1. 了解同源策略和跨域

1.1. 同源策略

1. 什么是同源

如果两个页面的**协议**，**域名**和**端口**都相同，则两个页面具有相同的源

例如下表给出了相对于http://www.test.com/index.html页面的同源检测：

URL	是否同源	原因
http://www.test.com/other.html	是	同源（协议、域名、端口相同）
https://www.test.com/about.html	否	协议不同（http 与 https）
http://blog.test.com/movie.html	否	域名不同（www.test.com 与 blog.test.com）
http://www.test.com:7001/home.html	否	端口不同（默认的 80 端口与 7001 端口）
http://www.test.com:80/main.html	是	同源（协议、域名、端口相同）

2. 什么是同源策略

同源策略（英文全称：Same origin policy）是**浏览器**提供的一个**安全功能**

MDN官方给定的概念：同源策略限制了从同一个源加载的文档或者脚本如何与来自另一个源的资源进行交互。这是一个用于隔离潜在恶意文件的重要安全机制

通俗理解：浏览器规定,A网站的JavaScript，不允许和非同源的网站C之间，进行资源的交互

例如：

- 无法读取非同源网页的cookie、LocalStorage和IndexedDB
- 无法接触非同源网页的DOM
- 无法向非同源地址发送ajax请求

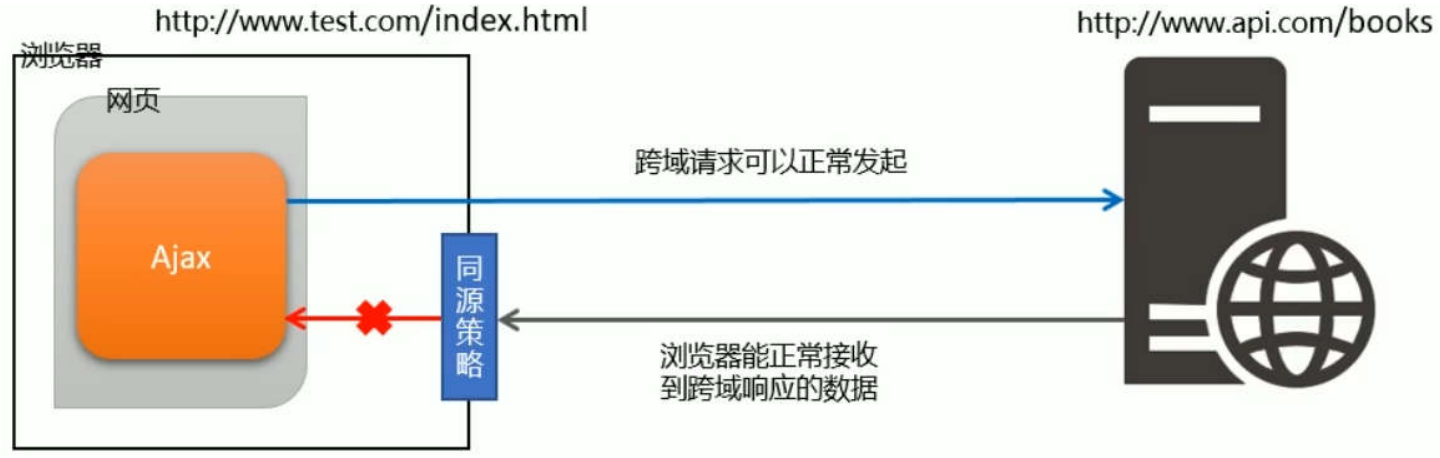
1.2. 跨域

1. 什么是跨域

同源指两个URL协议、域名、端口一致，反之就是跨域
（只要协议、域名、端口三项中有任何一项不一致就是跨域）

出现跨域的根本原因：**浏览器的同源策略**不允许非同源的URL之间进行资源的交互

2. 浏览器对跨域请求的拦截



浏览器允许发起跨域请求，但是，跨域请求回来的数据，会被浏览器拦截，无法被页面获取到

3. 如何实现跨域数据请求

两种解决方案：**JSONP**和**CORS**

JSONP:出现较早，兼容性好。是前端程序员为了解决跨域问题，被迫想出来的一种 **临时解决方案**。缺点是 **只支持get请求** ，不支持post请求

CORS：出现较晚，是W3C标准，属于跨域Ajax请求的根本解决方案。支持GET和POST请求。缺点是兼容性较差

2， JSONP

2.2. JSONP的实现原理

由于浏览器的同源策略的限制，网页中无法通过Ajax请求非同源的接口数据。但是<script>标签不受浏览器同源策略的影响，可以通过src属性，请求非同源的js脚本

