3. Vue对象提供的属性功能
3.1 过滤器
3.1.1 使用Vue.filter()进行全局定义
3.1.2 在vue对象中通过filters属性来定义
3.2 阻止事件冒泡和刷新页面
事件冒泡
阻止事件冒泡
阻止页面刷新
3.4 计算和侦听属性
3.4.1 计算属性
3.4.2 监听属性
3.5 vue对象的生命周期
4. 通过axios实现数据请求
4.1 json
4.1.1 json数据的语法
4.1.2 js中提供的json数据转换方法
4.2 ajax
4.2.1 数据接口
4.2.2 前后端分离
4.2.3 ajax的使用
4.2.4 同源策略
4.2.5 ajax跨域(跨源)方案之CORS
5. 组件开发
5.1 组件[component]
5.1.1 默认组件

3. Vue对象提供的属性功能

3.1 过滤器

过滤器，就是vue允许开发者自定义的文本格式化函数，可以使用在两个地方：输出内容和操作数据中。

定义过滤器的方式有两种。

3.1.1 使用Vue.filter()进行全局定义

```
Vue.filter("RMB1", function(v){
    //就是来格式化(处理)v这个数据的
    if(v==0){
            return v
    }

    return v+"元"
})
```

3.1.2 在vue对象中通过filters属性来定义

```
var vm = new Vue({
  el:"#app",
  data:{},
  filters:{
    RMB2:function(value){
      if(value==''){
        return;
      }else{
        return '¥ '+value;
      }
    }
    }
});
```

3.2 阻止事件冒泡和刷新页面

事件冒泡

在js的事件操作中,子元素的事件触发以后,会默认情况把事件传播给其父级元素,然后一层层往外传播,这种现象就是"事件冒泡".

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>事件冒泡</title>
</head>
<body>
    <div class="div1" onclick="alert('双加666')" onmouseover="alert('div1')">
        <div class="div2"  onmouseover="alert('div2')">
            <div class="div3"  onmouseover="alert('div3')">
                <p>点我呀~</p>
            </div>
        </div>
    </div>
    <script>

    </script>
</body>
</html>
```

阻止事件冒泡

下面案例中,在事件绑定的时候,vue.js提供了一个属性.top可以帮助我们阻止事件往外传播.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="js/vue.js"></script>
    <style>
    .box{
        background-color: #fff;
        border-radius: 5px;  /* 边框圆角 */
        padding-top: 15px;
        padding-left: 30px;
        padding-bottom: 15px;
        width: 290px;
        height: 160px;
        position: fixed;
        margin: auto;
        left: 0px;
        right: 0px;
        top:0;
        bottom: 0;
    }
    .container{
        background: rgba(0,0,0,0.6);
        width: 100%;
        margin:auto;
        position: fixed;
        top:0;
        left: 0;
        bottom:0;
        right:0;
    }
    </style>
</head>
<body>
    <div id="app">
        <h1 @click="is_show=true">显示</h1>
        <div class="container" v-show="is_show" @click="is_show=false">
            <div class="box" @click.stop="">
                账号: <input type="text"><br><br>
                密码: <input type="password"><br><br>
                <input type="submit" vlaue="提交">
           </div>
        </div>
    </div>
    <script>

        let vm = new Vue({
            el:"#app",
            data:{
                is_show:false,
            },
            methods:{
```

```
            },
        })
    </script>
</body>
</html>
```

阻止页面刷新

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="js/vue.js"></script>
</head>
<body>
    <div id="app">
        <!-- 超链接\表单中的提交,希望阻止页面刷新,可以使用 @事件.prevent="" -->
        <a href="" @click.prevent="">百度</a>
    </div>
    <script>

        let vm = new Vue({
            el:"#app",
            data:{
            },

        })
    </script>
</body>
</html>
```

使用.stop和.prevent的例子

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        .box1{
            width: 200px;
            height: 200px;
            background: #ccc;
```

```
        }
        .box2{
            width: 100px;
            height: 100px;
            background: pink;
        }
    </style>
    <script src="js/vue.min.js"></script>
    <script>
    window.onload = function(){
        var vm = new Vue({
            el:"#app",
            data:{}
        })
    }
    </script>
</head>
<body>
    <div id="app">
        <div class="box1" @click="alert('box1')">
            <div class="box2" @click.stop.prevent="alert('box2')"></div>   <!--
@click.stop来阻止事件冒泡 -->
        </div>

        <form action="#">
            <input type="text">
            <input type="submit">
            <input type="submit" value="提交02" @click.prevent=""> <!-- @click.prevent来
阻止表单提交 -->
        </form>
    </div>

</body>
</html>
```

