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** 页面同源检测 不设置端口默认为：80

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 （英文全程 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。

通俗的理解：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：

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

1.2跨域

1.什么是 跨域

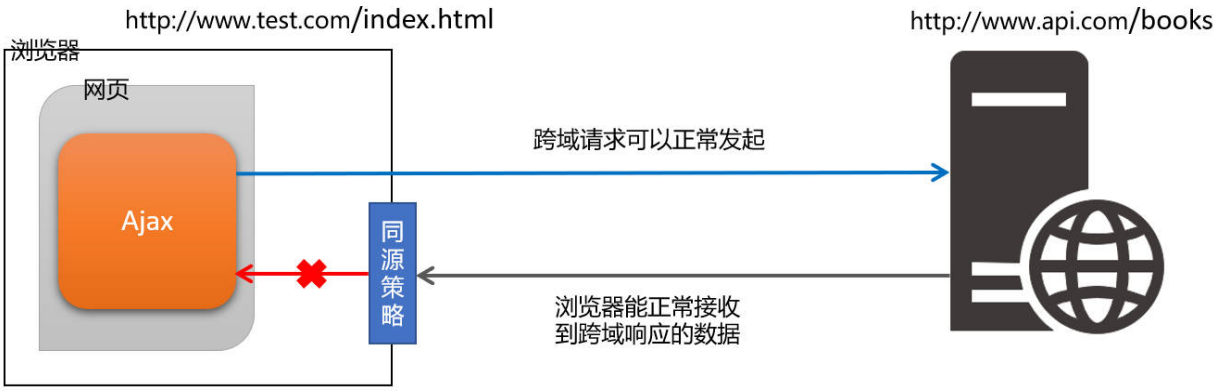
同源 指的是两个URL 的协议、域名、端口一致，反之，则是 **跨域** 。

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。

网页： **http://www.test.com/index.html**

接口： **http://www.api.com/userlist**

2.浏览器对跨域请求的拦截



> 注意：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

> 现如今，实现跨域数据请求，最主要的两种解决方案，分别是 **JSONP** 和 **CORS** 。

JSONP: 出现的早，兼容性好(兼容低版本IE)。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：出现的较晚，他是W3C标准，属于跨域Ajax的请求的根本解决方案。支持GET和POST请求。缺点是不兼容某些低版本的浏览器。

2.JSONP

> JSONP(JSON with Padding) 是JSON的一种“使用模式”，可用于解决主流浏览器的跨域数据访问的问题。

2.2JSONP的实现原理

由于 **浏览器同源策略** 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。

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('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://www.liulongbin.top:3006/api/jsonp?callback=success&name=zs&age=20">
</script>
```

2.4JSONP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。

2.5jQuery中的JSONP

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

```
$.ajax({
    url:'http://www.liulongbin.top:3006/api/jsonp?name=zs&age=20',
    // 如果要使用 $.ajax() 发起 JSONP 请求，必须指定 dataType的属性 为 jsonp
    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，**jQueryxxx**是随机生成的一个回调函数的名称。

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 以及 回调函数的名称 ，可以通过以下下两个参数来指定：

```
$.ajax({
    url: 'http://www.liulongbin.top:3006/api/jsonp',
    dataType = 'jsonp',
    // 发送到服务端的参数名称，默认值为 callback
    jsonp:'callback',
    // 自定义的函调函数名称，默认值为 jQueryxxx 格式
    jsonpCallback: 'abc',
    success:function(res) {
    console.log(res)
   }
})
```

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 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 以后，动态从 `<header>` 中移除刚才 append 进去的 `<script>` 标签

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，实例如下：

```
// 监听文本框的 keyup 事件
$('#ipt').on('click',function() {
    // 获得用户输入的内容
    let keywords = $(this).val().trim()
    // 判断用户输入的内容是否为空
    if(keywords.length <= 0) {
       return
    }
    // TODO: 获取搜索建议列表
})
```

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 getSuggsetList(kw) {
    $.ajax({
        // 请求的 URL 地址，其中 q 是用户输入的关键字
        url:'https://suggest.taobao.com/sug?q=' + kw,
        // 指定要发起的是 JSONP 请求
        dataType: 'jsonp',
        // 成功的回调函数
        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

```
<div class="box">
    <!-- tab 栏区域 -->
    <div class="tabs"></div>
    <!-- 搜索区域 -->
    <div class="search-box"></div>

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

2.定义模板结构

