

录Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

1. 客户端与服务器

1.1 上网的目的



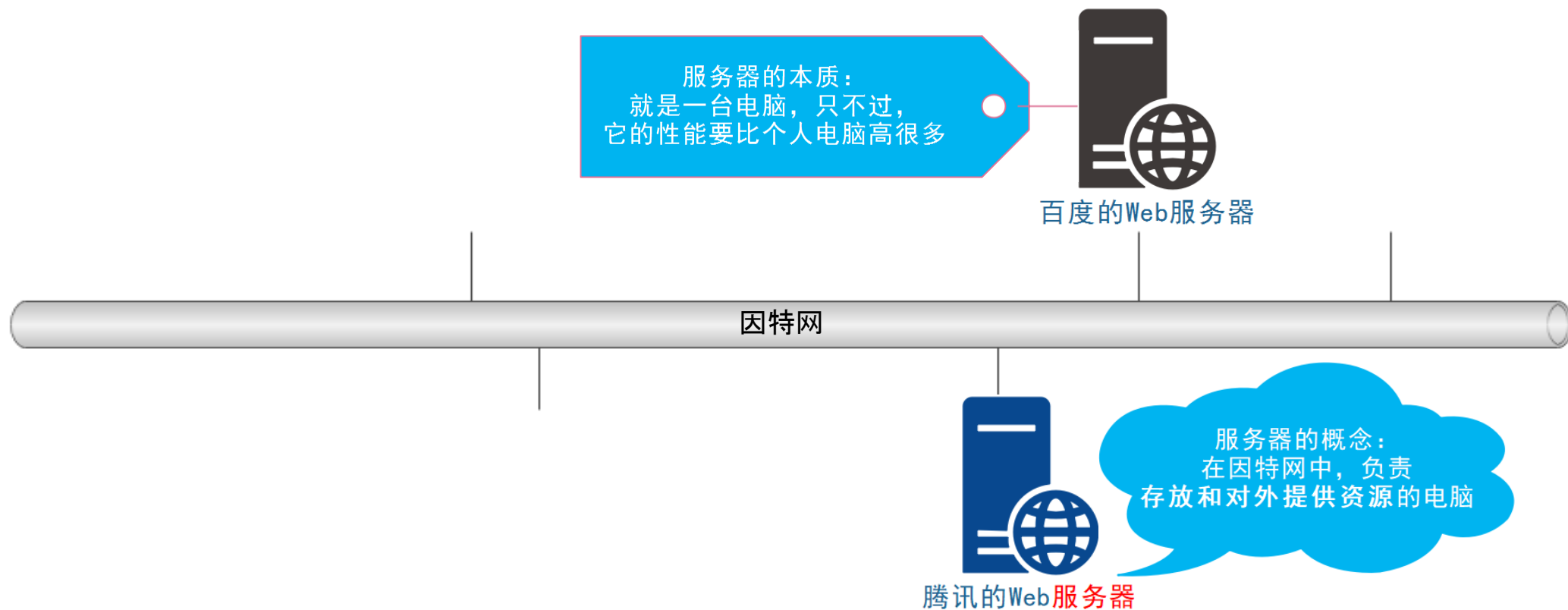
- 刷微博
- 浏览新闻
- 在线听音乐
- 在线看电影
- etc...

上网的**本质目的**：通过互联网的形式来**获取和消费资源**

1. 客户端与服务器

1.2 服务器

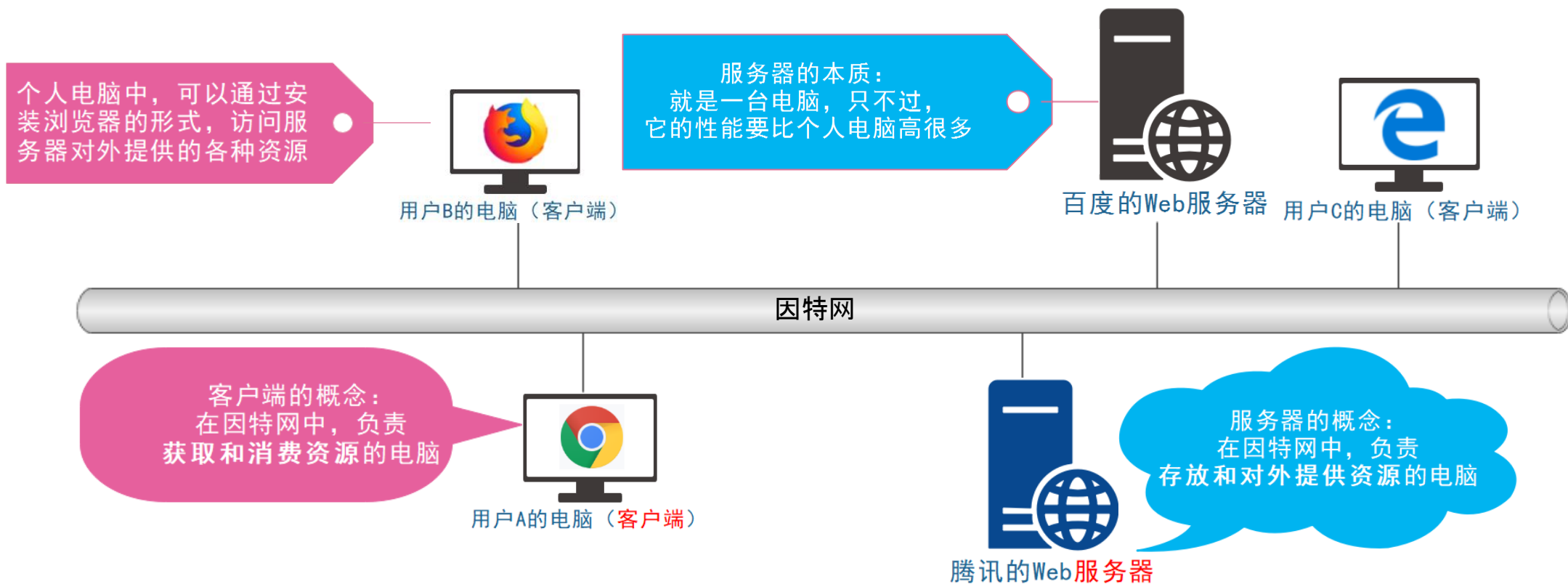
上网过程中，负责**存放和对外提供资源**的电脑，叫做服务器。



1. 客户端与服务器

1.3 客户端

上网过程中，负责**获取和消费资源**的电脑，叫做客户端。



目录

Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

2. URL地址

2.1 URL地址的概念

URL（全称是UniformResourceLocator）中文叫**统一资源定位符**，用于标识互联网上每个资源的唯一存放位置。浏览器只有通过URL地址，才能正确定位资源的存放位置，从而成功访问到对应的资源。

常见的URL举例：

`http://www.baidu.com`

`http://www.taobao.com`

`http://www.cnblogs.com/liulongbinblogs/p/11649393.html`


2. URL地址

2.2 URL地址的组成部分

URL地址一般由三部组成：


- ① 客户端与服务端之间的通信协议
- ② 存有该资源的服务器名称
- ③ 资源在服务器上具体的存放位置

<http://www.cnblogs.com/liulongbinblogs/p/11649393.html>



The diagram illustrates the three components of the URL `http://www.cnblogs.com/liulongbinblogs/p/11649393.html` using brackets and labels below the text:

- 通信协议** (Communication Protocol): Points to the `http` part of the URL.
- 服务器名称** (Server Name): Points to the `www.cnblogs.com` part of the URL.
- 资源在服务器上具体的存放位置** (Specific location of the resource on the server): Points to the `/liulongbinblogs/p/11649393.html` part of the URL.



录Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

3. 客户端与服务器的通信过程

3.1 图解客户端与服务器的通信过程

http://www.baidu.com



用户的电脑 (客户端)

1. 客户端请求服务器



web服务器

2. 服务器处理这次请求

3. 服务器响应客户端

1. 打开浏览器
2. 输入要访问的网站地址
3. 回车，向服务器发起资源请求

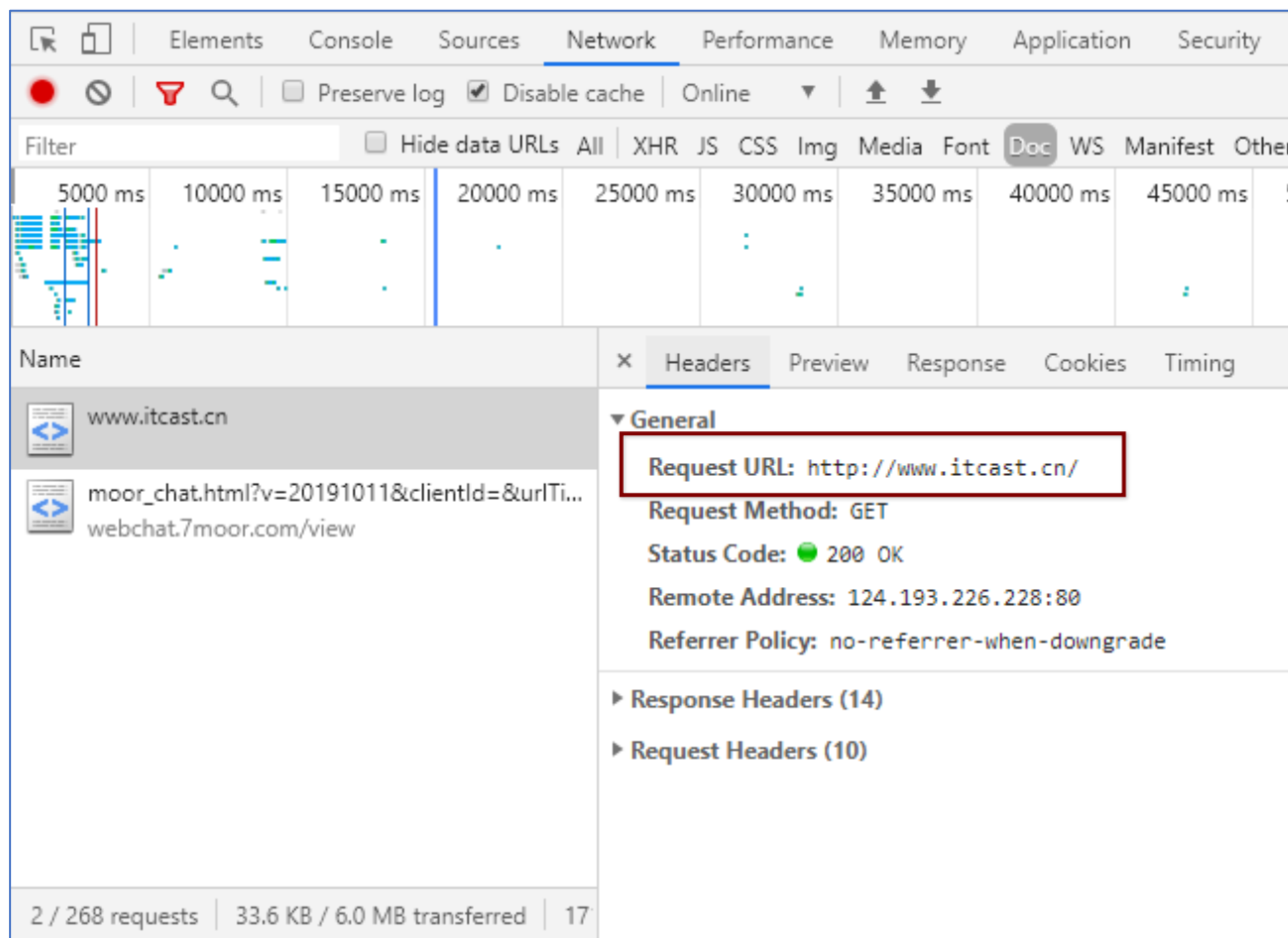
1. 服务器接收到客户端发来的资源请求
2. 服务器在内部处理这次请求，找到相关的资源
3. 服务器把找到的资源，响应（发送）给客户端

注意：

- ① 客户端与服务器的通信过程，分为 请求 - 处理 - 响应 三个步骤。
- ② 网页中的每一个资源，都是通过 请求 - 处理 - 响应 的方式从服务器获取回来的。

3. 客户端与服务器的通信过程

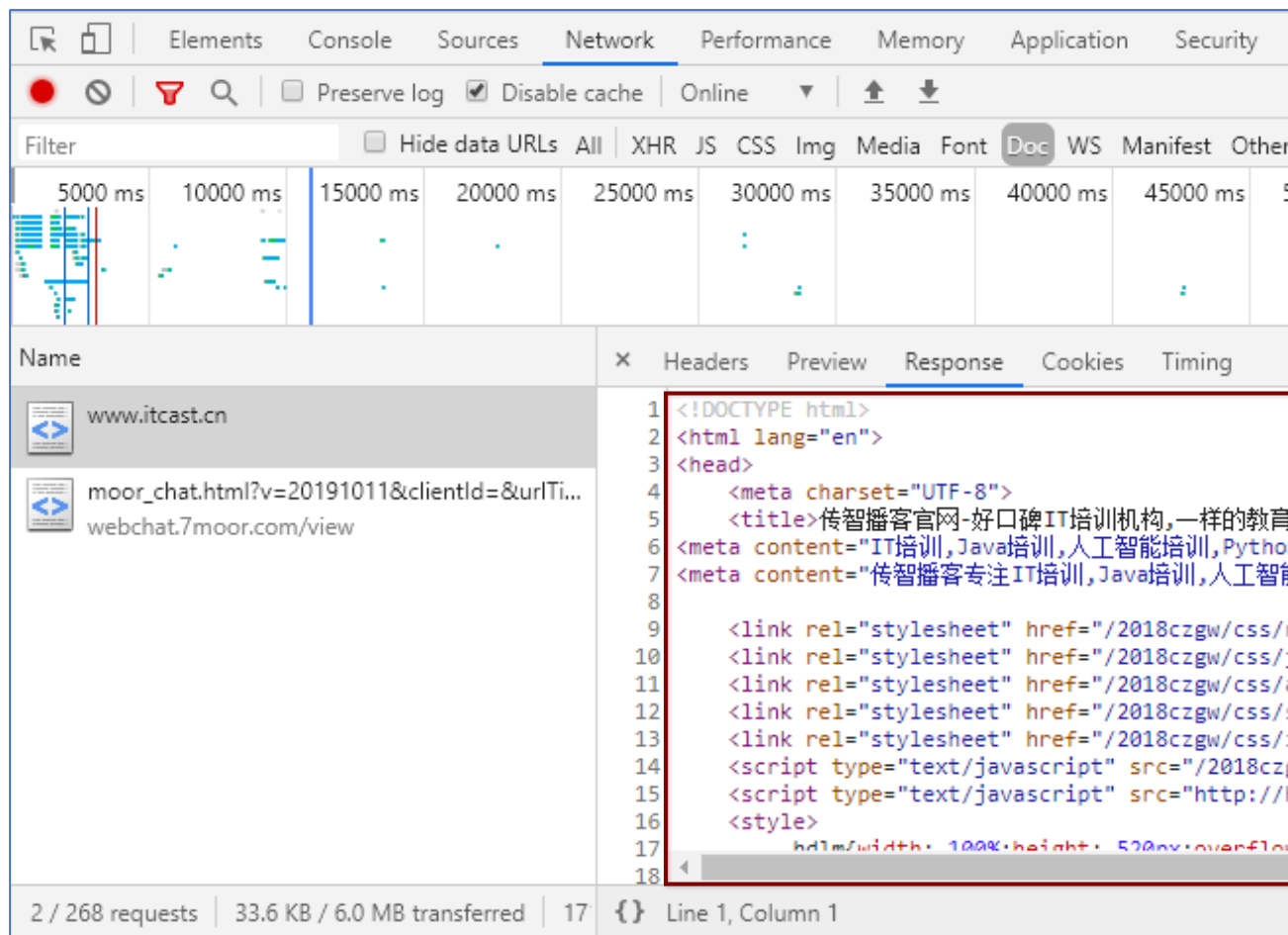
3.2 基于浏览器的开发者工具分析通信过程




1. 打开 Chrome 浏览器
2. Ctrl+Shift+I 打开 Chrome 的开发者工具
3. 切换到 Network 面板
4. 选中 Doc 页签
5. 刷新页面，分析客户端与服务器的通信过程

3.客户端与服务器的通信过程

3.2 基于浏览器的开发者工具分析通信过程



1. 打开 Chrome 浏览器
2. Ctrl+Shift+I 打开 Chrome 的开发者工具
3. 切换到 Network 面板
4. 选中 Doc 页签
5. 刷新页面，分析客户端与服务器的通信过程



录Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

4. 服务器对外提供了哪些资源

4.1 例举网页中常见的资源



文字内容



Image 图片



Audio 音频



Video 视频

and so on...

