{/if}}

```
<script type="text/html" id="tmp">
    <h1>{{name}}</h1>
    <div>{{if flag==0}}flag等于0{{else if flag==1}}flag等于1{{/if}}</div>
</script>
<script>
    //2.定义数据
    let data={
        name:"lx",
        age:"<h3>hello</h3>",
        flag:1
    }
```

3.循环输出：{{each 对象}} 通过$index获取当前循环值得索引，通过$value获得当前值。

{{each arr}}{{$index}}{{$value}}{{/each}}

```
<script type="text/html" id="tmp">
    <h1>{{name}}</h1>
```

```
<script type="text/html" id="tmp">
    <h1>{{name}}</h1>
    <div>{{if flag==0}}flag等于0{{else if flag==1}}flag等于1{{/if}}</div>
    <ul>{{each arr}}<li>循环索引：{{$index}},循环值：{{$value}}</li>{{/each}}</ul>
</script>
  <script>
    //2.定义数据
     let data={
       name:"lx",
       age:"<h3>hello</h3>",
       flag:1,
       arr:[1,2,3]
     }
```

4.过滤器：{{value | 过滤器函数名}} //首先将value值传给过滤器函数，处理完后，将处理的值return再赋值给value

过滤器函数得定义：templat.defaults.imports.过滤器函数名=function(value){return ...}

```
<script type="text/html" id="tmp">
    <h1>{{name}}</h1>
    <div>{{if flag==0}}flag等于0{{else if flag==1}}flag等于1{{/if}}</div>
    <ul>{{each arr}}<li>循环索引：{{$index}},循环值：{{$value}}</li>{{/each}}</ul>
    <div>{{flag | getTrue}}</div>

</script>
  <script>
    //2.定义数据
     let data={
       name:"lx",
       age:"<h3>hello</h3>",
       flag:1,
       arr:[1,2,3]
     }
     template.defaults.imports.getTrue=function(val){
       if(val>0){
         return 3
       }
     }
```

xhr:XMLHttpRequest,JavaScript对象，用于发起网络请求。

get请求使用步骤：

1. 创建xhr对象；
2. 调用xhr.open()函数；
3. 调用xhr.send()函数；
4. 监听xhr.onreadystatechange事件；

```
<script>
  //1.创建xhr对象
  let xhr=new XMLHttpRequest();
  //2.调用open函数,指定请求方式和接口地址
  xhr.open("GET","http://www.liulongbin.top:3006/api/getbooks");
  //3.调用send函数，发起请求
  xhr.send()
  //4.监听onreadystatechange事件
  xhr.onreadystatechange=function () {
    if(xhr.readyState===4 &&xhr.status===200){  //readyState请求状态；status服务器响应状态
      console.log(xhr.responseText)  //responseText是返回的数据
    }
  }
</script>
```

上图中readySate的属性：

1.3 了解xhr对象的readyState属性

XMLHttpRequest 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。

xhr发起带参数的get请求：在open函数里为url地址指定参数即可，添加的参数被称为查询字符串。

xhr. open('GET', 'http://www. liulongbin. top: 3006/api/getbooke?id=1')

```
<script>
  //1.创建xhr对象
  let xhr=new XMLHttpRequest();
  //2.调用open函数,指定请求方式和接口地址
  xhr.open("GET","http://www.liulongbin.top:3006/api/getbooks?id=1");
  //3.调用send函数，发起请求
  xhr.send()
  //4.监听onreadystatechange事件
  xhr.onreadystatechange=function () {
    if(xhr.readyState===4 &&xhr.status===200){  //readyState请求状态；status服务器响应状态
      console.log(xhr.responseText)  //responseText是返回的数据
    }
  }
</script>
```

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，可以将想要发送给服务器的数据添加到URL中。

url编码与解码：浏览器会自动编码url中的中文字符串

encodeURL() //编码 decodeURL()

```
encodeURI("黑马")
decodeURI(encodeURI("黑马"))
```

xhr发起post请求的步骤：

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

4. 监听xhr.onreadystatechange事件；

```
//1. 创建xhr对象
let xhr1=new XMLHttpRequest();
//2. 调用open函数，指定post请求方式；
xhr1.open("post","http://www.liulongbin.top:3006/api/addbook");
//3. 设置Content-type属性，固定写法
xhr1.setRequestHeader("Content-type","application/x-www-form-urlencoded")
//4. 调用send，添加请求数据,以查询字符串的形式添加；
xhr1.send("bookname=水浒传&author=sdfj&publisher=上海");
xhr1.onreadystatechange=function () {
 if(xhr1.readyState===4&&xhr1.status===200){
   console.log(xhr1.responseText)
 }
}
```