3.4 计算和侦听属性

3.4.1 计算属性

字符串反转

```
<script>
    let str1 = "hello";
    let reverse_str = str1.split('').reverse().join("");
    console.log(reverse_str);
</script>
```

如果直接把反转的代码写在元素中，则会使得其他同事在开发时时不易发现数据被调整了，所以vue提供了一个计算属性(computed)，可以让我们把调整data数据的代码存在在该属性中。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="js/vue.min.js"></script>
        <script>
        window.onload = function(){
        var vm = new Vue({
            el:"#app",
            data:{
                str1: "abcdefgh"
            },
            computed:{    //计算属性：里面的函数都必须有返回值
                strRevs: function(){
                    var ret = this.str1.split("").reverse().join("");
                    return ret
                }
            }
        });
    }
    </script>
</head>
<body>
    <div id="app">
         <p>{{ str1 }}</p>
         <p>{{ strRevs }}</p>
    </div>
</body>
</html>
```

3.4.2 监听属性

侦听属性，可以帮助我们侦听data某个数据的变化，从而做相应的自定义操作。

侦听属性是一个对象，它的键是要监听的对象或者变量，值一般是函数，当侦听的data数据发生变化时，会自定执行的对应函数，这个函数在被调用时，vue会传入两个形参，第一个是变化前的数据值，第二个是变化后的数据值。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="js/vue.min.js"></script>
    <script>
      window.onload = function(){
```

```
        var vm = new Vue({
            el:"#app",
            data:{
                num:20
            },
            watch:{
                num:function(newval,oldval){
                    //num发生变化的时候，要执行的代码
                    console.log("num已经发生了变化！");
                }
            }
        })
      }
    </script>
</head>
<body>
    <div id="app">
        <p>{{ num }}</p>
        <button @click="num++">按钮</button>
    </div>
</body>
</html>
```

3.5 vue对象的生命周期

每个Vue对象在创建时都要经过一系列的初始化过程。在这个过程中Vue.js会自动运行一些叫做生命周期的的钩子函数，我们可以使用这些函数，在对象创建的不同阶段加上我们需要的代码，实现特定的功能。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="js/vue.min.js"></script>
    <script>
    window.onload = function(){
        var vm = new Vue({
            el:"#app",
            data:{
                num:0
            },
            beforeCreate:function(){
                console.log("beforeCreate,vm对象尚未创建,num="+ this.num);  //undefined
                this.name=10; // 此时没有this对象呢，所以设置的name无效，被在创建对象的时候被覆盖为0
            },
            created:function(){
```

```
                console.log("created,vm对象创建完成,设置好了要控制的元素范围,num="+this.num
);  // 0
                this.num = 20;
            },
            beforeMount:function(){
                console.log( this.$el.innerHTML ); // <p>{{num}}</p>
                console.log("beforeMount,vm对象尚未把data数据显示到页面中,num="+this.num );
// 20
                this.num = 30;
            },
            mounted:function(){
                console.log( this.$el.innerHTML ); // <p>30</p>
                console.log("mounted,vm对象已经把data数据显示到页面中,num="+this.num); //
30
            },
            beforeUpdate:function(){
                // this.$el 就是我们上面的el属性了, $el表示当前vue.js所控制的元素#app
                console.log( this.$el.innerHTML );  // <p>30</p>
                console.log("beforeUpdate,vm对象尚未把更新后的data数据显示到页面
中,num="+this.num); // beforeUpdate----31