思考：网页中的数据是不是资源？

4. 服务器对外提供了哪些资源

4.2 数据也是资源

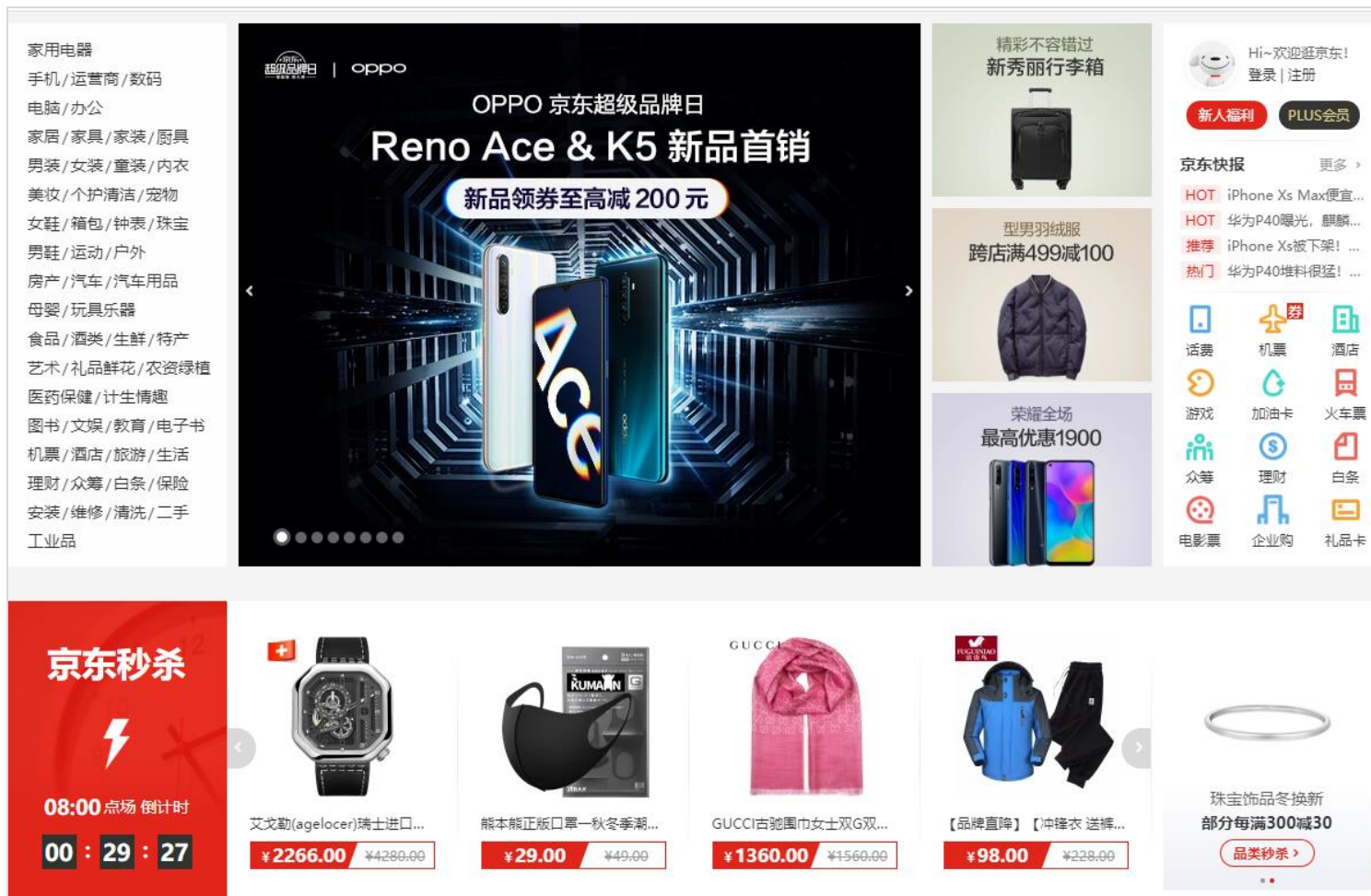
网页中的数据，也是服务器对外提供的一种资源。例如股票数据、各行业排行榜等。



手机行业排行 MOBILE PHONE	
搜索指数排行	资讯指数排行
1 IPHONE	10,078k —
2 华为	5,296k —
3 小米手机	3,128k —
4 OPPO	2,493k —
5 VIVO	1,968k —
更多手机行业榜单 >	

4. 服务器对外提供了哪些资源

4.3 数据是网页的灵魂



- **HTML**是网页的**骨架**
- **CSS**是网页的**颜值**
- **Javascript**是网页的**行为**
- **数据**，则是网页的**灵魂**

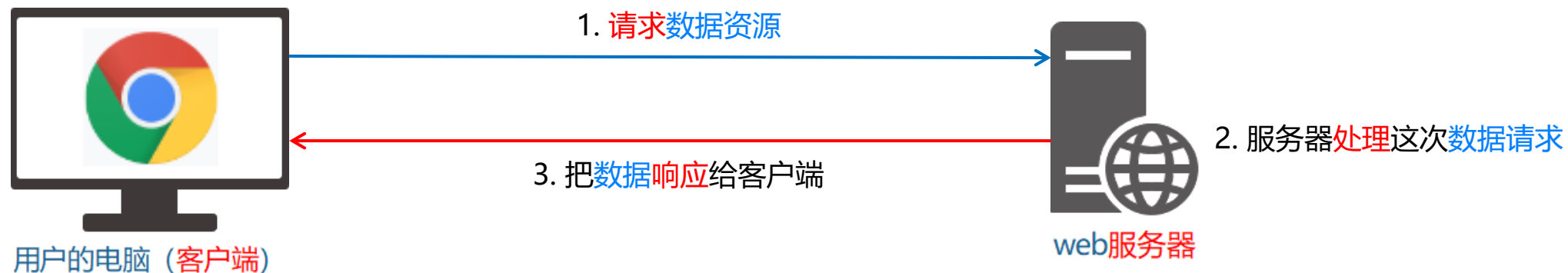
骨架、颜值、行为**皆为数据服务**

数据，在网页中**无处不在**

4. 服务器对外提供了哪些资源

4.4 网页中如何请求数据

数据，也是服务器对外提供的一种资源。只要是资源，必然要通过 请求 - 处理 - 响应 的方式进行获取。



如果要在网页中请求服务器上的数据资源，则需要用到 XMLHttpRequest 对象。

XMLHttpRequest (简称 xhr) 是浏览器提供的 js 成员，通过它，可以请求服务器上的数据资源。

最简单的用法 `var xhrObj = new XMLHttpRequest()`

4. 服务器对外提供了哪些资源

4.5 资源的请求方式


客户端请求服务器时，请求的方式有很多种，最常见的两种请求方式分别为 `get` 和 `post` 请求。

- `get` 请求通常用于获取服务端资源（向服务器要资源）

例如：根据 URL 地址，从服务器获取 HTML 文件、css 文件、js 文件、图片文件、数据资源等

- `post` 请求通常用于向服务器提交数据（往服务器发送资源）

例如：登录时向服务器提交的登录信息、注册时向服务器提交的注册信息、添加用户时向服务器提交的用户信息等各种数据提交操作



录Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

5. 了解Ajax

5.1 什么是Ajax

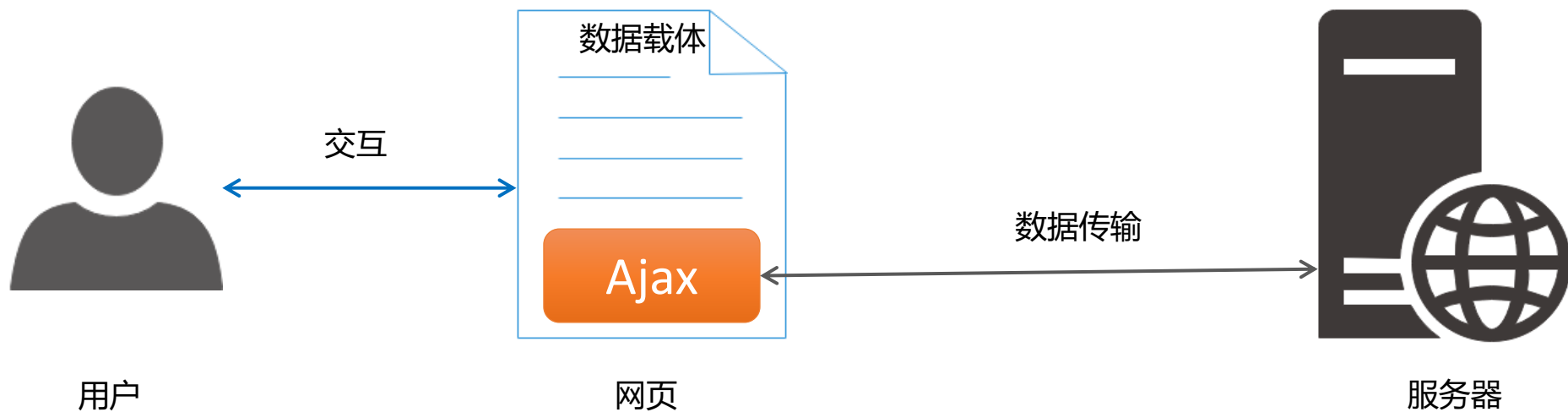
Ajax 的全称是 Asynchronous Javascript And XML（异步 JavaScript 和 XML）。

通俗的理解：在网页中利用 XMLHttpRequest 对象和服务端进行数据交互的方式，就是Ajax。

5. 了解Ajax

5.2 为什么要学Ajax

之前所学的技术，只能把网页做的更美观漂亮，或添加一些动画效果，但是，**Ajax**能让我们轻松实现网页与服务器的数据交互。



5. 了解Ajax

5.3 Ajax的典型应用场景

用户名检测：注册用户时，通过 ajax 的形式，动态检测用户名是否被占用

登录 · 注册

张三

手机号

设置密码

注册

点击“注册”即表示您同意并愿意遵守简书
[用户协议](#) 和 [隐私政策](#)。


社交帐号直接注册

微信 QQ

5. 了解Ajax

5.3 Ajax的典型应用场景

搜索提示：当输入搜索关键字时，通过 ajax 的形式，动态加载搜索提示列表

 黑马程序员™
www.itheima.com

搜索

ajax请求五个步骤

ajax同步和异步的区别

ajax是什么

ajax原理

反馈

5. 了解Ajax

5.3 Ajax的典型应用场景

数据分页显示：当点击页码值的时候，通过 ajax 的形式，根据页码值动态刷新表格的数据

#	姓名	邮箱	电话	角色
1	admin	1111222@qq.com	18170873540	超级管理员
2	zs	111@qq.com	13888888888	asdasdasd

共 8 条

2条/页

<

1

2

3

4

>

前往

1

页


5. 了解Ajax

5.3 Ajax的典型应用场景

数据的增删改查：数据的添加、删除、修改、查询操作，都需要通过 ajax 的形式，来实现数据的交互

添加分类

#	分类名称	是否有效	排序	操作
1	⊕ 大家电	✓	一级	<div>✎ 编辑</div> <div>🗑 删除</div>
2	⊕ 热门推荐	✓	一级	<div>✎ 编辑</div> <div>🗑 删除</div>
3	⊕ 海外购	✓	一级	<div>✎ 编辑</div> <div>🗑 删除</div>
4	⊕ 苏宁房产	✓	一级	<div>✎ 编辑</div> <div>🗑 删除</div>
5	⊕ 手机相机	✓	一级	<div>✎ 编辑</div> <div>🗑 删除</div>



录Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

6. jQuery中的Ajax

6.1 了解jQuery中的Ajax

浏览器中提供的 XMLHttpRequest 用法比较复杂，所以 jQuery 对 XMLHttpRequest 进行了封装，提供了一系列 Ajax 相关的函数，极大地降低了 Ajax 的使用难度。

jQuery 中发起 Ajax 请求最常用的三个方法如下：

- \$.get()
- \$.post()
- \$.ajax()

6. jQuery中的Ajax

6.2 \$.get()函数的语法

jQuery 中 \$.get() 函数的功能单一，专门用来发起 get 请求，从而将服务器上的资源请求到客户端来进行使用。

\$.get() 函数的语法如下：

```
$.get(url, [data], [callback])
```

其中，三个参数各自代表的含义如下：

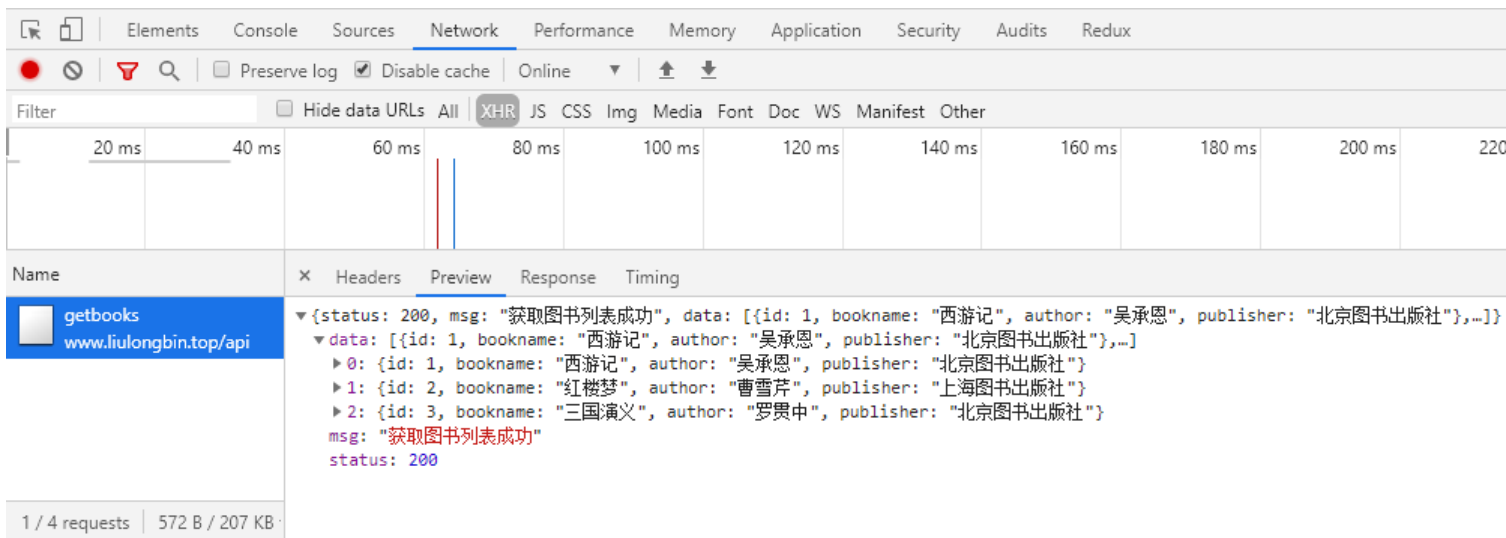
参数名	参数类型	是否必选	说明
url	string	是	要请求的资源地址
data	object	否	请求资源期间要携带的参数
callback	function	否	请求成功时的回调函数