数据交换格式：服务器与客户端数据传输与交换的格式。

（1）xml:Xtensible Markup Language,可扩展的标记语言。与html没任何关系

html与xml的区别：xml是网页内容的载体，而xml是数据的载体。

缺点：1. 格式臃肿，和数据无关的代码较多，体积较大，传输效率低。

2. 解析比较麻烦。

（2）json:JavaScript Object Notation, 是JavaScript对象和数组的字符串表示方法，它使用文本表示一个JS对象和数组的信息，JSON本质是字符串。

作用：类似于xml，用于数据存储和传输。但JSON比XML更小、更快、更易解析。

结构：

1.对象结构：{key:value},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.

2.数组结构：[],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 ] ]
```

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和JavaScript对象间的关系：

```
//这是一个对象
var obj = {a: 'Hello', b: 'World'}

//这是一个 JSON 字符串，本质是一个字符串
var json = '{"a": "Hello", "b": "World"}'
```

JSON和JavaScript对象的相互转换：

JSON转对象：JSON.parse(JSON)

对象转JSON：JSON.stringify(object)

序列化和序列化：

1.把数据对象转换为字符串的过程，叫做序列化，例如:调用JSON.stringify0函数的操作，叫做JSON序列化。

2.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.parse()所数的操作，叫做JSON反序列化.

xhr level2:

xhr的缺点:

1.只支持文本数据的传输，不能读取和上传文件。

2.传送和接受数据数据时，没有进度信息，只提示有无完成。

xhr level2的新特新:

1.可以设置http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.可以显示传输的进度信息

1.设置http的请求时限：timeout属性：xhr.timeout=毫秒数；

还可以设置超时是的回调函数：xhr.ontimeout=function(){};

```
let xhr=new XMLHttpRequest();
//设置请求的时限
xhr.timeout=10;
//设置超时的回调函数
xhr.ontimeout=function(){
  console.log("请求超时")
}
xhr.open("GET","http://www.liulongbin.top:3006/api/getbooks?id=1");
xhr.send()
xhr.onreadystatechange=function () {
  if(xhr.readyState===4 &&xhr.status===200){
    console.log(xhr.responseText)
  }
}
```

2.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
var 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
var 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管理表单数据
let fd=new FormData();
//追加数据
fd.append("uname","lx");
fd.append("upwd","123456");
let 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(xhr.responseText)
  }
}
```

第二种形式：

FormData对象也可以用来获取网页表单的值，示例代码如下：

```
// 获取表单元素
var 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 FormData 对象，会自动将表单数据填充到 FormData 对象中
    var fd = new FormData(form)
    var xhr = new XMLHttpRequest()
    xhr.open('POST', 'http://www.liulongbin.top:3006/api/formdata')
    xhr.send(fd)
    xhr.onreadystatechange = function() {}
})
```

3.上传文件：

步骤：1.定义ui结构

2.验证是否选择了文件

3.向FormData中追加文件

4.使用xhr发起文件上传请求

5.监听onreadystatechange事件

```
<form id="form">
  <input type="file" id="filr">
  <button type="submit" id="btn">上传文件</button>
!--    <br/>-->
  <img src="" alt="" id="img" width="800px">
</form>
<script>
  //获取文件上传按钮
  let btn=document.querySelector("#btn");
  //为上传按钮绑定事件
  btn.addEventListener("click",function (e) {
    //阻止默认行为
    e.preventDefault();
   //获取用户选择的文件列表
    let file=document.querySelector("#filr").files  //返回数组
    if(file.length<=0){
      return alert("请上传文件")
    }else{
      console.log("用户上传了文件");
    }
      let fd=new FormData();
      //将文件追加到formdata中
      fd.append("avatar",file[0]);
      let xhr=new XMLHttpRequest()
      xhr.open("post","http://www.liulongbin.top:3006/api/upload/avatar")
      xhr.send(fd);
      xhr.onreadystatechange=function () {
        if(xhr.readyState===4 && xhr.status===200){
          let result=JSON.parse(xhr.responseText);
          if(result.status==200){
            let img=document.querySelector("#img");
            img.src="http://www.liulongbin.top:3006"+result.url
          }else {
            console.log("上传失败")
          }

        }
      }
    })
```