```
<!-- 模板结构 -->
<scirpt src="text/html" id='tpl-suggestList'>
{{each result}}
   <div class="suggest-item">{{$value[0]}}</div>
{{/each}}

</script>
```

3.定义渲染模板结构的函数

```
// 渲染建议列表
function renderSuggestList(res) {
  // 如果没有需要渲染的数据，则直接 return
    if(res.result.length <= 0) {
        return $('#suggest-list').empty.hide()
    }
    // 渲染模板结构
    let htmlstr = template('suggest-list',res)
    $('#suggest-list').html(htmlstr).show()
}
```

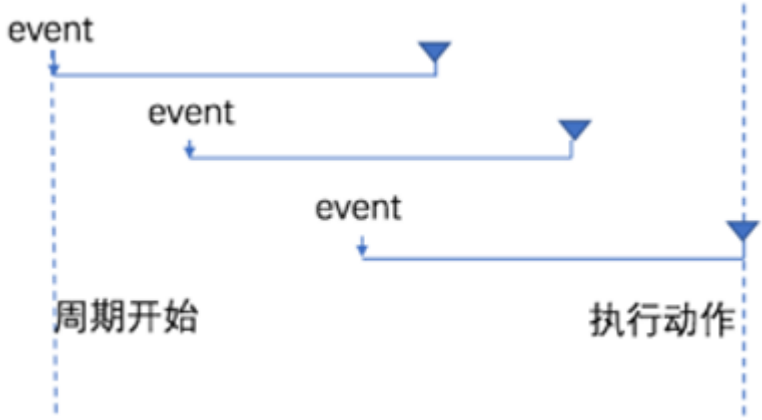
4.搜索关键词为空时隐藏搜索建议列表

```
$('#ipt').on('click',function() {
    // 获取用户输入的内容
    let keywords = $(this).val().trim()
    // 判断用户输入的内容是否为空
    if (keywords.length <= 0) {
        // 如果关键字为空，则清空后隐藏搜索建议列表
        return $('##suggest-list').empty().hide()
    }
    getSuggestList(keywords)
})
```

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。



3.实现输入框的防抖

```
let timer = null     // 1. 防抖的 timer

function debounceSearch(keywords) {   //2.定义防抖的函数
 timer = settimeout(function() {
     // 调用 jsonp 请求数据函数
     timer = getSuggestList(kw)
 },500)
}

$('#ipt').on('click',funcition() {    //3. 在触发 keyup 事件时 直接清除timer
    clearTimeout(timer)
    // ....
    debounceSearch(keywords)
 })
```

3.6 缓存搜索的建议列表

1.定义全局的缓存对象

```
// 缓存对象
let cacheObj = {}
```

2.将搜索结果保存到缓存对象中

```
// 渲染数据列表
function renderSuggestList(res) {
    // ....

    // 将搜索的结果，添加到缓存对象中
    let k = $('#ipt').val().trim()
    cacheObj[k] = res
}
```

4.优先从缓存中获取搜索建议