6. jQuery中的Ajax

6.2 \$.get()发起不带参数的请求

使用 \$.get() 函数发起不带参数的请求时，直接提供请求的 URL 地址和请求成功之后的回调函数即可，示例代码如下：

```
$.get('http://www.liulongbin.top:3006/api/getbooks', function(res) {  
    console.log(res) // 这里的 res 是服务器返回的数据  
})
```

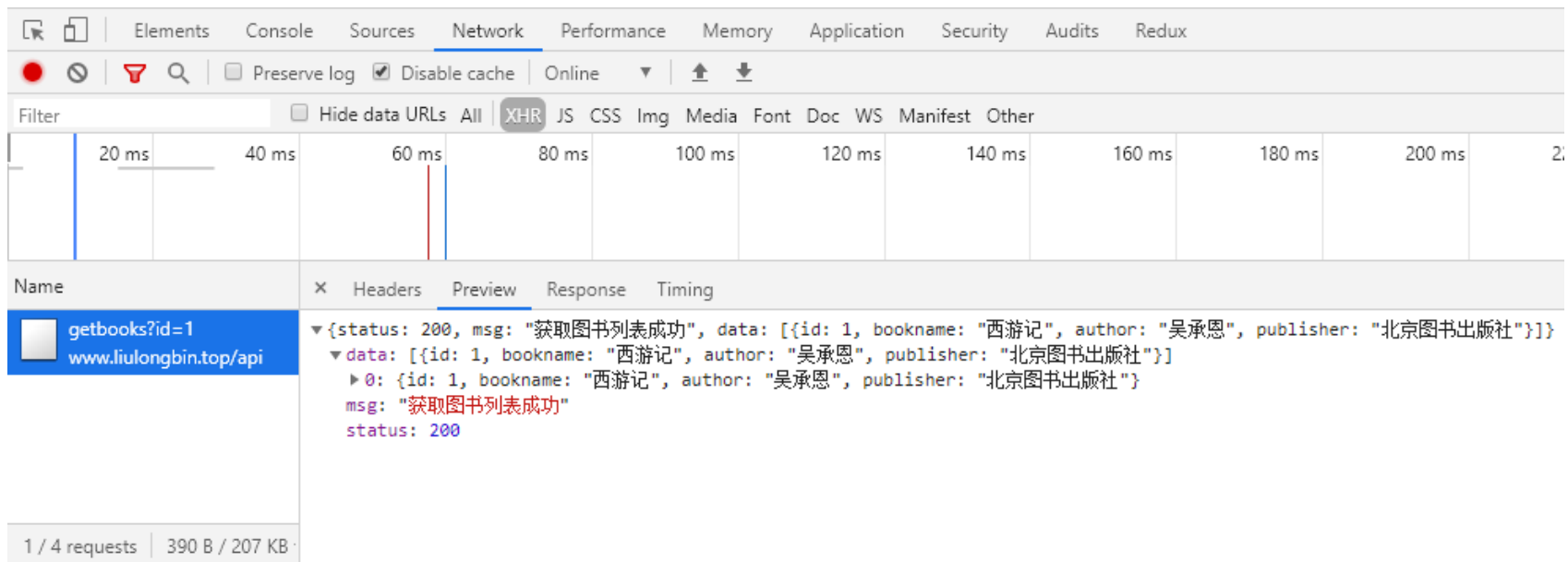


6. jQuery中的Ajax

6.2 \$.get()发起带参数的请求

使用 \$.get() 函数发起带参数的请求时，示例代码如下：

```
$.get('http://www.liulongbin.top:3006/api/getbooks', { id: 1 }, function(res) {  
    console.log(res)  
})
```



6. jQuery中的Ajax

6.3 \$.post()函数的语法

jQuery 中 \$.post() 函数的功能单一，专门用来发起 post 请求，从而向服务器提交数据。

\$.post() 函数的语法如下：

```
$.post(url, [data], [callback])
```

其中，三个参数各自代表的含义如下：

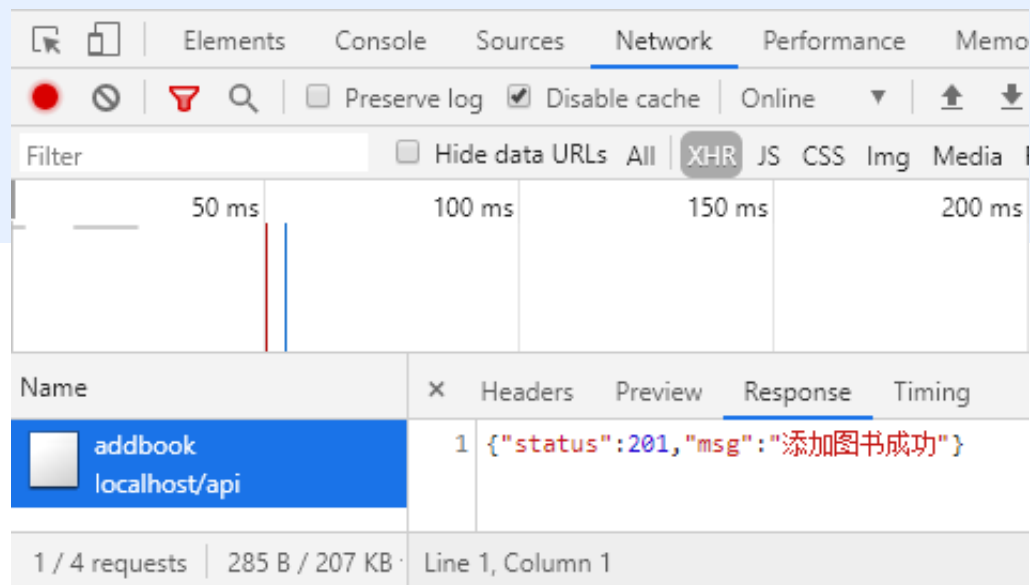
参数名	参数类型	是否必选	说明
url	string	是	提交数据的地址
data	object	否	要提交的数据
callback	function	否	数据提交成功时的回调函数

6. jQuery中的Ajax

6.3 \$.post()向服务器提交数据

使用 \$.post() 向服务器提交数据的示例代码如下：

```
$.post(  
    'http://www.liulongbin.top:3006/api/addbook', // 请求的URL地址  
    { bookname: '水浒传', author: '施耐庵', publisher: '上海图书出版社' }, // 提交的数据  
    function(res) { // 回调函数  
        console.log(res)  
    }  
)
```



6. jQuery中的Ajax

6.4 \$.ajax()函数的语法

相比于 \$.get() 和 \$.post() 函数，jQuery 中提供的 \$.ajax() 函数，是一个功能比较综合的函数，它允许我们对 Ajax 请求进行更详细的配置。

\$.ajax() 函数的基本语法如下：

```
$.ajax({  
    type: '', // 请求的方式，例如 GET 或 POST  
    url: '', // 请求的 URL 地址  
    data: { }, // 这次请求要携带的数据  
    success: function(res) { } // 请求成功之后的回调函数  
})
```


6. jQuery中的Ajax

6.4 使用\$.ajax()发起GET请求

使用 \$.ajax() 发起 GET 请求时，只需要将 **type 属性** 的值设置为 '**GET**' 即可：


```
$.ajax({  
    type: 'GET', // 请求的方式  
    url: 'http://www.liulongbin.top:3006/api/getbooks', // 请求的 URL 地址  
    data: { id: 1 }, // 这次请求要携带的数据  
    success: function(res) { // 请求成功之后的回调函数  
        console.log(res)  
    }  
})
```

6. jQuery中的Ajax

6.4 使用\$.ajax()发起POST请求

使用 \$.ajax() 发起 POST 请求时，只需要将 **type 属性** 的值设置为 '**POST**' 即可：

```
$.ajax({
  type: 'POST', // 请求的方式
  url: 'http://www.liulongbin.top:3006/api/addbook', // 请求的 URL 地址
  data: { // 要提交给服务器的数据
    bookname: '水浒传',
    author: '施耐庵',
    publisher: '上海图书出版社'
  },
  success: function(res) { // 请求成功之后的回调函数
    console.log(res)
  }
})
```



录Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

7. 接口

7.1 接口的概念

使用 Ajax 请求数据时，**被请求的 URL 地址**，就叫做**数据接口**（简称**接口**）。同时，每个接口必须有**请求方式**。

例如：

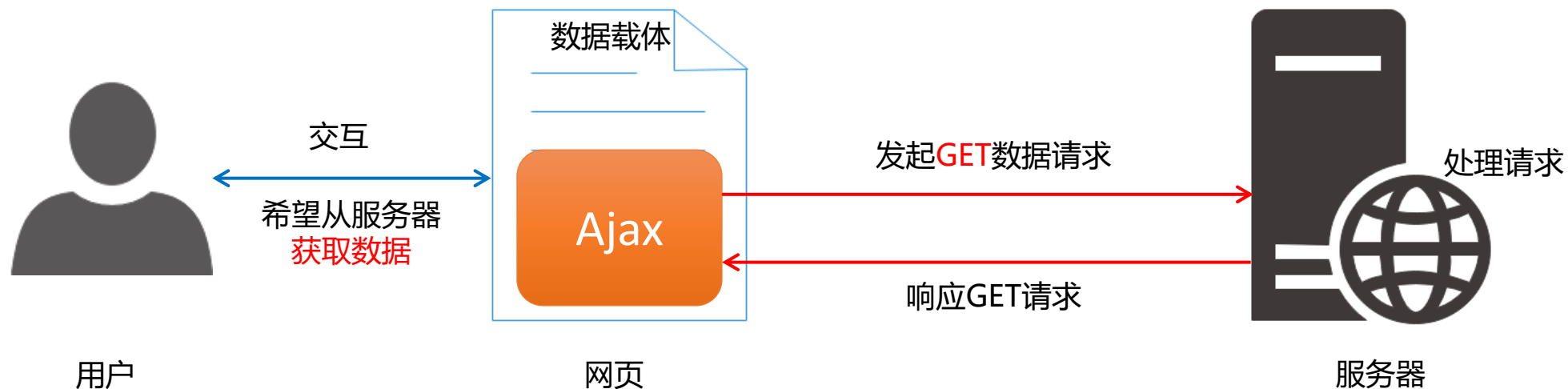
`http://www.liulongbin.top:3006/api/getbooks` 获取图书列表的接口 (GET请求)

`http://www.liulongbin.top:3006/api/addbook` 添加图书的接口 (POST请求)