4.显示文件上传进度：

```
// 创建 XHR 对象
var 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 需传输的总字节
        var percentComplete = Math.ceil((e.loaded / e.total) * 100)
    }
}
```

```
let xhr=new XMLHttpRequest();
xhr.upload.onprogress=function(e){
  //布尔值，表示当前文件是否有可计算的长度
  if(e.lengthComputable){
    //e.loaded,表示已经上传的大小
    //e.total表示文件总的大小
    let comp=Math.ceil((e.loaded/e.total)*100);
    console.log(comp)
  }

}
```

jQuery上传文件：

```
$(function () {
 $("#btn").on("click",function (e) {
   e.preventDefault();
    let file=$("#file")[0].files;
    if(file.length<=0){
      console.log("请选择文件上传")
    }
      let fd=new FormData();
      fd.append("avatar",file[0]);
      //必须调用$.ajax()上传数据
      $.ajax({
        method:"post",
        url:"http://www.liulongbin.top:3006/api/upload/avatar",
        data:fd,
        //固定写法，contentType表示不修改content-type的属性
        contentType:false,
        //固定写法，表示不对formdata的数据做处理
        processData:false,
        success:function (res) {
                console.log(res)
        }
      })
 })
})
```

jQuery显示上传进度：

```
<img src="../img/loading.gif" style="display: none">
<script>
  $(function () {
    //ajax一旦请求，就会调用ajaxStart函数，会监听整个文档的ajax请求
    $(document).ajaxStart(function () {
      $("img").show();
    })
    //ajax请求结束时，调用ajaxStop函数
    $(document).ajaxStop(function () {
      $("img").hide()
    })
```

axios：相较于jquery，axios轻量化，专注于网络请求

（1）发起get请求:axios.get(url,{params:数据对象}).then(callback)

```
<script>
  let data={name:"lx",age:24}
  let url="http://www.liulongbin.top:3006/api/get"
  axios.get(url,{params:data}).then(function (res) {
    console.log(res) //res是axios包装的结果，通过res.data获取真正的返回数据；
  })
</script>
```

（2）发起post请求：axios.post(url,{数据对象}).then(callback)

```
//发起post请求
let data={location:"lx",address:24}
let url="http://www.liulongbin.top:3006/api/post"
axios.post(url,{data}).then(function (res) {
  console.log(res)  //res.data存的是真正的数据
})
```

（3）直接发起请求：

post和get请求的不同点是：get传参的属性是params、而post的传参属性是data。如下。

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

```
//发起get请求
let data={name:"lx",age:24}
let url="http://www.liulongbin.top:3006/api/get"
axios({
  method:"get",
  url,
  params:data
}).then(function (res) {
  console.log(res.data)
})
```

```
//发起post请求
let data={location:"lx",address:24}
let url="http://www.liulongbin.top:3006/api/post"
axios({
  method:"post",
  url,
  data
}).then(function (res) {
  console.log(res.data)
})
```

axios的全局配置：

axios.defaults.timeout=5000 //设置请求超时的时间

axios.defaluts.baseURL="http://localhost:80" //默认基础请求地址

axios.defaults.heders["mytoken"]="dklshfjkasskdlf" //设置请求头

通过上面的参数配置全局属性，不太灵活，我们可以通过拦截器，为不同的请求设置不同的参数.

(1)请求拦截器：在请求之前做一些配置处理

```
//请求拦截器
axios.interceptors.request.use(function (config) {
        //在请求发出之前做一些配置
  console.log(config.url);
  // config.headers.mytoen="hello";
  //在结尾一定要添加config参数，否则设置的参数配置无效
  return config;
},function f(err) {
  // 处理请求中的错误信息，该函数可意不写
  console.log(err)
})
```

(2) 响应拦截器：对返回的结果做预处理，交给后续的响应回调函数

```
//响应拦截器
axios.interceptors.response.use((res)=>{
  //对返回的数据做预处理
  //交给后续的响应函数
  //必须调用return，否则对数据的处理无效
  return res.data;
},function (err) {
  //处理响应的错误信息
  console.log(err)
})
```

同源策略：

同源：若两个页面的http协议、域名和端口号都相同的话，则两个页面具有相同的源。

同源策略(same origin policy)：是浏览器提供的一个安全功能。同源策略限制了从一个源加载的文档或脚本如何与另一个源的资源之间的交互。及A网站的JavaScript不能与非同源的网站c之间进行资源的交互。

如：1.无法读取非同源网页的Cookie、LocalStorage 和IndexedDB

2.无法接触非同源网页的DOM

3.无法向非同源地址发送Ajax请求

跨域：两个url的http协议、域名、端口不一致，就是跨域。由浏览器的同源策略不允许不同源的url间的资源交互导致。

跨域的解决方案：jsonp、CORS

jsonp:兼容性较好（支持低版本浏览器）只支持get请求，不支持post请求。

CORS：兼容性较差，支持get和post请求。

JSONP:JSONP的实现原理，就是通过<script>标签的src属性（src不受同源策略的影响），请求跨域的数据接口，并通过函数调用的形式，接收跨域接口响应回来的数据。