            },
            updated:function(){
                console.log( this.$el.innerHTML ); // <p>31</p>
                console.log("updated,vm对象已经把过呢更新后的data数据显示到页面中,num=" +
this.num ); // updated----31
            },
        });
    }
    </script>
</head>
<body>
    <div id="app">
        <p>{{num}}</p>
        <button @click="num++">按钮</button>
    </div>
</body>
</html>
```

总结:

在vue使用的过程中，如果要初始化操作，把初始化操作的代码放在 mounted 中执行。
mounted阶段就是在vm对象已经把data数据实现到页面以后。一般页面初始化使用。例如，用户访问页面加载成功以后，就要执行的ajax请求。

另一个就是created，这个阶段就是在 vue对象创建以后，把ajax请求后端数据的代码放进 created

4. 通过axios实现数据请求

vue.js默认没有提供ajax功能的。

所以使用vue的时候，一般都会使用axios的插件来实现ajax与后端服务器的数据交互。

注意，axios本质上就是javascript的ajax封装，所以会被同源策略限制。

下载地址：

```
https://unpkg.com/axios@0.18.0/dist/axios.js
https://unpkg.com/axios@0.18.0/dist/axios.min.js
```

axios提供发送请求的常用方法有两个：axios.get() 和 axios.post() 。

```
// 发送get请求
    // 参数1：必填，字符串，请求的数据接口的url地址
    // 参数2：必填，json对象，要提供给数据接口的参数
    // 参数3：可选，json对象，请求头信息
    axios.get('/user',{
            ID:'12345',
      })
    .then(function (response) { // 请求成功以后的回调函数
            console.log("请求成功");
            console.log(response);
    })
    .catch(function (error) {   // 请求失败以后的回调函数
            console.log("请求失败");
            console.log(error);
    });

// 发送post请求，参数和使用和axios.get()一样。
    axios.post('/user',{
        params:{
            firstName: 'Fred',
            lastName: 'Flintstone'
        }
    })
    .then(function (response) {
      console.log(response);
    })
    .catch(function (error) {
      console.log(error);
    });

// b'firstName=Fred&lastName=Flintstone'
```

4.1 json

json是 JavaScript Object Notation 的首字母缩写，单词的意思是javascript对象表示法，这里说的json指的是类似于javascript对象的一种数据格式。

json的作用：在不同的系统平台，或不同编程语言之间传递数据。

4.1.1 json数据的语法

json数据对象类似于JavaScript中的对象，但是它的键对应的值里面是没有函数方法的，值可以是普通变量，不支持undefined，值还可以是数组或者json对象。

```
// json数据的对象格式：
{
    "name":"tom",
    "age":18
}

// json数据的数组格式：
["tom",18,"programmer"]
```

复杂的json格式数据可以包含对象和数组的写法。

```
{
  "name":"小明",
  "age":200,
  "fav":["code","eat","swim","read"],
  "son":{
    "name":"小小明",
    "age":100,
  }
}

// 数组结构也可以作为json传输数据。
```

json数据可以保存在.json文件中，一般里面就只有一个json对象。

总结：

1. json文件的后缀是json
2. json文件一般保存一个单一的json数据对象
3. json数据的属性不能是方法或者undefined，属性值只能：数值、字符串、对象和数组
4. json数据只使用双引号、每一个属性成员之间使用逗号隔开，并且最后一个成员没有逗号。

```
{
   "name":"小明",
   "age":200,
   "fav":["code","eat","swim","read"],
   "son":{
     "name":"小小明",
     "age":100
   }
 }
```

4.1.2 js中提供的json数据转换方法

javascript提供了一个JSON对象来操作json数据的数据转换.

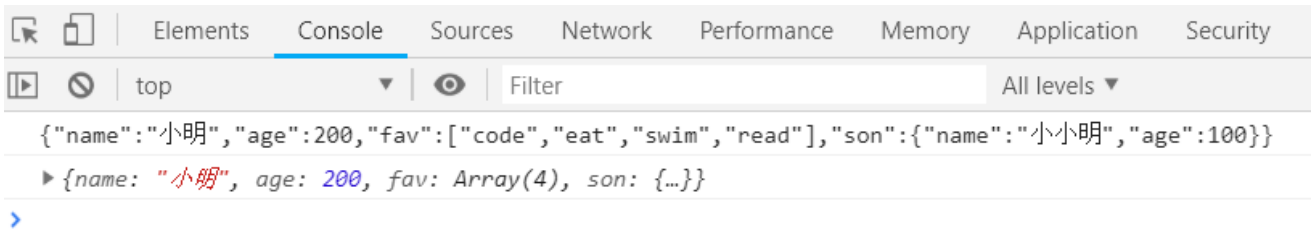
方法	参数	返回值	描述
stringify	json对象	字符串	json对象转成字符串
parse	字符串	json对象	字符串格式的json数据转成json对象

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
    var json_obj = {
      "name":"小明",
      "age":200,
      "fav":["code","eat","swim","read"],
      "son":{
        "name":"小小明",
        "age":100
      }
    };
        // json对象转换成字符串格式的json数据
    var data_str = JSON.stringify(json_obj);
    console.log( data_str );
        // 字符串格式的json数据转换成json对象
    var data_json = JSON.parse(data_str);
    console.log( data_json );
    </script>
```

```
</head>
<body>

</body>
</html>
```