7. 接口

7.2 分析接口的请求过程

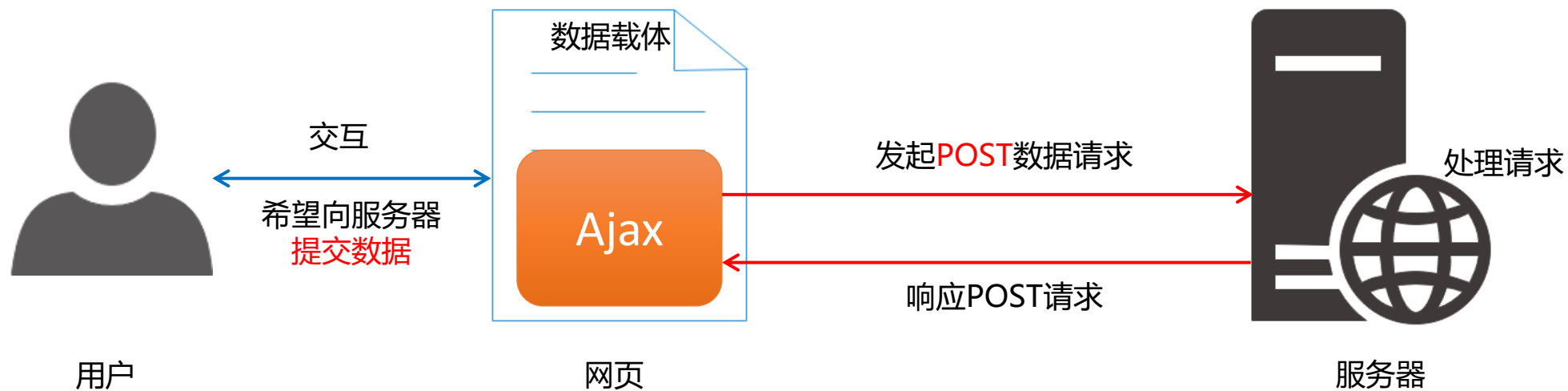
1. 通过GET方式请求接口的过程



7. 接口

7.2 分析接口的请求过程

2. 通过POST方式请求接口的过程



7. 接口

7.3 接口测试工具

1. 什么是接口测试工具

为了验证接口能否被正常被访问，我们常常需要使用接口测试工具，来对数据接口进行检测。

好处：接口测试工具能让我们在**不写任何代码**的情况下，对接口进行**调用和测试**。



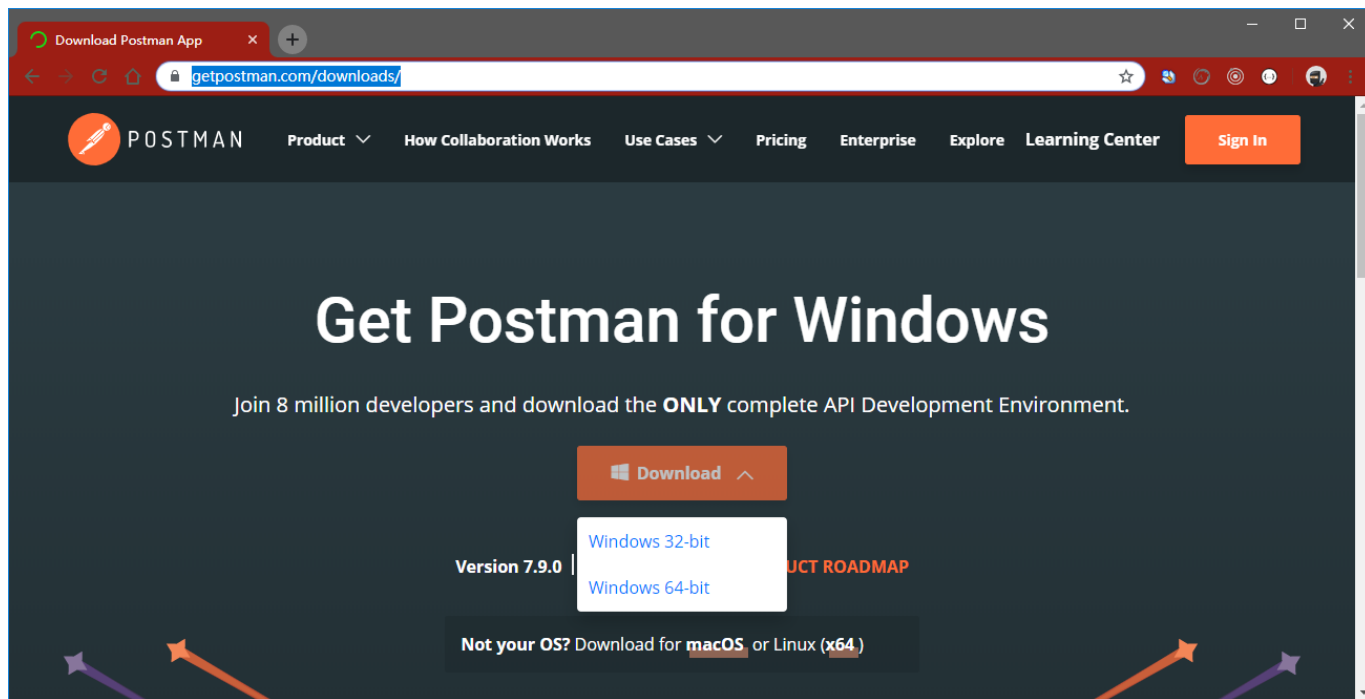
PostMan

7. 接口

7.3 接口测试工具

2. 下载并安装PostMan

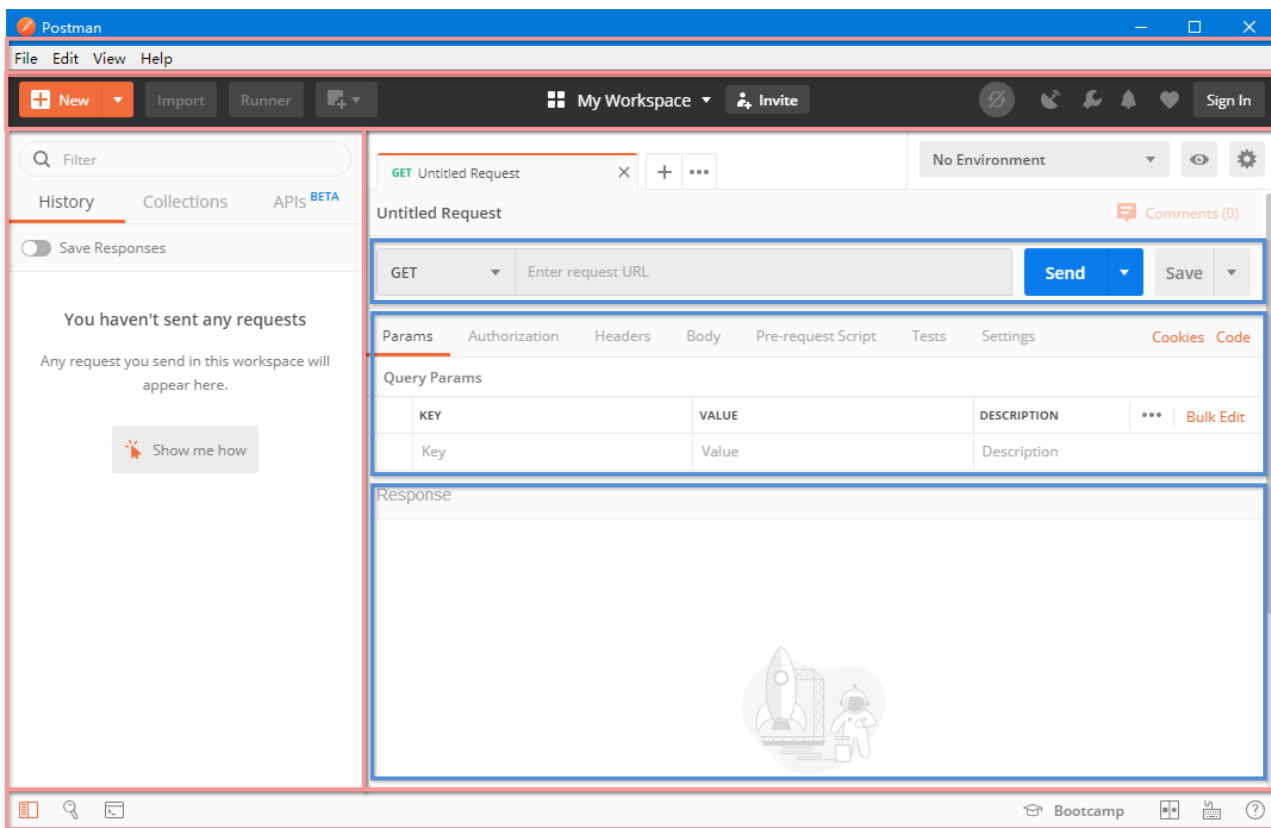
访问 PostMan 的官方下载网址 <https://www.getpostman.com/downloads/>，下载所需的安装程序后，直接安装即可。



7. 接口

7.3 接口测试工具

3. 了解PostMan界面的组成部分

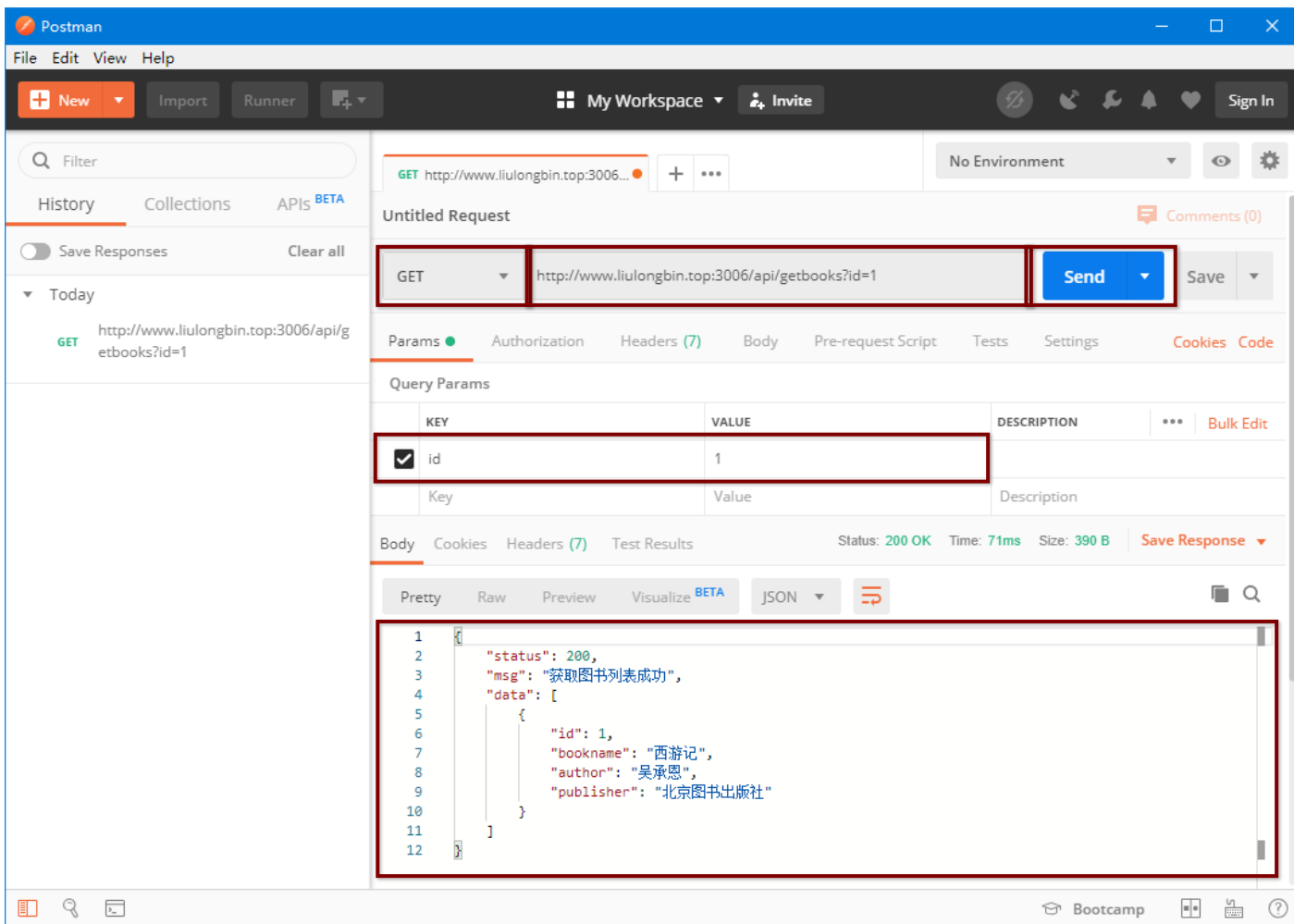


PostMan界面的组成部分，从上到下，从左到右，分别是：

- 菜单栏
- 工具栏
- 左侧历史记录与集合面板
- 请求页签
- 请求地址区域
- 请求参数区域
- 响应结果区域
- 状态栏

7. 接口

7.4 使用PostMan测试GET接口

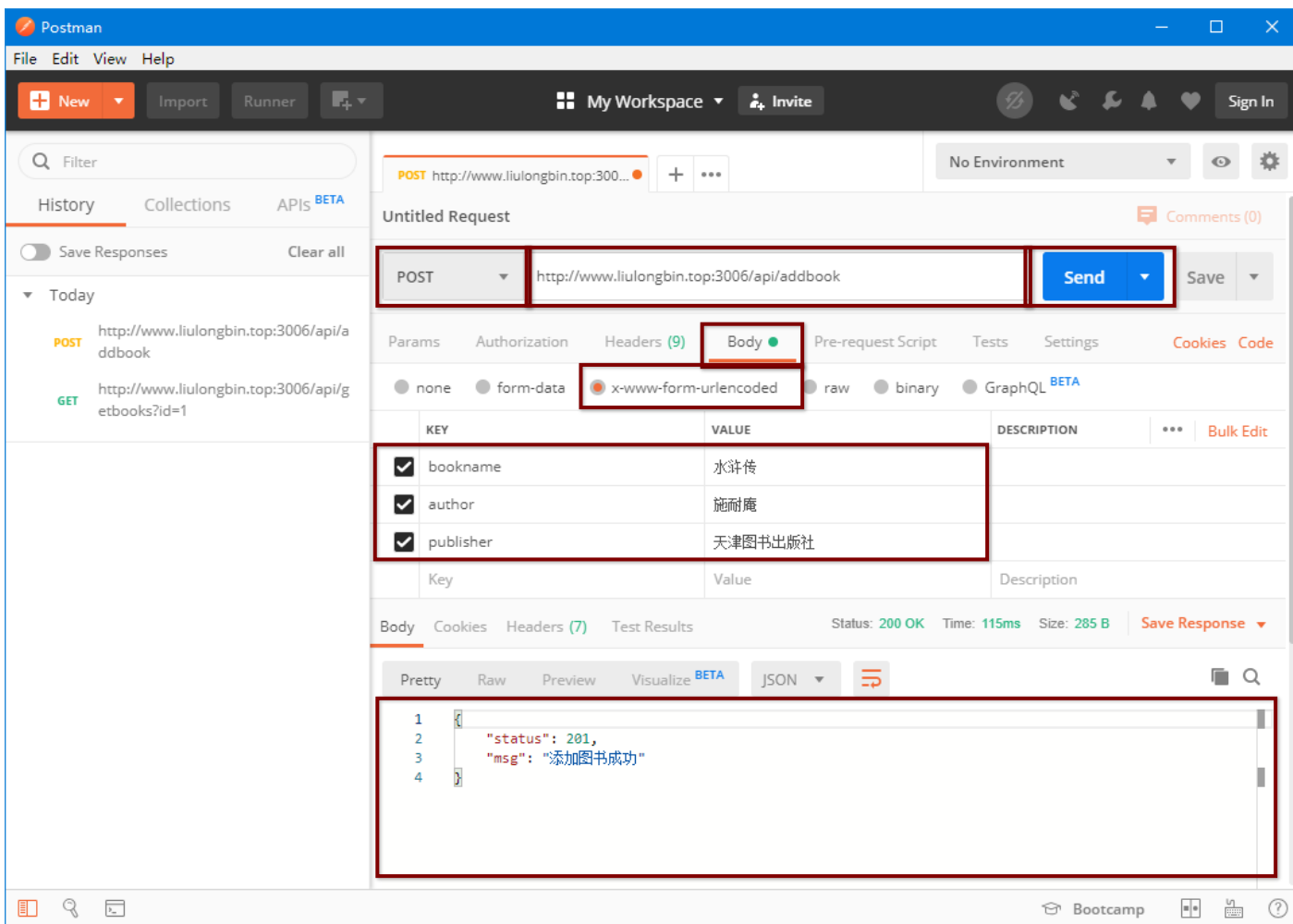


步骤：

1. 选择请求的方式
2. 填写请求的URL地址
3. 填写请求的参数
4. 点击 Send 按钮发起 GET 请求
5. 查看服务器响应的结果

7. 接口

7.5 使用PostMan测试POST接口



步骤：

1. 选择请求的方式
2. 填写请求的URL地址
3. 选择 Body 面板并勾选数据格式
4. 填写要发送到服务器的数据
5. 点击 Send 按钮发起 POST 请求
6. 查看服务器响应的结果

7. 接口

7.6 接口文档

1. 什么是接口文档

接口文档，顾名思义就是**接口的说明文档，它是我们调用接口的依据**。好的接口文档包含了对**接口URL**，**参数**以及**输出内容**的说明，我们参照接口文档就能方便的知道接口的作用，以及接口如何进行调用。

7. 接口

7.6 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](#)、[是否必选](#)、[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。

7. 接口

7.6 接口文档

3. 接口文档示例

图书列表

- 接口URL: <http://www.liulongbin.top:3006/api/getbooks>
- 调用方式: GET
- 参数格式:

参数名称	参数类型	是否必选	参数说明
id	Number	否	图书Id
bookname	String	否	图书名称
author	String	否	作者
publisher	String	否	出版社

7. 接口

7.6 接口文档

3. 接口文档示例

▪ 响应格式:

数据名称	数据类型	说明
status	Number	200 成功; 500 失败;
msg	String	对 status 字段的详细说明
data	Array	图书列表
+id	Number	图书Id
+bookname	String	图书名称
+author	String	作者
+publisher	String	出版社


7. 接口

7.6 接口文档

3. 接口文档示例

▪ 返回示例:

```
1 {  
2   "status": 200,  
3   "msg": "获取图书列表成功",  
4   "data": [  
5     { "id": 1, "bookname": "西游记", "author": "吴承恩", "publisher": "北京图书出版社" },  
6     { "id": 2, "bookname": "红楼梦", "author": "曹雪芹", "publisher": "上海图书出版社" },  
7     { "id": 3, "bookname": "三国演义", "author": "罗贯中", "publisher": "北京图书出版社" }  
8   ]  
9 }  
10
```

目录

Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

8. 案例 - 图书管理

8.1 渲染UI结构

Document

+

图书管理案例.html

添加新图书

书名

请输入书名

作者

请输入作者

出版社

请输入出版社

添加

Id	书名	作者	出版社	操作
1	西游记	吴承恩	北京图书出版社	删除
2	红楼梦	曹雪芹	上海图书出版社	删除
3	三国演义	罗贯中	北京图书出版社	删除

8. 案例 - 图书管理

8.2 案例用到的库和插件

用到的 css 库 `bootstrap.css`

用到的 javascript 库 `jquery.js`

用到的 vs code 插件 `Bootstrap 3 Snippets`

8. 案例 - 图书管理

8.3 渲染图书列表（核心代码）

```
function getBookList() {  
    // 1. 发起 ajax 请求获取图书列表数据  
    $.get('http://www.liulongbin.top:3006/api/getbooks', function(res) {  
        // 2. 获取列表数据是否成功  
        if (res.status !== 200) return alert('获取图书列表失败！')  
        // 3. 渲染页面结构  
        var rows = []  
        $.each(res.data, function(i, item) { // 4. 循环拼接字符串  
            rows.push('<tr><td>' + item.id + '</td><td>' + item.bookname +  
'</td><td>' + item.author + '</td><td>' + item.publisher + '</td><td><a  
href="javascript:;">删除</a></td></tr>')  
        })  
        $('#bookBody').empty().append(rows.join('')) // 5. 渲染表格结构  
    })  
}
```