定义一个 success 回调函数：

```
<script>
  function success(data) {
    console.log('获取到了data数据 ')
    console.log(data)
  }
</script>
```

通过 <script> 标签，请求接口数据：

```
<script src="http://ajax.frontend.itheima.net:3006/api/jsonp?callback=success&name=zs&age=20"></script>
```

```
<script>
  function success(data) {
    console.log("jsonp返回来的数据是：")
    console.log(data);
  }
</script>
<script src="http://ajax.frontend.itheima.net:3006/api/jsonp?callback=success?name=zs&age=24"></script>
```

使用jQuery发起jsonp请求：

```
$(function () {
  $.ajax({
    url:"http://ajax.frontend.itheima.net:3006/api/jsonp?name=zs&age=24",
    dataType:"jsonp",
    success:function (res) {
      console.log(res)
    }
  })
})
```

默认情况下，使用jQuery发起JSONP请求,会自动携带一个callback=jQueryxxx的参数，jQucryxxx是随机生成的一个的一个回调函数，也可手动设置属性：jsonp："名称"，设置callback的名称，将callback替换;jsonpCallBack，一般不用："函数名",自定义函数名称。

防抖：

防抖策略：当一个事件被触发后，延迟n秒执行回调函数，如果在n秒内事件又被触发。则重新计时。

应用场景：在输入框中输入数据时，通过防抖策略，只有在输入完后，才执行查询请求，可以有效减少请求的次数，节约请求资源。

实现输入框的防抖：

```
var timer = null                    // 1. 防抖动的 timer

function debounceSearch(keywords) { // 2. 定义防抖的函数
  timer = setTimeout(function() {
  // 发起 JSONP 请求
  getSuggestList(keywords)
  }, 500)
}

$('#ipt').on('keyup', function() {  // 3. 在触发 keyup 事件时，立即清空 timer
  clearTimeout(timer)
  // ...省略其他代码
  debounceSearch(keywords)
})
```

```
<button>按钮</button>
<script>
  //1.设置防抖的timer
  let timer=null
  //2.定义防抖函数
  function printMessage() {
    timer=setTimeout(function () {
        console.log("hello");
    },500)
  }
  //3.绑定事件，清除timer
  document.querySelector("button").addEventListener("click",function () {
      clearTimeout(timer);
      printMessage()
  })
</script>
```

节流：可以减少一段时间内事件的触发频率。

节流阀：节流阀为空，表示可执行下次操作；若不为空，则不能执行。当前操作执行完毕后，必须将节流阀设置为空。每次执行操作前，要检查节流阀是否为空。

节流的应用：1.鼠标不断的触发某个事件（如点击），只在单位时间内触发一次。

2.懒加载的时候要监听计算滚动条的位置，但不必每次滑动都触发，可以降低计算频率，从而节约cpu的资源。

案例：鼠标跟随效果

```
<img src="angel.gif" id="angel">
<script>
  $(function () {
    //定义节流阀
    let timer=null;
    let angel=$("#angel")
    $(document).on("mousemove",function (e) {
      //判断节流阀是否为空
      if (timer){return}
      timer=setTimeout(function () {
        let x=e.pageX;
        let y=e.pageY;
        angel.css("top",y).css("left",x);
        //操作执行完毕后，将节流阀设置为null
        timer=null;
      },150)
    })
  })
```

防抖和截留的区别：

如果事件被频繁触发，防抖能保证只有最后一次生效，前面的n次都被忽略，而节流能够减少事件触发的频率，可以认为节流是有选择的触发一部分事件。

http协议：

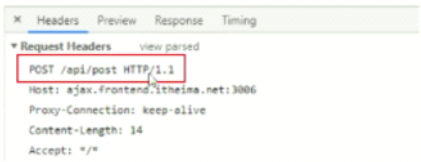
通信：信息的传递和交互。三要素：通信主体、通信的内容，通信的方式。

通信协议：通信双方完成通信所必须遵守的规则和约定。客户端与服务器之间要实现网页内容的传输，则通信双方必须遵守网页内容的传输协议（http协议）。

http协议的交互模型：请求/响应模型

http请求消息：客户端发起的请求叫做http请求，客户端发送给服务器的消息叫做http请求消息（请求报文）。
http请求消息的组成部分：请求行、请求头部、空行、请求体。