代码执行结果：



4.2 ajax

ajax，一般中文称之为："阿贾克斯"，是英文 “Async Javascript And Xml”的简写，译作：异步js和xml数据传输数据。

ajax的作用： ajax可以让js代替浏览器向后端程序发送http请求，与后端通信，在用户不知道的情况下操作数据和信息，从而实现页面局部刷新数据/无刷新更新数据。

所以开发中ajax是很常用的技术，主要用于操作后端提供的`数据接口`，从而实现网站的`前后端分离`。

ajax技术的原理是实例化js的XMLHttpRequest对象，使用此对象提供的内置方法就可以与后端进行数据通信。

4.2.1 数据接口

数据接口，也叫api接口，表示`后端提供`操作数据/功能的url地址给客户端使用。

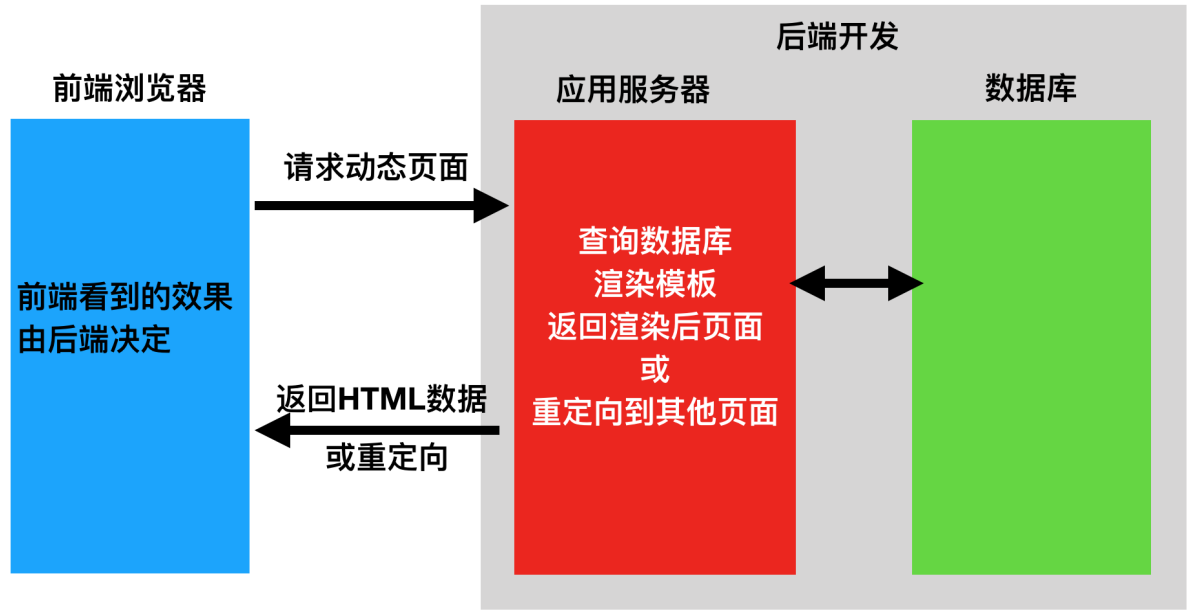
客户端通过发起请求向服务端提供的url地址申请操作数据【操作一般：增删查改】

同时在工作中，大部分数据接口都不是手写，而是通过函数库/框架来生成。

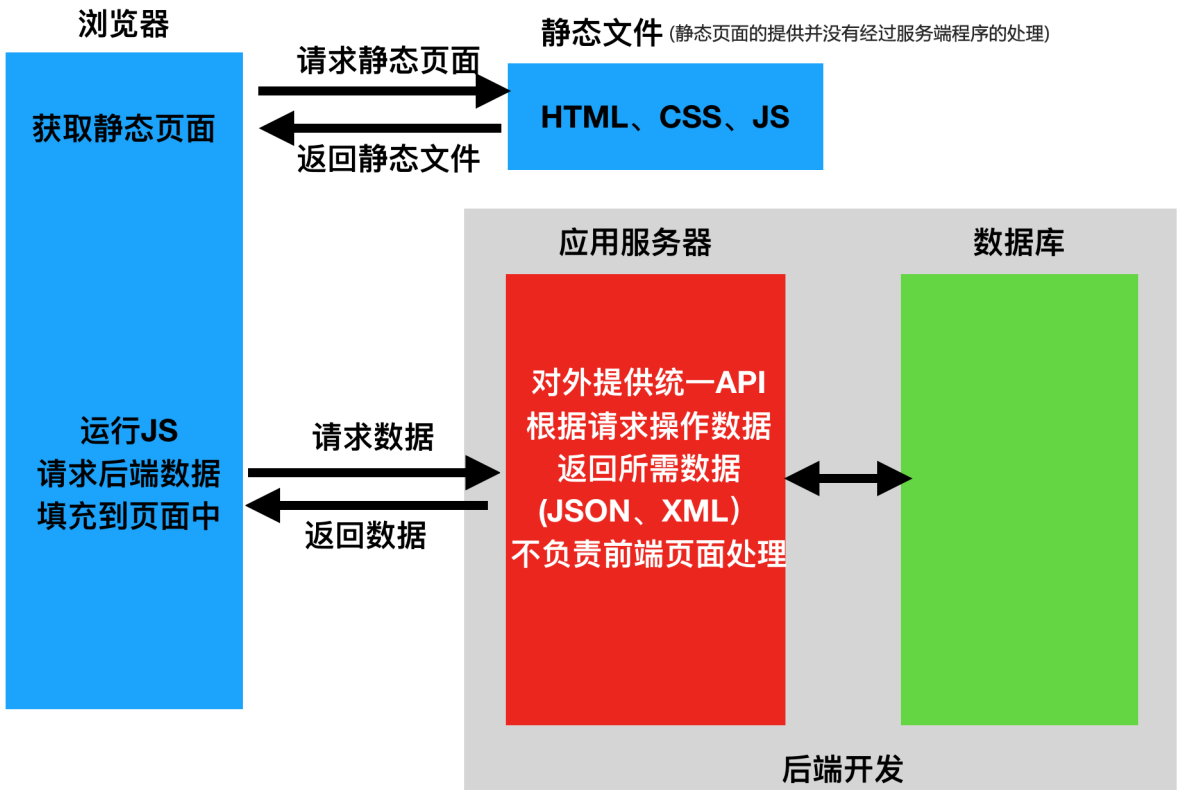
4.2.2 前后端分离

在开发Web应用中，有两种应用模式：

- 前后端不分离



- 前后端分离



4.2.3 ajax的使用

ajax的使用必须与服务端程序配合使用，但是目前我们先学习ajax的使用，所以暂时先不涉及到服务端python代码的编写。因此，我们可以使用别人写好的数据接口进行调用。

jQuery将ajax封装成了一个函数$.ajax()，我们可以直接用这个函数来执行ajax请求。

接口	地址
天气接口	http://wthrcdn.etouch.cn/weather_mini?city=城市名称
音乐接口搜索	http://tingapi.ting.baidu.com/v1/restserver/ting?method=baidu.ting.search.catalogSug&query=歌曲标题
音乐信息接口	http://tingapi.ting.baidu.com/v1/restserver/ting?method=baidu.ting.song.play&songid=音乐ID