8. 案例 - 图书管理


8.4 删除图书（核心代码）

```
// 1. 为按钮绑定点击事件处理函数
$('tbody').on('click', '.del', function() {
    // 2. 获取要删除的图书的 id
    var id = $(this).attr('data-id')
    $.ajax({ // 3. 发起 ajax 请求，根据 id 删除对应的图书
        type: 'GET',
        url: 'http://www.liulongbin.top:3006/api/delbook',
        data: { id: id },
        success: function(res) {
            if (res.status !== 200) return alert('删除图书失败！')
            getBookList() // 4. 删除成功后，重新加载图书列表
        }
    })
})
```

8. 案例 - 图书管理

8.5 添加图书（核心代码）

```
// 1. 检测内容是否为空
var bookname = $('#bookname').val()
var author = $('#author').val()
var publisher = $('#publisher').val()
if (bookname === '' || author === '' || publisher === '') {
    return alert('请完整填写图书信息！')
}
// 2. 发起 ajax 请求，添加图书信息
$.post(
    'http://www.liulongbin.top:3006/api/addbook',
    { bookname: bookname, author: author, publisher: publisher },
    function(res) {
        // 3. 判断是否添加成功
        if (res.status !== 201) return alert('添加图书失败！')
        getBookList() // 4. 添加成功后，刷新图书列表
        $('#input:text').val('') // 5. 清空文本框内容
    }
)
```

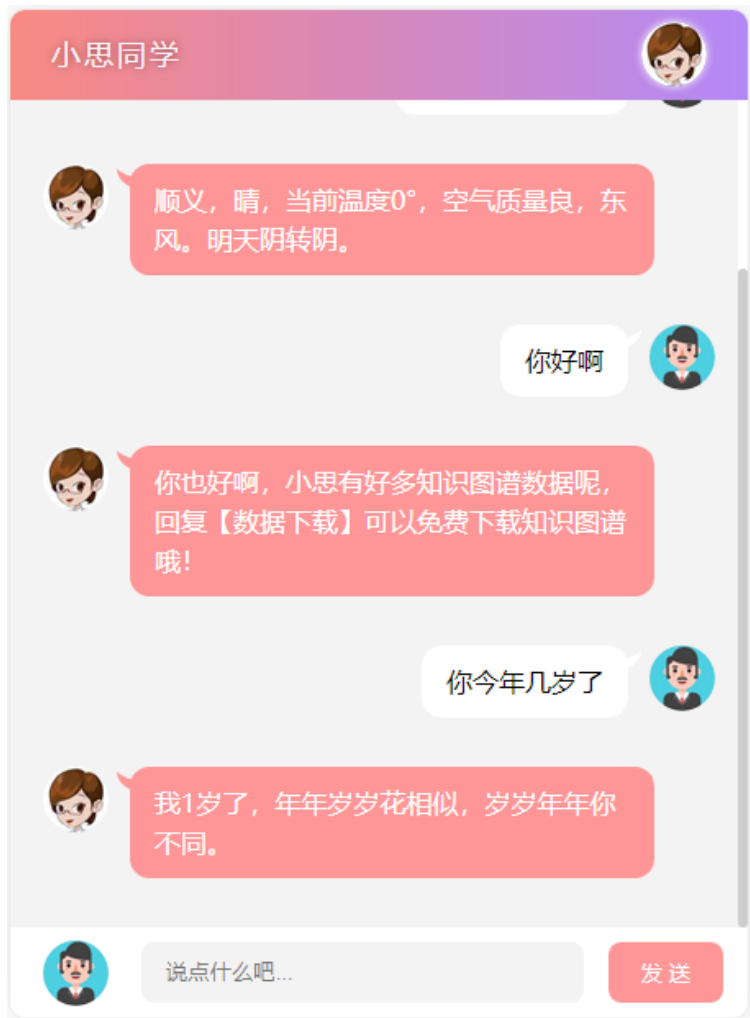


录Contents

- ◆ 客户端与服务器
- ◆ URL地址
- ◆ 分析网页的打开过程
- ◆ 服务器对外提供了哪些资源
- ◆ 了解Ajax
- ◆ jQuery中的Ajax
- ◆ 接口
- ◆ 案例 - 图书管理
- ◆ 案例 - 聊天机器人

9. 案例 – 聊天机器人

9.1 演示案例要完成的效果



实现步骤：

- ① 梳理案例的代码结构
- ② 将用户输入的内容渲染到聊天窗口
- ③ 发起请求获取聊天消息
- ④ 将机器人的聊天内容转为语音
- ⑤ 通过 `<audio>` 播放语音
- ⑥ 使用回车键发送消息

9. 案例 – 聊天机器人

9.2 梳理案例的代码结构

- ① 梳理页面的 UI 布局
- ② 将业务代码抽离到 chat.js 中
- ③ 了解 resetui() 函数的作用

9. 案例 – 聊天机器人

9.3 将用户输入的内容渲染到聊天窗口

```
// 为发送按钮绑定点击事件处理函数

$('#btnSend').on('click', function () {

    var text = $('#ipt').val().trim() // 获取用户输入的内容

    if (text.length <= 0) { // 判断用户输入的内容是否为空

        return $('#ipt').val('')

    }

    // 将用户输入的内容显示到聊天窗口中

    $('#talk_list').append('<li class="right_word"> <span>'
+ text + '</span></li>')

    resetui() // 重置滚动条的位置

    $('#ipt').val('') // 清空输入框的内容

    // TODO: 发起请求，获取聊天消息

})
```

9. 案例 – 聊天机器人

9.4 发起请求获取聊天消息

```
function getMsg(text) {  
    $.ajax({  
        method: 'GET',  
        url: 'http://ajax.frontend.itheima.net:3006/api/robot',  
        data: {  
            spoken: text  
        },  
        success: function (res) {  
            if (res.message === 'success') {  
                var msg = res.data.info.text  
                $('#talk_list').append('<li class="left_word">  
<span>' + msg + '</span></li>')  
                resetui()  
                // TODO: 发起请求，将机器人的聊天消息转为语音格式  
            }  
        }  
    })  
}
```

9. 案例 – 聊天机器人

9.5 将机器人的聊天内容转为语音

```
function getVoice(text) {  
    $.ajax({  
        method: 'GET',  
        url: 'http://ajax.frontend.itheima.net:3006/api/synthesize',  
        data: {  
            text: text  
        },  
        success: function (res) {  
            // 如果请求成功，则 res.voiceUrl 是服务器返回的音频 URL 地址  
            if (res.status === 200) {  
                $('#voice').attr('src', res.voiceUrl)  
            }  
        }  
    })  
}
```

9. 案例 – 聊天机器人

9.6 通过 <audio> 播放语音

```
<!-- 音频播放语音内容 -->
```

```
<audio src="" id="voice" autoplay style="display: none;"></audio>
```

9. 案例 – 聊天机器人

9.7 使用回车发送消息

```
// 让文本输入框响应回车事件后，提交消息
$('#ipt').on('keyup', function (e) {
    // e.keyCode 可以获取到当前按键的编码
    if (e.keyCode === 13) {
        // 调用按钮元素的 click 函数，可以通过编程的形式触发按钮的点击事件
        $('#btnSend').click()
    }
})
```



传智播客旗下高端IT教育品牌

form表单与模板引擎

目录

Contents

- ◆ form表单的基本使用
- ◆ 通过Ajax提交表单数据
- ◆ 案例 - 评论列表
- ◆ 模板引擎的基本概念
- ◆ art-template模板引擎
- ◆ 模板引擎的实现原理

1. form表单的基本使用

1.1 什么是表单

表单在网页中主要负责**数据采集功能**。HTML中的<form>标签，就是用于采集用户输入的信息，并通过<form>标签的提交操作，把采集到的信息提交到服务器端进行处理。



The image shows a login form with the following elements:

- Links: [登录](#) (Login) and [注册](#) (Register).
- Input fields: A text input for "手机号或邮箱" (Phone number or email) and a password input for "密码" (Password).
- Checkbox: A checkbox labeled "记住我" (Remember me).
- Link: A link labeled "登录遇到问题?" (Having trouble logging in?).
- Button: A blue button labeled "登录" (Login).
- Social login section: A section titled "社交帐号登录" (Social account login) with icons for Weibo, WeChat, QQ, and a link for "其它" (Others).

<form>

```
<input type="text" name="email_or_mobile" />
```

```
<input type="password" name="password" />
```

```
<input type="checkbox" name="remember_me" checked />
```

```
<button type="submit">提交</button>
```

</form>

1. form表单的基本使用

1.2 表单的组成部分

```
<form>
  <input type="text" name="email_or_mobile" />
  <input type="password" name="password" />
  <input type="checkbox" name="remember_me" checked />
  <button type="submit">提交</button>
</form>
```

表单由三个基本部分组成：

1. form表单的基本使用

1.2 表单的组成部分

```
<form>
  <input type="text" name="email_or_mobile" />
  <input type="password" name="password" />
  <input type="checkbox" name="remember_me" checked />
  <button type="submit">提交</button>
</form>
```

表单由三个基本部分组成：

- 表单标签

1. form表单的基本使用

1.2 表单的组成部分

```
<form>
  <input type="text" name="email_or_mobile" />
  <input type="password" name="password" />
  <input type="checkbox" name="remember_me" checked />
  <button type="submit">提交</button>
</form>
```

表单由三个基本部分组成：

- 表单标签
- 表单域

表单域：包含了文本框、密码框、隐藏域、多行文本框、复选框、单选框、下拉选择框和文件上传框等。

1. form表单的基本使用

1.2 表单的组成部分

```
<form>
  <input type="text" name="email_or_mobile" />
  <input type="password" name="password" />
  <input type="checkbox" name="remember_me" checked />
  <button type="submit">提交</button>
</form>
```

表单由三个基本部分组成：

- 表单标签
- 表单域
- 表单按钮

1. form表单的基本使用

1.3 <form>标签的属性

<form>标签用来采集数据，<form>标签的属性则是用来规定**如何把采集到的数据发送到服务器**。

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 <i>framename</i>	规定在何处打开 action URL

1. form表单的基本使用

1.3 <form>标签的属性

1. action

action 属性用来规定当提交表单时，**向何处发送表单数据**。

action 属性的值应该是后端提供的一个 URL 地址，这个 URL 地址专门负责接收表单提交过来的数据。

当 <form> 表单在未指定 action 属性值的情况下，action 的默认值为当前页面的 URL 地址。

注意：当提交表单后，页面会立即跳转到 action 属性指定的 URL 地址

1. form表单的基本使用

1.3 <form>标签的属性

2. target

target 属性用来规定在何处打开 action URL。

它的可选值有5个，默认情况下，target 的值是 `_self`，表示在相同的框架中打开 action URL。

值	描述
<code>_blank</code>	在新窗口中打开。
<code>_self</code>	默认。在相同的框架中打开。
<code>_parent</code>	在父框架集中打开。（很少用）
<code>_top</code>	在整个窗口中打开。（很少用）
<code>framename</code>	在指定的框架中打开。（很少用）

1. form表单的基本使用

1.3 <form>标签的属性

3. method

method 属性用来规定**以何种方式**把表单数据提交到 action URL。

它的可选值有两个，分别是 get 和 post。

默认情况下，method 的值为 get，表示**通过URL地址的形式**，把表单数据提交到 action URL。

注意：

get 方式适合用来提交少量的、简单的数据。

post 方式适合用来提交**大量的、复杂的、或包含文件上传**的数据。

在实际开发中，<form> 表单的 post 提交方式用的最多，很少用 get。例如登录、注册、添加数据等表单操作，都需要使用 post 方式来提交表单。

1. form表单的基本使用

1.3 <form>标签的属性

4. enctype

enctype 属性用来规定在发送表单数据之前如何对数据进行编码。

它的可选值有三个，默认情况下，enctype 的值为 application/x-www-form-urlencoded，表示在发送前编码所有的字符。

值	描述
application/x-www-form-urlencoded	在发送前编码所有字符（默认）
multipart/form-data	不对字符编码。 在使用包含文件上传控件的表单时，必须使用该值。
text/plain	空格转换为 “+” 加号，但不对特殊字符编码。（很少用）

1. form表单的基本使用

1.3 <form>标签的属性

4. enctype

注意：

在涉及到**文件上传**的操作时，**必须**将 enctype 的值设置为 **multipart/form-data**

如果表单的提交不涉及到文件上传操作，则直接将 enctype 的值设置为 application/x-www-form-urlencoded 即可！

1. form表单的基本使用

1.4 表单的同步提交及缺点

1. 什么是表单的同步提交

通过点击 submit 按钮，触发表单提交的操作，从而使页面跳转到 action URL 的行为，叫做表单的同步提交。

1. form表单的基本使用

1.4 表单的同步提交及缺点

2. 表单同步提交的缺点

- ① <form>表单同步提交后，整个页面会发生跳转，**跳转到 action URL 所指向的地址**，用户体验很差。
- ② <form>表单同步提交后，**页面之前的状态和数据会丢失**。

思考：如何解决上述两个问题？

1. form表单的基本使用

1.4 表单的同步提交及缺点

3. 如何解决表单同步提交的缺点

如果使用表单提交数据，则会导致以下两个问题：

- ① 页面会发生跳转
- ② 页面之前的状态和数据会丢失

解决方案：**表单只负责采集数据，Ajax 负责将数据提交到服务器。**

目录

Contents

- ◆ form表单的基本使用
- ◆ 通过Ajax提交表单数据
- ◆ 案例 - 评论列表
- ◆ 模板引擎的基本概念
- ◆ art-template模板引擎
- ◆ 模板引擎的实现原理

2. 通过Ajax提交表单数据

2.1 监听表单提交事件

在 jQuery 中，可以使用如下两种方式，监听到表单的提交事件：

```
$('#form1').submit(function(e) {  
    alert('监听到了表单的提交事件')  
})  
  
$('#form1').on('submit', function(e) {  
    alert('监听到了表单的提交事件')  
})
```

2. 通过Ajax提交表单数据

2.2 阻止表单默认提交行为

当监听到表单的提交事件以后，可以调用事件对象的 `event.preventDefault()` 函数，来阻止表单的提交和页面的跳转，示例代码如下：

```
$('#form1').submit(function(e) {  
    // 阻止表单的提交和页面的跳转  
    e.preventDefault()  
})  
  
$('#form1').on('submit', function(e) {  
    // 阻止表单的提交和页面的跳转  
    e.preventDefault()  
})
```

2. 通过Ajax提交表单数据

2.3 快速获取表单中的数据

1. serialize()函数

为了简化表单中数据的获取操作，jQuery 提供了 serialize() 函数，其语法格式如下：

```
$(selector).serialize()
```

serialize() 函数的好处：**可以一次性获取到表单中的所有数据。**

2. 通过Ajax提交表单数据

2.3 快速获取表单中的数据

2. serialize()函数示例

```
<form id="form1">
  <input type="text" name="username" />
  <input type="password" name="password" />
  <button type="submit">提交</button>
</form>
```

```
$('#form1').serialize()
// 调用的结果：
// username=用户名的值&password=密码的值
```

注意：在使用 serialize() 函数快速获取表单数据时，**必须为每个表单元素添加 name 属性**！

目录

Contents

- ◆ form表单的基本使用
- ◆ 通过Ajax提交表单数据
- ◆ 案例 - 评论列表
- ◆ 模板引擎的基本概念
- ◆ art-template模板引擎
- ◆ 模板引擎的实现原理

3. 案例 - 评论列表

3.1 渲染UI结构

Document

评论列表案例.html

发表评论

评论人:

请输入评论人

评论内容:

请输入评论内容 (最多120字)