因此，JSONP的实现原理，就是通过<script>标签的src属性，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

```
$.ajax({
    method: 'get',
    url: 'http://ajax.frontend.itheima.net:3006/api/jsonp',
    data: {
        name: 'zhang',
        age: 20
    },
    success: function (res) {
        console.log(res);
    }
})
```

2.3. 自己实现一个简单的JSONP

定义一个success回调函数

```
<script>
    function success(data){
        console.log('JSONP响应回来的数据是：');
        console.log(data);
    }
</script>

<script src="http://ajax/frontend.itheima.net:3006/api/jsonp?callback=abc"></script>
```

2.4.JSONP的缺点

由于JSONP是通过<script>标签的src属性，来实现跨域数据获取的，所以，JSONP只支持get数据请求，不支持POST数据请求

注意：JSONP和Ajax之间没有任何关系，不能把JSONP请求数据的方式叫做Ajax，因为JSONP没有用到XMLHttpRequest这个对象

2.5. jQuery中的JSONP

jQuery提供的$.ajax()函数，除了可以发起真正的Ajax请求之外，还能够发起JSONP数据请求

```
<script src="lib/jquery.js"></script>

<script>
    $(function () {
        // 发起JSONP请求
        $.ajax({
            url: 'http://ajax.frontend.itheima.net:3006/api/jsonp?name=zs&age=20',

            // 代表我们要发起JSONP的数据请求
            dataType: 'jsonp',
            success: function (res) {
                console.log(res);
            }
        })
    })
</script>
```

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，jQueryxxx是随机生成的一个回调函数名称

2.6. 自定义参数及回调函数名称

在使用jQuery发起JSONP请求时，如果想要自定义JSONP的参数及回调函数名称，可以通过如下两个参数来指定：

```
//发送到服务端的参数名称，默认值为callback
jsonp:'cb',

//自定义的回调函数名称，默认值为jQueryxxx格式
jsonpCallback:'abc',
```

2.7. jQuery中JSONP的实现过程

jQuery中的JSONP,也是通过<script>标签的src属性实现跨域数据访问的，只不过，jQuery采用的是动态创建和移除<script>标签的方式，来发起JSONP的数据请求

- 在发起JSONP请求的时候，动态向<header>中append一个<script>标签；
- 在JSONP请求成功之后，动态从<header>中移除刚才append进去的<script>标签

```
<script src="lib/jquery.js"></script>

<button id="btnJSONP">发起JSONP数据请求</button>
<script>
    $(function () {
        $('#btnJSONP').on('click', function () {
            $.ajax({
                url: 'http://ajax.frontend.itheima.net:3006/api/jsonp?address=北京&location=顺义
                dataType: 'jsonp',
                jsonpCallback: 'abc',
                success: function (res) {
                    console.log(res);
                }

            })
        })
    })
</script>
```

3. 案例-淘宝搜索

3.1. 要实现的UI效果

3.2.获取用户输入的搜索关键词

为了获取到用户每次按下键盘输入的内容，需要监听输入框的keyup事件

```
$(function () {
    //为输入框绑定keyup事件
    $('#ipt').on('keyup',function(){
    var keywords=$(this).val().trim()

    if (keywords.length<=0){
        reutrn
    }
    TODO://获取搜索建议列表
    console.log(keywords);
    })
})
```

3.3. 封装getSuggestList函数

将获取搜索建议列表的代码，封装到getSuggestList函数中

```
function getSuggestList(kw) {
$.ajax({
    url: 'http://suggest.taobao.com/sug?q=' + kw,
    dataType: 'jsonp',
    success: function (res) {
    console.log(res);
    }
})
}
```