（1）请求行：由请求方式、请求的url、http协议版本组成，他们之间用空格隔开。



（2）请求头部：用来描述客户端信息，从而把客户端相关的信息告知服务器。请求头部由多行键/值对组成，每行的键和值之间用英文的冒号分隔。

头部字段	说明
Host	要请求的服务器域名
Connection	客户端与服务器的连接方式(close 或 keepalive)
Content-Length	用来描述请求体的大小
Accept	客户端可识别的响应内容类型列表
User-Agent	产生请求的浏览器类型
Content-Type	客户端告诉服务器实际发送的数据类型
Accept-Encoding	客户端可接收的内容压缩编码形式
Accept-Language	用户期望获得的自然语言的优先顺序

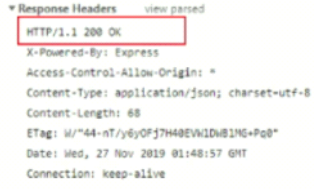
（3）空行：最后一个请求头字段的后面是一个空行， 通知服务器请求头部至此结束。请求消息中的空行，用来分隔清求头部与请求体.

（4）请求体：存放通过post方式提交到服务器的数据。注意，get请求没有请求体！

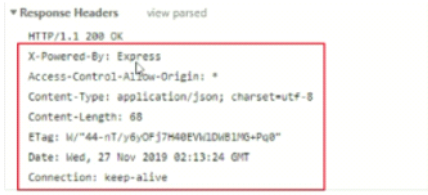
http响应消息（响应报文）：由状态行、响应头部、空行、响应体组成。

（1）状态行：由http协议、状态码、状态码的描述文本组成，他们之间用空格隔开。

状态行由 HTTP 协议版本、状态码和状态码的描述文本 3 个部分组成，他们之间使用空格隔开；



（2）响应头部：用来描述服务器的基本信息，响应头部由多行独/值对组成，每行的键和值之间用英文的冒号分隔。



（3）空行：用来通知客户端响应头部至此结束. 响应消息中的空行，用来分隔响应头部与响应体。

（4）响应体：存放服务器相应给客户端的内容。

http请求的方法：用来表明要对服务器上的资源执行的操作。常用的有GET和POST.

序号	方法	描述
1	GET	(查询)发送请求来获得服务器上的资源，请求体中不会包含请求数据，请求数据放在协议头中。
2	POST	(新增)向服务器提交资源（例如提交表单或上传文件）。数据被包含在请求体中提交给服务器。
3	PUT	(修改)向服务器提交资源，并使用提交的新资源，替换掉服务器对应的旧资源。
4	DELETE	(删除)请求服务器删除指定的资源。
5	HEAD	HEAD 方法请求一个与 GET 请求的响应相同的响应，但没有响应体。
6	OPTIONS	获取http服务器支持的http请求方法，允许客户端查看服务器的性能，比如ajax跨域时的预检等。
7	CONNECT	建立一个到由目标资源标识的服务器的隧道。
8	TRACE	沿着到目标资源的路径执行一个消息环回测试，主要用于测试或诊断。
9	PATCH	是对 PUT 方法的补充，用来对已知资源进行局部更新。

响应状态码：也属于HTTP协议的一部分，用来标识响应的状态。响应状态码会随着响应消息一-起被发送至客户端浏览器，浏览器根据服务器返回的响应状态码，就能知道这次HTTP请求的结果是成功还是失败了。

HTTP状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字用来对状态码进行细分。共分五种类型：

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作（实际开发中很少遇到 1** 类型的状态码）
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

状态码	状态码英文名称	中文描述
200	OK	请求成功。一般用于 GET 与 POST 请求
201	Created	已创建。成功请求并创建了新的资源，通常用于 POST 或 PUT 请求

状态码	状态码英文名称	中文描述
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
302	Found	临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源（响应消息中不包含响应体）。客户端通常会缓存访问过的资源。

状态码	状态码英文名称	中文描述
400	Bad Request	1、语义有误，当前请求无法被服务器理解。除非进行修改，否则客户端不应该重复提交这个请求。 2、请求参数有误。
401	Unauthorized	当前请求需要用户验证。
403	Forbidden	服务器已经理解请求，但是拒绝执行它。
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。
408	Request Timeout	请求超时。服务器等待客户端发送的请求时间过长，超时。

状态码	状态码英文名称	中文描述
500	Internal Server Error	服务器内部错误，无法完成请求。
501	Not Implemented	服务器不支持该请求方法，无法完成请求。只有 GET 和 HEAD 请求方法是要求每个服务器必须支持的，其它请求方法在不支持的服务器上会返回501
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。