发表评论

欢迎大家

评论人: 小红 评论时间: 2019-10-25 16:28:19

萨瓦迪卡

评论人: 小白 评论时间: 2019-10-25 16:23:05

宝塔镇河妖

评论人: 李四 评论时间: 2019-10-25 16:18:52

天王盖地虎

评论人: 张三 评论时间: 2019-10-25 16:12:57

3. 案例 - 评论列表

3.2 获取评论列表

```
function getCmtList() {  
    $.get('http://www.liulongbin.top:3006/api/cmtlist', function (res) {  
        if(res.status !== 200) {  
            return alert('获取评论列表失败！')  
        }  
        var rows = []  
        $.each(res.data, function (i, item) { // 循环拼接字符串  
            rows.push('<li class="list-group-item">' + item.content + '<span  
class="badge cmt-date">评论时间：' + item.time + '</span><span class="badge cmt-  
person">评论人：' + item.username + '</span></li>')  
        })  
        $('#cmt-list').empty().append(rows.join(' ')) // 渲染列表的UI结构  
    })  
}
```

3. 案例 - 评论列表

3.3 发表评论

```
$('#formAddCmt').submit(function(e) {  
    e.preventDefault() // 阻止表单的默认提交行为  
    // 快速得到表单中的数据  
    var data = $(this).serialize()  
    $.post('http://www.liulongbin.top:3006/api/addcmt', data, function(res) {  
        if (res.status !== 201) {  
            return alert('发表评论失败！')  
        }  
        // 刷新评论列表  
        getCmtList()  
        // 快速清空表单内容  
        $('#formAddCmt')[0].reset()  
    })  
})
```


目录 Contents

- ◆ form表单的基本使用
- ◆ 通过Ajax提交表单数据
- ◆ 案例 - 评论列表
- ◆ 模板引擎的基本概念
- ◆ art-template模板引擎
- ◆ 模板引擎的实现原理

4. 模板引擎的基本概念

4.1 渲染UI结构时遇到的问题

```
var rows = []  
$.each(res.data, function (i, item) { // 循环拼接字符串  
    rows.push('<li class="list-group-item">' + item.content + '<span class="badge  
cmt-date">评论时间: ' + item.time + '</span><span class="badge cmt-person">评论人: ' +  
item.username + '</span></li>')  
})  
$('#cmt-list').empty().append(rows.join('')) // 渲染列表的UI结构
```

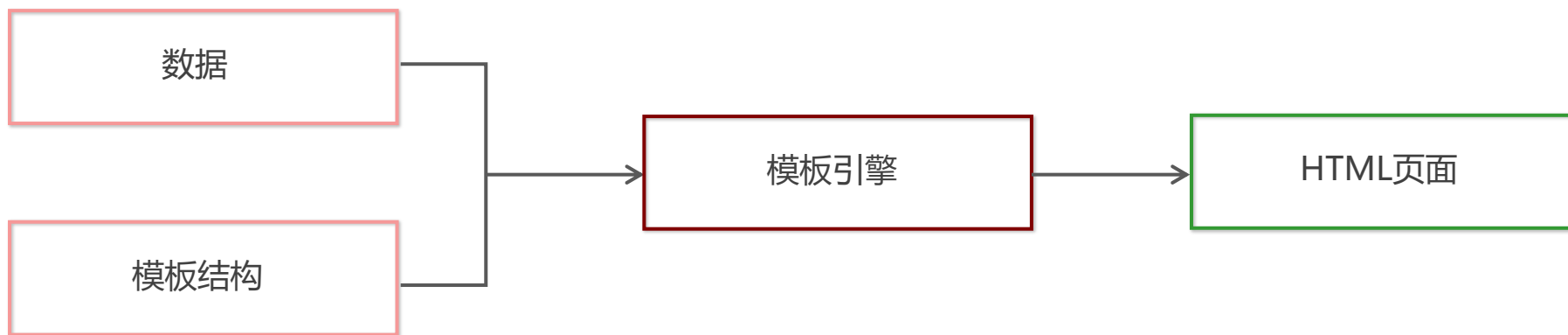
上述代码是通过**字符串拼接**的形式，来渲染UI结构。

如果UI结构比较复杂，则拼接字符串的时候需要格外注意**引号之前的嵌套**。且一旦需求发生变化，**修改起来也非常麻烦**。

4. 模板引擎的基本概念

4.2 什么是模板引擎


模板引擎，顾名思义，它可以根据程序员指定的**模板结构**和**数据**，自动生成一个完整的HTML页面。



4. 模板引擎的基本概念

4.3 模板引擎的好处

- ① 减少了字符串的拼接操作
- ② 使代码结构更清晰
- ③ 使代码更易于阅读与维护



目录

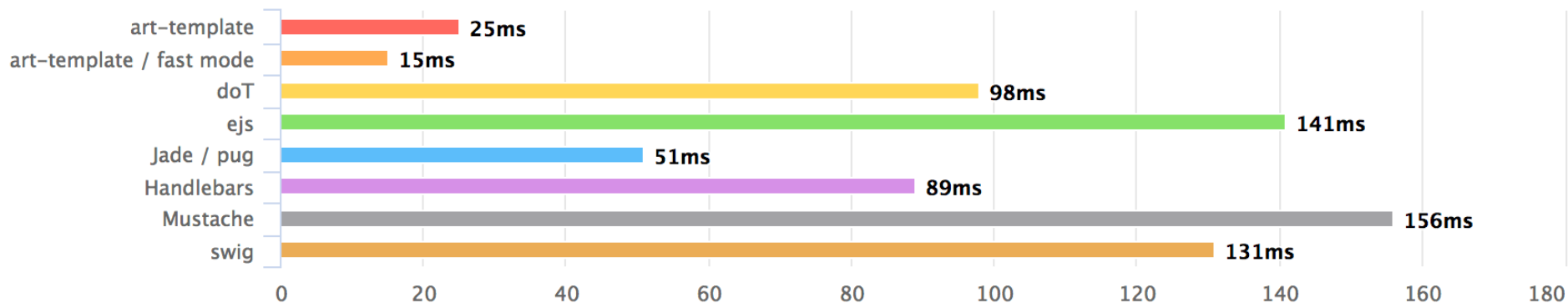
Contents

- ◆ form表单的基本使用
- ◆ 通过Ajax提交表单数据
- ◆ 案例 - 评论列表
- ◆ 模板引擎的基本概念
- ◆ art-template模板引擎
- ◆ 模板引擎的实现原理

5. art-template模板引擎

5.1 art-template简介

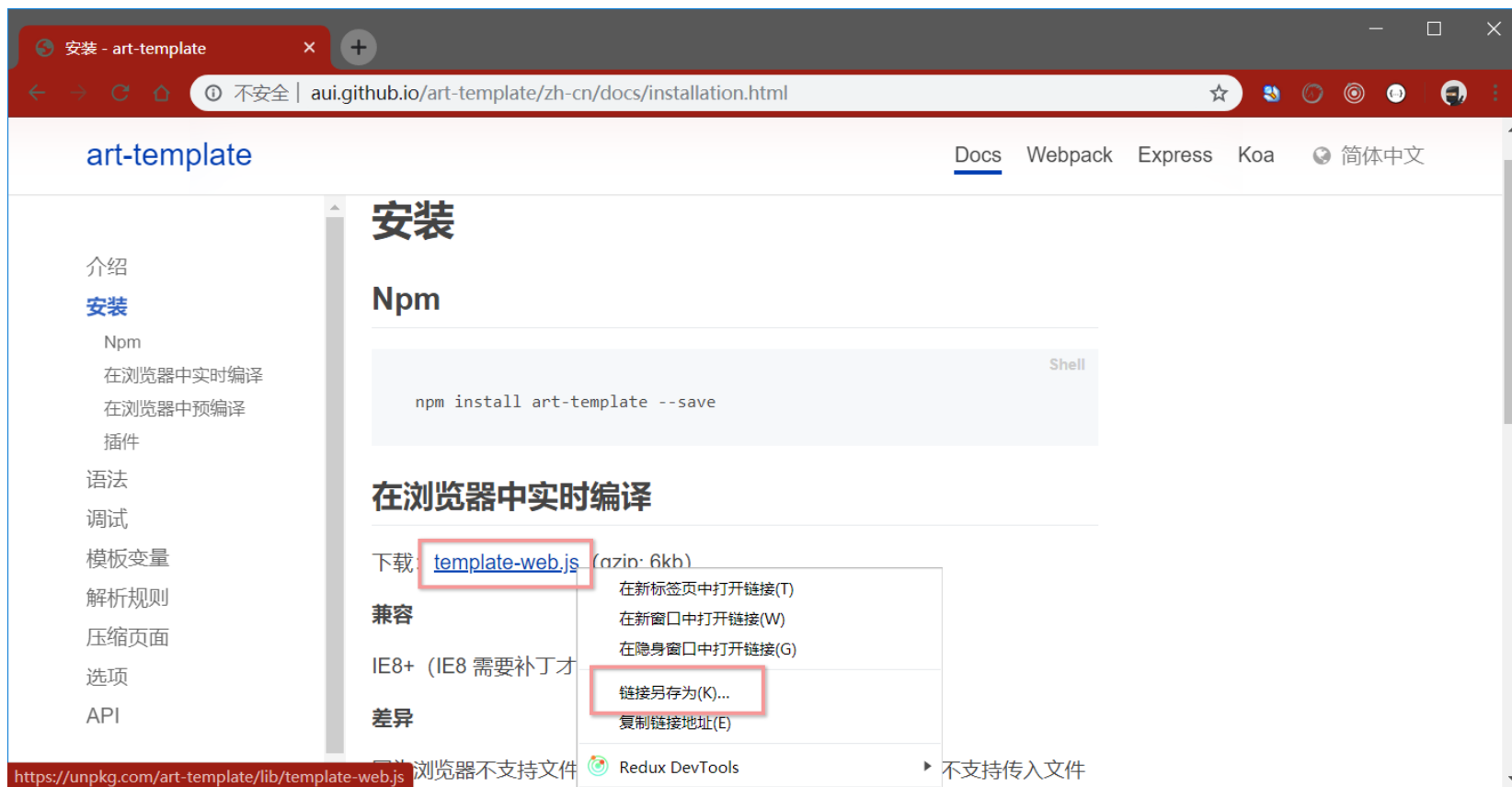
art-template 是一个简约、超快的模板引擎。中文官网首页为 <http://aui.github.io/art-template/zh-cn/index.html>



5. art-template模板引擎

5.2 art-template的安装

在浏览器中访问 <http://aui.github.io/art-template/zh-cn/docs/installation.html> 页面，找到下载链接后，鼠标右键，选择“**链接另存为**”，将 art-template 下载到本地，然后，通过 <script> 标签加载到网页上进行使用。



5. art-template模板引擎

5.3 art-template模板引擎的基本使用

1. 使用传统方式渲染UI结构

```
var data = {  
  title: '<h3>用户信息</h3>',  
  name: 'zs',  
  age: 20,  
  isVIP: true,  
  regTime: new Date(),  
  hobby: ['吃饭', '睡觉', '打豆豆']  
}
```

用户信息

姓名: zs

年龄: 20

会员: 否

注册时间: 2019-10-28

爱好:

- 吃饭
- 睡觉
- 打豆豆

5. art-template模板引擎

5.3 art-template模板引擎的基本使用

2. art-template的使用步骤

- 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

5. art-template模板引擎

5.4 art-template标准语法

1. 什么是标准语法

art-template 提供了 **{{ }}** 这种语法格式，在 **{{ }}** 内可以进行**变量输出**，或**循环数组**等操作，这种 **{{ }}** 语法在 art-template 中被称为标准语法。

5. art-template模板引擎

5.4 art-template标准语法

2. 标准语法 - 输出

```
{{value}}  
{{obj.key}}  
{{obj['key']}}  
{{a ? b : c}}  
{{a || b}}  
{{a + b}}
```

在 {{ }} 语法中，可以进行变量的输出、对象属性的输出、三元表达式输出、逻辑或输出、加减乘除等表达式输出。

5. art-template模板引擎

5.4 art-template标准语法

3. 标准语法 – 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。

5. art-template模板引擎

5.4 art-template标准语法

4. 标准语法 – 条件输出

如果要实现条件输出，则可以在 `{{}}` 中使用 `if ... else if ... /if` 的方式，进行按需输出。

```
{{if value}} 按需输出的内容 {{/if}}
```

```
{{if v1}} 按需输出的内容 {{else if v2}} 按需输出的内容 {{/if}}
```

5. art-template模板引擎

5.4 art-template标准语法

5. 标准语法 – 循环输出

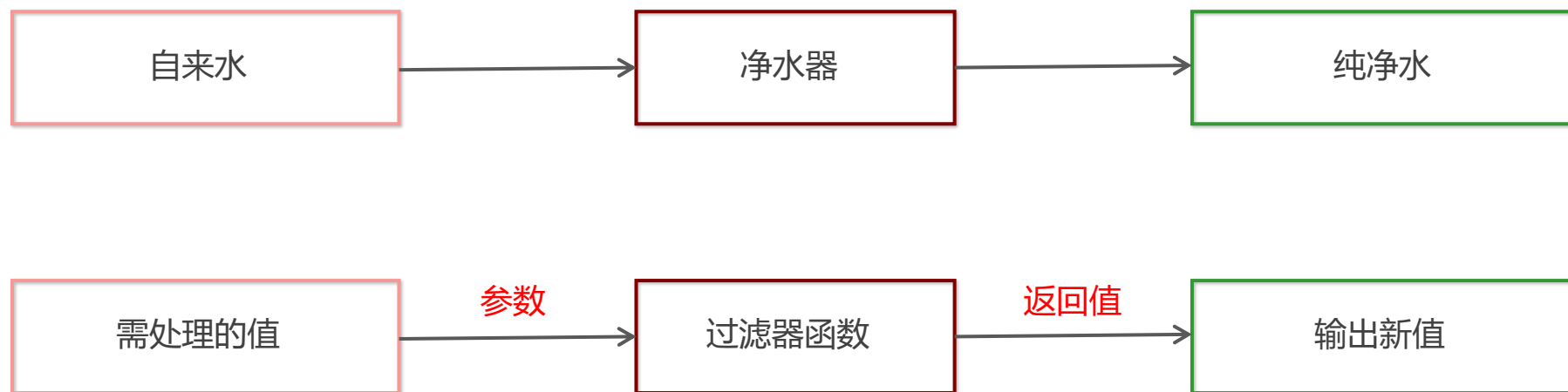
如果要实现循环输出，则可以在 `{{}}` 内，通过 `each` 语法循环数组，当前循环的索引使用 `$index` 进行访问，当前的循环项使用 `$value` 进行访问。

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

5. art-template模板引擎

5.4 art-template标准语法

6. 标准语法 – 过滤器



过滤器的本质，就是一个 function 处理函数。

5. art-template模板引擎

5.4 art-template标准语法

6. 标准语法 – 过滤器

```
{{value | filterName}}
```

过滤器语法类似**管道操作符**，它的上一个输出作为下一个输入。

定义过滤器的基本语法如下：

```
template.defaults.imports.filterName = function(value) { /*return处理的结果*/ }
```


5. art-template模板引擎

5.4 art-template标准语法

6. 标准语法 – 过滤器

```
<div>注册时间：{{regTime | dateFormat}}</div>
```

定义一个格式化时间的过滤器 dateFormat 如下：

```
template.defaults.imports.dateFormat = function(date) {  
  var y = date.getFullYear()  
  var m = date.getMonth() + 1  
  var d = date.getDate()  
  
  return y + '-' + m + '-' + d // 注意，过滤器最后一定要 return 一个值  
}
```