编写代码获取接口提供的数据：

jQ版本

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script>
    $(function(){
        $("#btn").on("click",function(){
            $.ajax({
                // 后端程序的url地址
                url: 'http://wthrcdn.etouch.cn/weather_mini',
                // 也可以使用method，提交数据的方式，默认是'GET'，常用的还有'POST'
                type: 'get',
                dataType: 'json',  // 返回的数据格式，常用的有是'json','html',"jsonp"
                data:{ // 设置发送给服务器的数据，如果是get请求，也可以写在url地址的?后面
                    "city":'北京'
                }
            })
            .done(function(resp) {      // 请求成功以后的操作
                console.log(resp);
            })
            .fail(function(error) {     // 请求失败以后的操作
                console.log(error);
            });
        });
    })
    </script>
</head>
<body>
<button id="btn">点击获取数据</button>
</body>
</html>
```

点击按钮以后的效果：

总结:

1. 发送ajax请求，要通过$.ajax()，参数是对象，里面有固定的参数名称。

```
$.ajax({
  "url":"数据接口url地址",
  "method":"http请求方式，前端只支持get和post",
  "dataType":"设置服务器返回的数据格式，常用的json，html，jsonp，默认值就是json",
  // 要发送给后端的数据参数，post时，数据必须写在data，get可以写在data,也可以跟在地址栏?号后面
  "data":{
    "数据名称":"数据值",
  }
}).then(function(resp){ // ajax请求数据成功时会自动调用then方法的匿名函数
  console.log( resp ); // 服务端返回的数据
}).fail(function(error){ // ajax请求数据失败时会自动调用fail方法的匿名函数
  console.log( error );
});
```

2. ajax的使用往往配合事件操作进行调用。

jQuery还提供了$.get 和 $post简写$.ajax的操作。

```
// 发送get请求
// 参数1：数据接口的请求地址
// 参数2：发送给接口地址的数据参数
// 参数3：ajax请求成功以后，调用的匿名函数，匿名函数的第一个参数还是服务端返回的数据
// 参数4：设置服务端返回的数据格式，告诉给jQuery
$.get("test.php", { "func": "getNameAndTime" },
   function(data){
     alert(data.name); // John
     console.log(data.time); //  2pm
   }, "json");
```

```
// 发送post请求
// 参数1：数据接口的请求地址
// 参数2：发送给接口地址的数据参数
// 参数3：ajax请求成功以后，调用的匿名函数，匿名函数的第一个参数还是服务端返回的数据
// 参数4：设置服务端返回的数据格式，告诉给jQuery
$.post("test.php", { "func": "getNameAndTime" },
   function(data){
     alert(data.name); // John
     console.log(data.time); //  2pm
   }, "json");
```

4.2.4 同源策略

同源策略，是浏览器为了保护用户信息安全的一种安全机制。所谓的同源就是指代通信的两个地址（例如服务端接口地址与浏览器客户端页面地址）之间比较，是否协议、域名(IP)和端口相同。不同源的客户端脚本[javascript]在没有明确授权的情况下，没有权限读写对方信息。

ajax本质上还是javascript，是运行在浏览器中的脚本语言，所以会被受到浏览器的同源策略所限制。

前端地址： `http://www.oldboy.cn/index.html`	**是否同源**	**原因**
`http://www.oldboy.cn/user/login.html`	是	协议、域名、端口相同
`http://www.oldboy.cn/about.html`	是	协议、域名、端口相同
`https://www.oldboy.cn/user/login.html`	否	协议不同 (https和http)
`http:/www.oldboy.cn:5000/user/login.html`	否	端口 不同(5000和80)
`http://bbs.oldboy.cn/user/login.html`	否	域名不同 (bbs和www)

同源策略针对ajax的拦截，代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script>
    $(function(){
        $("#btn").on("click",function(){
            $.ajax({
                url: 'http://weatherapi.market.xiaomi.com/wtr-v2/weather',
                type: 'get',
                dataType: 'json',
                data:{
                    "cityId":101010100
                }
            })
```

```
            .done(function(resp) {      // 请求成功以后的操作
                console.log(resp);
            })
            .fail(function(error) {     // 请求失败以后的操作
                console.log(error);
            });
        });
    })
    </script>
</head>
<body>
<button id="btn">点击获取数据</button>
</body>
</html>
```