3.4. 渲染建议列表的UI结构

1. 定义搜索建议列表

```
<!-- 搜索建议列表 -->
<div id="suggest-list"></div>
```

2. 定义模板结构

```
<!-- 模板结构 -->

<script type="text/html" id="tpl-suggestList">
  {{each}}
  <!-- 搜索建议项 -->
  <div class="suggest-item">{{$value[0]}}</div>
  {{/each}}
</script>
```

3. 定义渲染模板结构的函数

```
// 渲染UI结构
function renderSuggestList(res) {
if (res.result.length <= 0) {
    return $('#suggest-list').empty().hide()
}
var htmlStr = template('tpl-suggestList', res)
$('#suggest-list').html(htmlStr).show()
}
})
```

4. 搜索关键词为空时隐藏搜索建议列表

在keyup事件中判断输入关键字符串长度是否为0中加入：

```
return $('#suggest-list').empty().hide()
```

5. 搜索建议列表美化

```
#suggest-list {
  border:1px solid #ccc;
  display: none;
}

.suggest-item{
  line-height: 30px;
  padding-left: 5px;
}

.suggest-item:hover{
  cursor:pointer;
  background-color: #eee;
}
```

3.5. 输入框的防抖

1. 什么是防抖

防抖策略：（debounce）是当事件被触发后，**延迟n秒**后再执行回调，如果在这n秒内事件又被触发，则重新计时

例子：王者荣耀游戏回城操作读条

2. 防抖的应用场景

用户在输入框中连续输入一串字符时，可以通过防抖策略，在输入完后，才执行查询的请求，这样可以有效减少请求次数，节约请求资源

3. 实现输入框的防抖

```
// 定义延时器的id
var timer = null

// 定义防抖的函数
function debounceSearch(kw) {
timer = setTimeout(function () {
    getSuggestList(kw)
}, 500)
}
...
clearTimeout(timer)
...
debounceSearch(keywords)
```

3.6. 缓存搜索的建议列表

1. 定义全局缓存对象

```
// 定义缓存对象
var cacheObj={}
...
// 在渲染UI结构时，获取到用户输入的数据当做键
var k=$('#ipt').val().trim()

// 在渲染UI结构时，需要将数据作为值，进行缓存
cacheObj[k]=res
...
// 当触发keyup的时候，先判断缓存中是否有数据
if (cacheObj[keywords]){
    return renderSuggestList(cacheObj[keywords])
}
```

4. 防抖和节流

4.1. 什么是节流

节流策略（throttle），顾名思义，可以减少一段时间内事件的触发频率

4.2.节流的应用场景

- 鼠标不断地触发某事件（比如点击），只在单位时间内触发一次
- 懒加载时要监听计算滚动条的位置，但不必每次滑动都触发，可以降低计算的频率

4.3. 节流案例-鼠标跟随效果

1. 渲染UI结构并美化样式

```
<style>
    html,body{
        margin: 0;
        padding: 0;
        overflow: hidden;
    }

    #angel{
        position: absolute;
    }
</style>

<img src="angel.gif" alt="" id="angel" />
```

2. 不使用节流实现鼠标跟随效果

```
<script>
    $(function () {
        // 1. 获取到图片
        var angel = $('#angel')

        // 2. 绑定mousemove事件
        $(document).on('mousemove', function (e) {
            // console.log(e.pageX);
            // console.log(e.pageY);

            // 3. 设置图片的位置
            $(angel).css('top', e.pageY + 'px').css('left', e.pageX + 'px')
        })
    })
</script>
```

3. 节流阀的概念

如果节流阀为空，表示可以执行下次操作；不为空，表示不能执行下次操作

当前操作执行完毕，必须将节流阀重置为空，表示可以执行下次操作了

每次执行操作前，必须先判断节流阀是否为空

4. 使用节流阀优化鼠标跟随效果

```
// a. 定义节流阀
var timer = null

// c. 判断节流阀是否为空
if (timer) {
    return
}

// b. 开启延时器
timer = setTimeout(function () {
    // 3. 设置图片的位置
    $(angel).css('top', e.pageY + 'px').css('left', e.pageX + 'px')

    timer = null
}, 16)
```

4. 防抖和节流的区别

- 防抖：如果事件被频繁触发，防抖能保证 **只有最后一次触发生效**，前面N多次的触发都会被忽略
- 节流：如果事件被频繁触发，节流能够减少事件触发的频率，因此，节流是有选择性地执行一部分事件