5. art-template模板引擎

5.6 案例 – 新闻列表




5. art-template模板引擎

5.6 案例 – 新闻列表

1. 实现步骤

- ① 获取新闻数据
- ② 定义 template 模板
- ③ 编译模板
- ④ 定义时间过滤器
- ⑤ 定义补零函数



目录

Contents

- ◆ form表单的基本使用
- ◆ 通过Ajax提交表单数据
- ◆ 案例 - 评论列表
- ◆ 模板引擎的基本概念
- ◆ art-template模板引擎
- ◆ 模板引擎的实现原理

6. 模板引擎的实现原理

6.1 正则与字符串操作

1. 基本语法

`exec()` 函数用于检索字符串中的正则表达式的匹配。

如果字符串中有匹配的值，则返回该匹配值，否则返回 `null`。

```
RegExpObject.exec(string)
```

示例代码如下：

```
var str = 'hello'  
var pattern = /o/  
// 输出的结果["o", index: 4, input: "hello", groups: undefined]  
console.log(pattern.exec(str))
```

6. 模板引擎的实现原理

6.1 正则与字符串操作

2. 分组

正则表达式中 () 包起来的内容表示一个分组，可以通过分组来提取自己想要的内容，示例代码如下：

```
var str = '<div>我是{{name}}</div>'
var pattern = /{{ ([a-zA-Z]+) }} /

var patternResult = pattern.exec(str)
console.log(patternResult)
// 得到 name 相关的分组信息
// ["{{name}}", "name", index: 7, input: "<div>我是{{name}}</div>", groups: undefined]
```

6. 模板引擎的实现原理

6.1 正则与字符串操作

3. 字符串的replace函数

replace() 函数用于在字符串中用一些字符替换另一些字符，语法格式如下：

```
var result = '123456'.replace('123', 'abc') // 得到的 result 的值为字符串 'abc456'
```

示例代码如下：

```
var str = '<div>我是{{name}}</div>'
var pattern = /{{([a-zA-Z]+)}}/

var patternResult = pattern.exec(str)
str = str.replace(patternResult[0], patternResult[1]) // replace 函数返回值为替换后的新字符串
// 输出的内容是：<div>我是name</div>
console.log(str)
```

6. 模板引擎的实现原理

6.1 正则与字符串操作

4. 多次replace

```
var str = '<div>{{name}}今年{{ age }}岁了</div>'
var pattern = /{{\s*([a-zA-Z]+)\s*}}/

var patternResult = pattern.exec(str)
str = str.replace(patternResult[0], patternResult[1])
console.log(str) // 输出 <div>name今年{{ age }}岁了</div>

patternResult = pattern.exec(str)
str = str.replace(patternResult[0], patternResult[1])
console.log(str) // 输出 <div>name今年age岁了</div>

patternResult = pattern.exec(str)
console.log(patternResult) // 输出 null
```


6. 模板引擎的实现原理

6.1 正则与字符串操作

5. 使用while循环replace

```
var str = '<div>{{name}}今年{{ age }}岁了</div>'  
var pattern = /{{\s*([a-zA-Z]+\s*)}}/  
  
var patternResult = null  
while(patternResult = pattern.exec(str)) {  
    str = str.replace(patternResult[0], patternResult[1])  
}  
  
console.log(str) // 输出 <div>name今年age岁了</div>
```

6. 模板引擎的实现原理

6.1 正则与字符串操作

6. replace替换为真值

```
var data = { name: '张三', age: 20 }
var str = '<div>{{name}}今年{{ age }}岁了</div>'
var pattern = /{{\s*([a-zA-Z]+\s*)\s*}}/

var patternResult = null
while ((patternResult = pattern.exec(str))) {
    str = str.replace(patternResult[0], data[patternResult[1]])
}
console.log(str)
```

6. 模板引擎的实现原理

6.2 实现简易的模板引擎

1. 实现步骤

- ① 定义模板结构
- ② 预调用模板引擎
- ③ 封装 `template` 函数
- ④ 导入并使用自定义的模板引擎

6. 模板引擎的实现原理

6.2 实现简易的模板引擎

2. 定义模板结构

```
<!-- 定义模板结构 -->  
<script type="text/html" id="tpl-user">  
    <div>姓名 : {{name}}</div>  
    <div>年龄 : {{ age }}</div>  
    <div>性别 : {{ gender }}</div>  
    <div>住址 : {{address }}</div>  
</script>
```

6. 模板引擎的实现原理

6.2 实现简易的模板引擎

3. 预调用模板引擎

```
<script>
  // 定义数据
  var data = { name: 'zs', age: 28, gender: '男', address: '北京顺义马坡' }

  // 调用模板函数
  var htmlStr = template('tpl-user', data)

  // 渲染HTML结构
  document.getElementById('user-box').innerHTML = htmlStr
</script>
```

6. 模板引擎的实现原理

6.2 实现简易的模板引擎

4. 封装template函数

```
function template(id, data) {  
    var str = document.getElementById(id).innerHTML  
    var pattern = /{{\s*([a-zA-Z]+\s*)\s*}}/  
  
    var pattResult = null  
  
    while ((pattResult = pattern.exec(str))) {  
        str = str.replace(pattResult[0], data[pattResult[1]])  
    }  
  
    return str  
}
```

6. 模板引擎的实现原理

6.2 实现简易的模板引擎


5. 导入并使用自定义的模板引擎

```
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>自定义模板引擎</title>
  <!-- 导入自定义的模板引擎 -->
  <script src="./js/template.js"></script>
</head>
```



传智播客旗下高端IT教育品牌

Ajax加强



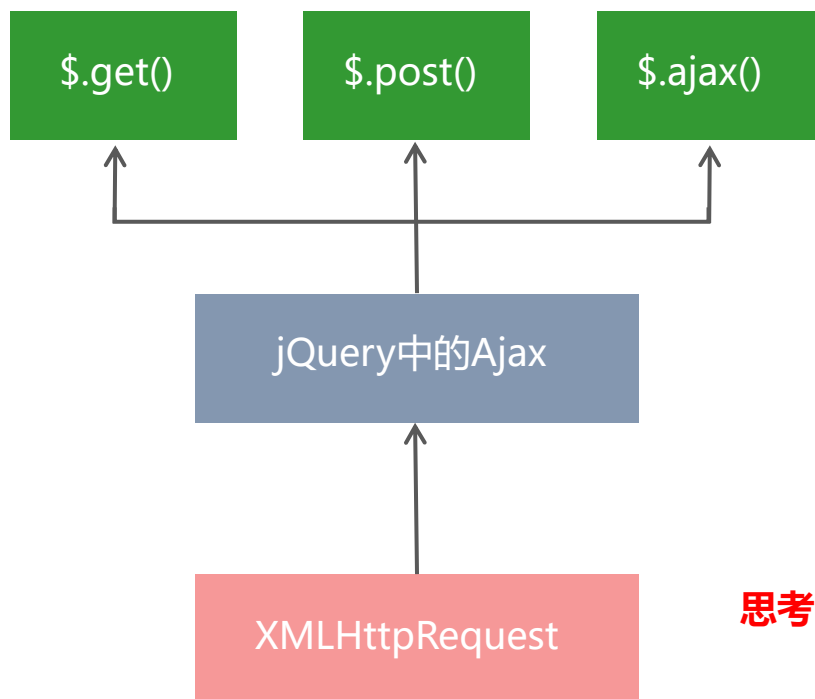
目录Contents

- ◆ XMLHttpRequest的基本使用
- ◆ 数据交换格式
- ◆ 封装自己的Ajax函数
- ◆ XMLHttpRequest Level2的新特性
- ◆ jQuery高级用法
- ◆ axios

1. XMLHttpRequest的基本使用

1.1 什么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。



思考：能否直接使用xhr对象发起Ajax请求？

1. XMLHttpRequest的基本使用

1.2 使用xhr发起GET请求

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

1. XMLHttpRequest的基本使用

1.2 使用xhr发起GET请求

```
// 1. 创建 XHR 对象
var 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. XMLHttpRequest的基本使用

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。

1. XMLHttpRequest的基本使用

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. XMLHttpRequest的基本使用

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，可以将想要发送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`

1. XMLHttpRequest的基本使用

1.5 查询字符串

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，都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. XMLHttpRequest的基本使用

1.6 URL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
```

1. XMLHttpRequest的基本使用

1.6 URL编码与解码

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：

- encodeURI() 编码的函数
- decodeURI() 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
```

1. XMLHttpRequest的基本使用

1.6 URL编码与解码

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. XMLHttpRequest的基本使用

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. XMLHttpRequest的基本使用

1.7 使用xhr发起POST请求


```
// 1. 创建 xhr 对象
var xhr = new 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)
    }
}
```



目录

Contents

- ◆ XMLHttpRequest的基本使用
- ◆ 数据交换格式
- ◆ 封装自己的Ajax函数
- ◆ XMLHttpRequest Level2的新特性
- ◆ jQuery高级用法
- ◆ axios

2. 数据交换格式

2.1 什么是数据交换格式

数据交换格式，就是服务器端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. 数据交换格式

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. 数据交换格式

2.2 XML

2. XML和HTML的区别

XML 和 HTML 虽然都是标记语言，但是，它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



2. 数据交换格式

2.2 XML

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. 数据交换格式

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. 数据交换格式

2.3 JSON

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。

2. 数据交换格式

2.3 JSON

2. JSON的两种结构

对象结构：对象结构在JSON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"]
}
```

2. 数据交换格式

2.3 JSON

2. JSON的两种结构

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 ] ]
```

2. 数据交换格式

2.3 JSON

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

2. 数据交换格式

2.3 JSON

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  
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"}'
```

2. 数据交换格式

2.3 JSON

5. JSON和JS对象的互转

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

```
var obj = JSON.parse('{"a": "Hello", "b": "World"}')  
//结果是 {a: 'Hello', b: 'World'}
```

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

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


2. 数据交换格式

2.3 JSON

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，叫做 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.parse()` 函数的操作，叫做 JSON 反序列化。



目录

Contents

- ◆ XMLHttpRequest的基本使用
- ◆ 数据交换格式
- ◆ 封装自己的Ajax函数
- ◆ XMLHttpRequest Level2的新特性
- ◆ jQuery高级用法
- ◆ axios

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. 封装自己的Ajax函数

3.2 定义options参数选项

itheima() 函数是我们自定义的 Ajax 函数，它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 URL 地址
- data 请求携带的数据
- success 请求成功之后的回调函数

3. 封装自己的Ajax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. 封装自己的Ajax函数

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. 封装自己的Ajax函数

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)  
}
```



目录

Contents

- ◆ XMLHttpRequest的基本使用
- ◆ 数据交换格式
- ◆ 封装自己的Ajax函数
- ◆ XMLHttpRequest Level2的新特性
- ◆ jQuery高级用法
- ◆ axios

4. XMLHttpRequest Level2的新特性

4.1 认识XMLHttpRequest Level2

1. 旧版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

4. XMLHttpRequest Level2的新特性

4.1 认识XMLHttpRequest Level2

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. XMLHttpRequest Level2的新特性

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(event){  
    alert('请求超时！')  
}
```

4. XMLHttpRequest Level2的新特性

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
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)
```

4. XMLHttpRequest Level2的新特性

4.3 FormData对象管理表单数据

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() {}
})
```

4. XMLHttpRequest Level2的新特性

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 FormData 中追加文件
- ④ 使用 xhr 发起上传文件的请求
- ⑤ 监听 onreadystatechange 事件

4. XMLHttpRequest Level2的新特性

4.4 上传文件

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" />
```

4. XMLHttpRequest Level2的新特性

4.4 上传文件

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
})
```

4. XMLHttpRequest Level2的新特性

4.4 上传文件

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. XMLHttpRequest Level2的新特性

4.4 上传文件

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)
```

4. XMLHttpRequest Level2的新特性

4.4 上传文件

5. 监听onreadystatechange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. XMLHttpRequest Level2的新特性

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
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)
    }
}
```

4. XMLHttpRequest Level2的新特性

4.5 显示文件上传进度

1. 导入需要的库

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

4. XMLHttpRequest Level2的新特性

4.5 显示文件上传进度

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>
```


4. XMLHttpRequest Level2的新特性

4.5 显示文件上传进度

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. XMLHttpRequest Level2的新特性

4.5 显示文件上传进度

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')  
}
```



录Contents

- ◆ XMLHttpRequest的基本使用
- ◆ 数据交换格式
- ◆ 封装自己的Ajax函数
- ◆ XMLHttpRequest Level2的新特性
- ◆ jQuery高级用法
- ◆ axios

5. jQuery高级用法

5.1 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>
```

5. jQuery高级用法

5.1 jQuery实现文件上传

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！')  
    }  
})
```

5. jQuery高级用法

5.1 jQuery实现文件上传

3. 向FormData中追加文件

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

5. jQuery高级用法

5.1 jQuery实现文件上传

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. jQuery高级用法

5.2 jQuery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，示例代码如下：

```
// 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**会监听当前文档内所有的 Ajax 请求**。


5. jQuery高级用法

5.2 jQuery实现loading效果

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，示例代码如下：

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



目录

Contents

- ◆ XMLHttpRequest的基本使用
- ◆ 数据交换格式
- ◆ 封装自己的Ajax函数
- ◆ XMLHttpRequest Level2的新特性
- ◆ jQuery高级用法
- ◆ axios

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. axios

6.2 axios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
var url = 'http://www.liulongbin.top:3006/api/get'
// 请求的参数对象
var 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. axios

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
var url = 'http://www.liulongbin.top:3006/api/post'
// 要提交到服务器的数据
var 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
    var result = res.data
    console.log(result)
})
```

6. axios

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)
```

6. axios

6.4 直接使用axios发起请求

1. 直接使用axios发起GET请求

```
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)
})
```

6. axios

6.4 直接使用axios发起请求

2. 直接使用axios发起POST请求

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




传智播客旗下高端IT教育品牌

跨域与JSONP

目录 Contents

- ◆ 了解同源策略和跨域
- ◆ JSONP
- ◆ 案例-淘宝搜索
- ◆ 防抖和节流

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）

1. 了解同源策略和跨域

1.1 同源策略

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. 了解同源策略和跨域

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。

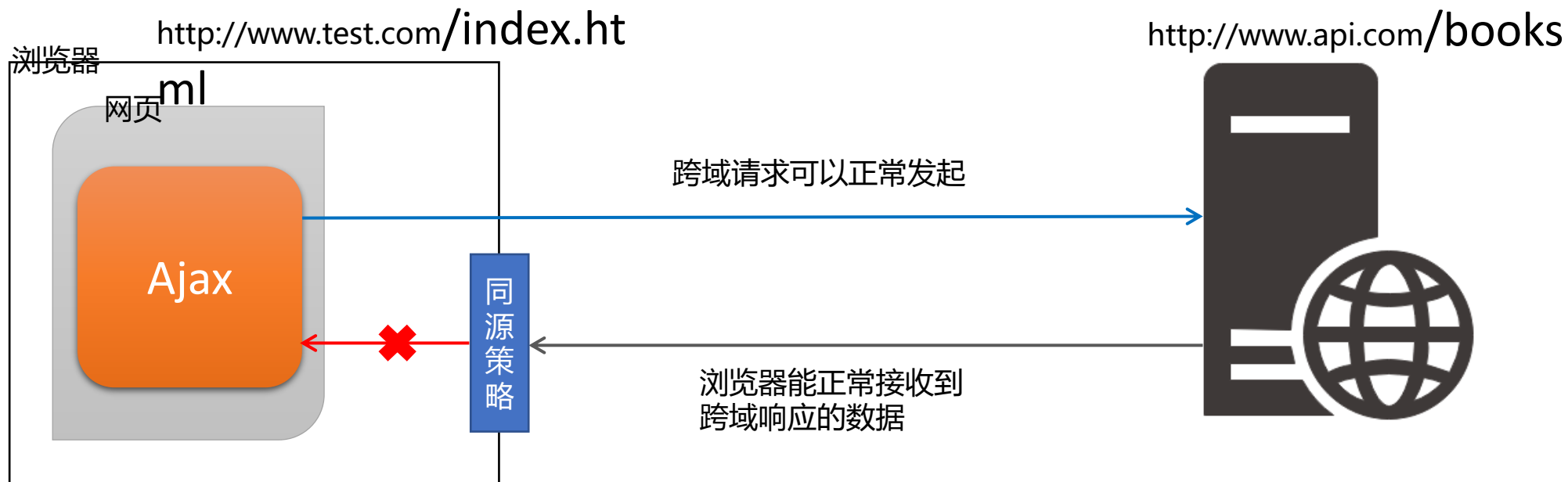
网页：`http://www.test.com/index.html`

接口：`http://www.api.com/userlist`

1. 了解同源策略和跨域

1.2 跨域

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！

1. 了解同源策略和跨域

1.2 跨域

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，它是 W3C 标准，属于跨域 Ajax 请求的根本解决方案。支持 GET 和 POST 请求。缺点是不兼容某些低版本的浏览器。

目录 Contents

- ◆ 了解同源策略和跨域
- ◆ JSONP
- ◆ 案例-淘宝搜索
- ◆ 防抖和节流

2. JSONP

2.1 什么是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. JSONP

2.2 JSONP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. JSONP

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

2. JSONP

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. JSONP

2.5 jQuery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://ajax.frontend.itheima.net:3006/api/jsonp?name=zs&age=20',  
  // 如果要使用 $.ajax() 发起 JSONP 请求，必须指定 datatype 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名称。

2. JSONP

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**回调函数名称**，可以通过如下两个参数来指定：