```
// 监听文本框的 keyup 事件
 $('#ipt').on('keyup', function() {
    // ...省略其他代码

    // 优先从缓存中获取搜索建议
    if (cacheObj[keywords]) {
       return renderSuggestList(cacheObj[keywords])
    }
    // 获取搜索建议列表
    debounceSearch(keywords)
  })
```

4.防抖和节流

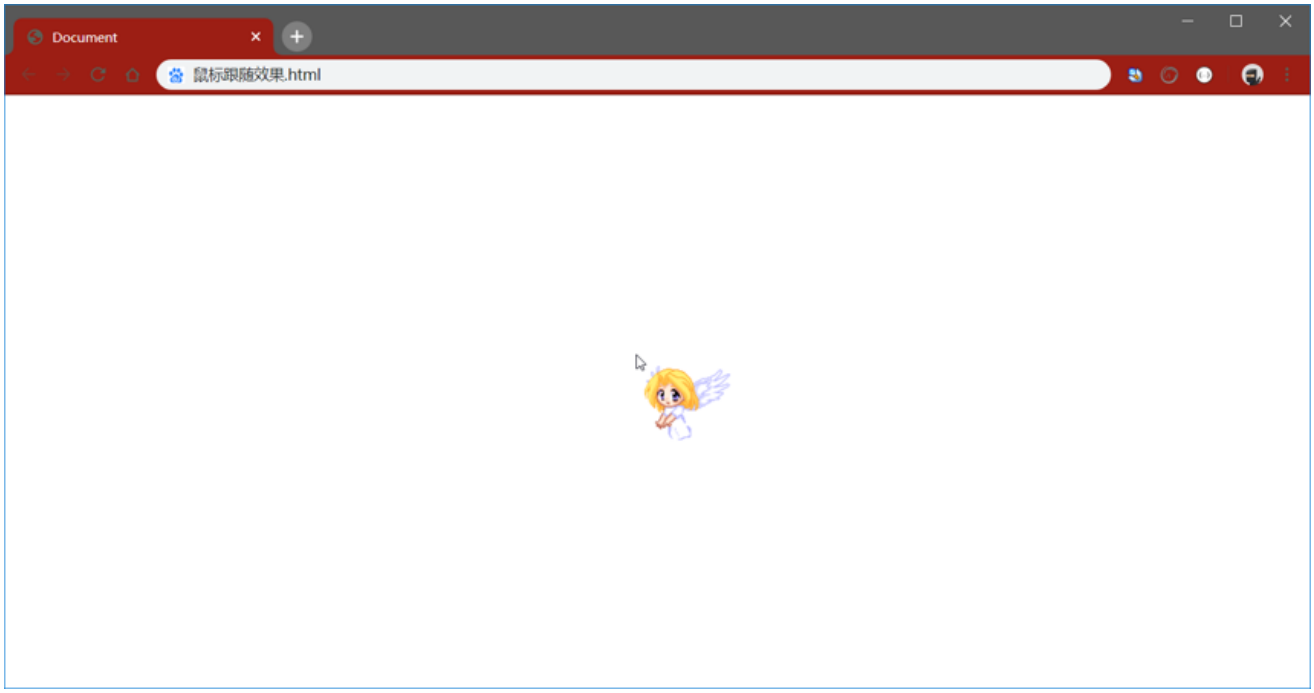
4.1什么是节流

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

> 1.鼠标连续不断地触发某事件(如点击), 只在单位事件内触发一次； 2.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，而不必去浪费cpu资源

4.3节流案例 — 鼠标跟随效果



1. 渲染 UI 结构并美化样式

```
<!-- UI 结构 -->
<img src="./assets/angel.gif" alt="" id="angel" />

/* CSS 样式 */
html, body {
  margin: 0;
  padding: 0;
  overflow: hidden;
}
#angel {
  position: absolute;
}
```

2.不使用节流实现鼠标跟随效果

```
$(function() {
    // 获取图片元素
    var angel = $('#angel')
    // 监听文档的 mousemove 事件
    $(document).on('mousemove', function(e) {      // 设置图片的位置
        $(angel).css('left', e.pageX + 'px').css('top', e.pageY + 'px')
    })
})
```

3. 节流阀 的概念

高铁卫生间是否被占用，有红绿灯控制， **红灯** 表示 **被占用** ， **绿灯** 表示 **可使用** 。

假设每个人上卫生间都需要 **花费5分钟** ，则 **五分钟之内** ，被占用的卫生间无法被其他人使用。

上一个人使用完毕后，需要将红灯 **重置** 为绿灯，表示下一个人可以使用卫生间。

下一个人在上卫生间之前，需要 **先判断控制灯** 是否为绿色，来知晓能否上卫生间。

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，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.使用节流优化鼠标跟随效果

```
$(function() {
    let angel = $('#angel')
    let timer = null   // 1.定义一个节流阀
    $(document).on('mouseover',function(e) {
        if (timer) return   // 3.判断节流阀是否为空，如果不为空，则证明距离上次执行间隔不足16毫秒
        timer = setTimeout(function(){
            $(angel).css('left',e.pageX + 'px').css('top',e.pageY + 'px')
             console.log('ok')
             timer = null // 2. 当鼠标设置了跟随效果后，清空 timer 节流阀 方便下次开启定时器
        },16)
    })
})
```

4.4 总结防抖和节流的区别

- 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 **有选择性的执行** 一部分事件！