效果：

4.2.5 ajax跨域(跨源)方案之CORS

CORS是一个W3C标准，全称是"跨域资源共享"，它允许浏览器向跨源的后端服务器发出ajax请求，从而克服了AJAX只能同源使用的限制。

实现CORS主要依靠后端服务器中响应数据中设置响应头信息返回的。

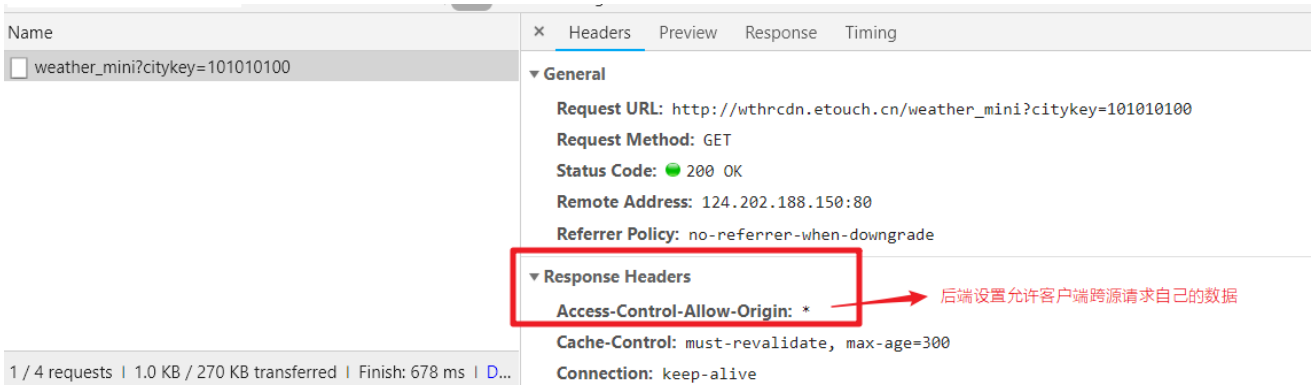
response = new Response()

response .set_header("")

```
// 在响应行信息里面设置以下内容：
Access-Control-Allow-Origin: ajax所在的域名地址

Access-Control-Allow-Origin: www.oldboy.cn  # 表示只允许www.oldboy.cn域名的客户端的ajax跨域访问

// * 表示任意源，表示允许任意源下的客户端的ajax都可以访问当前服务端信息
Access-Control-Allow-Origin: *
```



总结：

```
0. 同源策略：浏览器的一种保护用户数据的一种安全机制。
   浏览器会限制脚本语法不能跨源访问其他源的数据地址。
   同源：判断两个通信的地址之间，是否协议，域名[IP]，端口一致。

   ajax:   http://127.0.0.1/index.html
   api数据接口:   http://localhost/index

   这两个是同源么？不是同源的。

1. ajax默认情况下会受到同源策略的影响，一旦受到影响会报错误如下：
     No 'Access-Control-Allow-Origin' header is present on the requested resource

2. 解决ajax只能同源访问数据接口的方式：
   1. 在服务端的响应行中设置：
      Access-Control-Allow-Origin: 允许访问的域名地址
```

5. 组件开发

5.1 组件[component]

组件（Component）是自定义封装的功能。在前端开发过程中，经常出现多个网页的功能是重复的，而且很多不同的网站之间，也存在同样的功能。

而在网页中实现一个功能，需要使用html定义功能的内容结构，使用css声明功能的外观样式，还要使用js来定义功能的特效，因此就产生了把一个功能相关的[HTML、css和javascript]代码封装在一起组成一个整体的代码块封装模式，我们称之为“组件”。

所以，组件就是一个html网页中的功能，一般就是一个标签，标签中有自己的内容结构，样式和特效。

这样，前端人员就可以在开发时，只需要书写一次代码，随处引入即可使用。

组件有两种：默认组件[全局组件] 和 单文件组件

5.1.1 默认组件

```
<div id="app">
    <addnum></addnum>
    <addnum></addnum>
    <addnum></addnum>
    <addnum></addnum>
    <addnum></addnum>
</div>
<script>
    Vue.component("addnum",{
        template:'<div><input type="text" v-model="num"><button @click="num+=1">点击
</button></div>',
        data: function(){
            // 写在这里的数据只有当前组件可以使用
            return {
                num:1,
            }
        }
    });


    var vm = new Vue({
        el:"#app",
        // 这里写的数据是全局公用的，整个文件共享
        data:{

        }
    })
</script>
```