```
$.ajax({  
  url: 'http://ajax.frontend.itheima.net:3006/api/jsonp?name=zs&age=20',  
  dataType: 'jsonp',  
  // 发送到服务端的参数名称，默认值为 callback  
  jsonp: 'callback',  
  // 自定义的回调函数名称，默认值为 jQueryxxx 格式  
  jsonpCallback: 'abc',  
  success: function(res) {  
    console.log(res)  
  }  
})
```

2. JSONP

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；

目录

Contents

- ◆ 了解同源策略和跨域
- ◆ JSONP
- ◆ 案例-淘宝搜索
- ◆ 防抖和节流

3. 案例 – 淘宝搜索

3.1 要实现的UI效果



3. 案例 – 淘宝搜索

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，示例代码如下：

```
// 监听文本框的 keyup 事件
$('#ipt').on('keyup', function() {
    // 获取用户输入的内容
    var 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. 案例 – 淘宝搜索

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，示例代码如下：

```
function getSuggestList(kw) {  
    $.ajax({  
        // 指定请求的 URL 地址，其中，q 是用户输入的关键词  
        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. 案例 – 淘宝搜索

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>
```

3. 案例 – 淘宝搜索

3.4 渲染建议列表的UI结构

2. 定义模板结构

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

3. 案例 – 淘宝搜索

3.4 渲染建议列表的UI结构

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
  var htmlStr = template('tpl-suggestList', res)
  $('#suggest-list').html(htmlStr).show()
}
```

3. 案例 – 淘宝搜索

3.4 渲染建议列表的UI结构

4. 搜索关键词为空时隐藏搜索建议列表

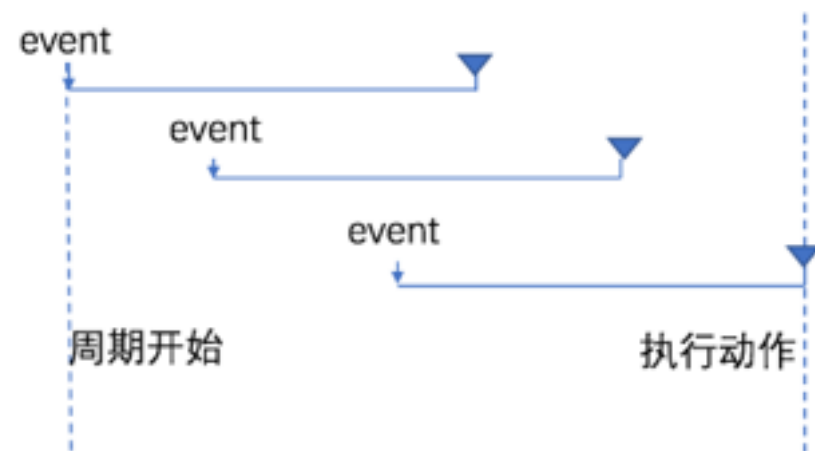
```
$('#ipt').on('keyup', function() {  
    // 获取用户输入的内容  
    var keywords = $(this).val().trim()  
    // 判断用户输入的内容是否为空  
    if (keywords.length <= 0) {  
        // 如果关键词为空，则清空后隐藏搜索建议列表  
        return $('#suggest-list').empty().hide()  
    }  
    getSuggestList(keywords)  
})
```


3. 案例 – 淘宝搜索

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. 案例 – 淘宝搜索

3.5 输入框的防抖

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，只在输入完后，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. 案例 – 淘宝搜索

3.5 输入框的防抖

3. 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)
})
```

3. 案例 – 淘宝搜索

3.6 缓存搜索的建议列表

1. 定义全局缓存对象

```
// 缓存对象  
var cacheObj = {}
```

3. 案例 – 淘宝搜索

3.6 缓存搜索的建议列表

2. 将搜索结果保存到缓存对象中

```
// 渲染建议列表

function renderSuggestList(res) {
    // ...省略其他代码

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

3. 案例 – 淘宝搜索

3.6 缓存搜索的建议列表

3. 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)
})
```

目录

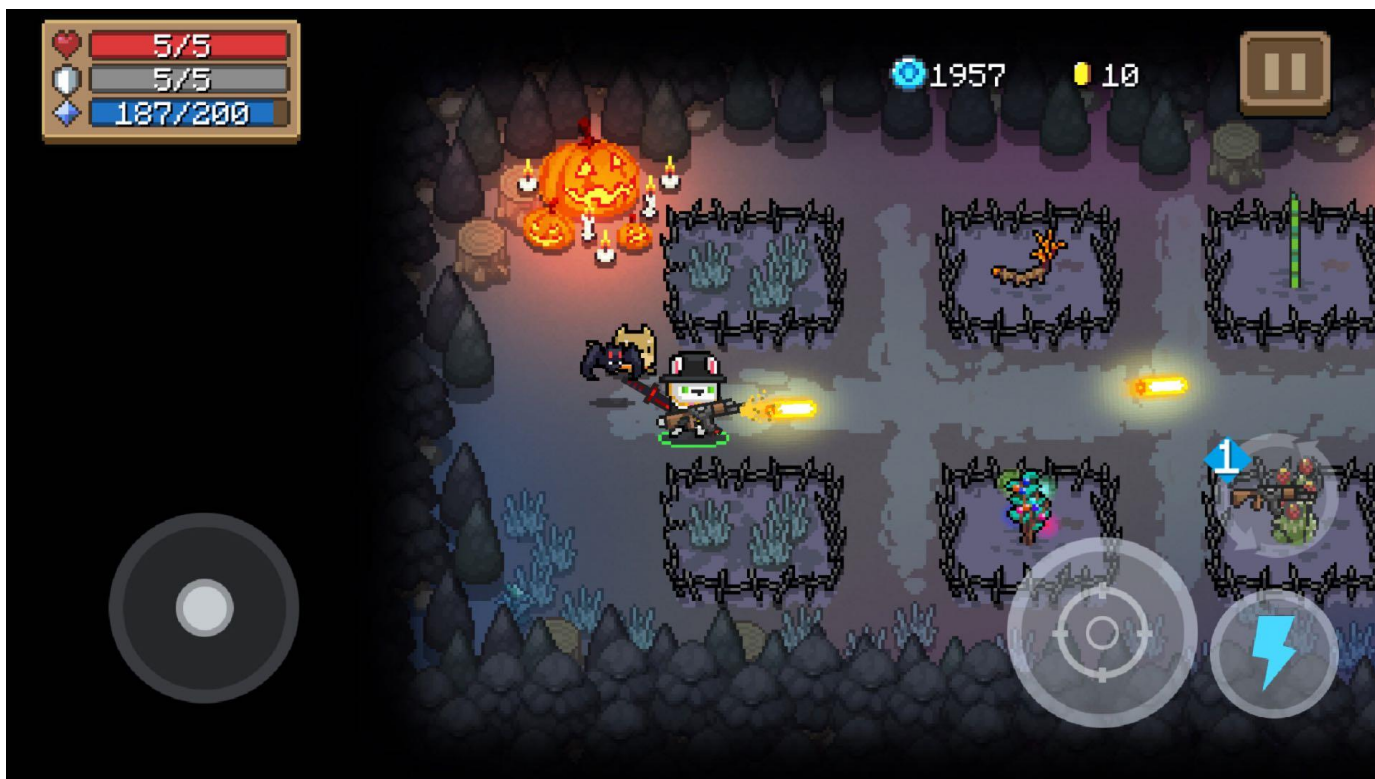
Contents

- ◆ 了解同源策略和跨域
- ◆ JSONP
- ◆ 案例-淘宝搜索
- ◆ 防抖和节流

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。



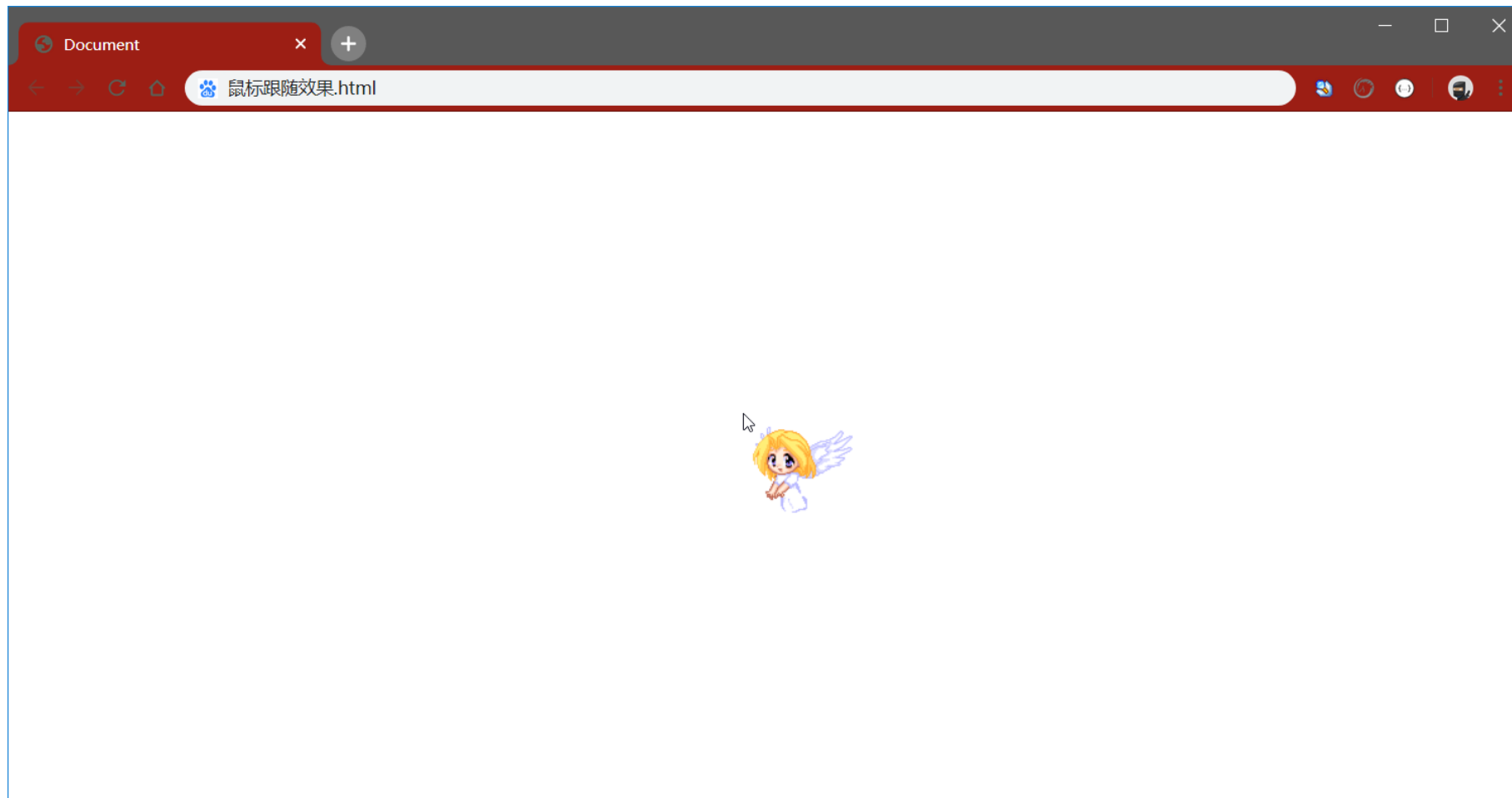
4. 防抖和节流

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），只在单位时间内只触发一次；
- 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. 防抖和节流

4.3 节流案例 – 鼠标跟随效果



4. 防抖和节流

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 -->


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

4. 防抖和节流

4.3 节流案例 – 鼠标跟随效果

2. 不使用节流时实现鼠标跟随效果

```
$(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')  
    })  
})
```

4. 防抖和节流

4.3 节流案例 – 鼠标跟随效果

3. 节流阀的概念

高铁卫生间是否被占用，由红绿灯控制，**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. 防抖和节流

4.3 节流案例 – 鼠标跟随效果

4. 使用节流优化鼠标跟随效果

```
$(function() {  
    var angel = $('#angel')  
    var timer = null // 1.预定义一个 timer 节流阀  
    $(document).on('mousemove', 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')  
            timer = null // 2.当设置了鼠标跟随效果后，清空 timer 节流阀，方便下次开启延时器  
        }, 16)  
    })  
})
```

4. 防抖和节流

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**只有最有一次触发生效**！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！



传智播客旗下高端IT教育品牌

HTTP协议加强

目录

Contents

- ◆ HTTP协议简介
- ◆ HTTP请求
- ◆ HTTP响应
- ◆ HTTP请求方法
- ◆ HTTP响应状态代码

1. HTTP协议简介

1.1 什么是通信

通信，就是**信息的传递和交换**。

通信三要素：

- 通信的**主体**
- 通信的**内容**
- 通信的**方式**

1. HTTP协议简介

1.1 什么是通信

1. 现实生活中的通信

案例：张三要把自己考上传智专修学院的好消息写信告诉自己的好朋友李四。

其中：

通信的主体是张三和李四；

通信的内容是考上传智专修学院；

通信的方式是写信；

1. HTTP协议简介

1.1 什么是通信

2. 互联网中的通信

案例：服务器把传智专修学院的简介通过响应的方式发送给客户端浏览器。

其中，

通信的主体是服务器和客户端浏览器；

通信的内容是传智专修学院的简介；

通信的方式是响应；

1. HTTP协议简介

1.2 什么是通信协议

通信协议（Communication Protocol）是指通信的双方完成通信所**必须遵守**的**规则和约定**。

通俗的理解：通信双方**采用约定好的格式**来发送和接收消息，这种**事先约定好的通信格式**，就叫做通信协议。

1. HTTP协议简介

1.2 什么是通信协议

1. 现实生活中的通信协议

张三与李四采用写信的方式进行通信，在填写信封时，写信的双方需要遵守固定的规则。**信封的填写规则**就是一种通信协议。

1	5	0	0	5	6	← 这里写收信人邮编		贴 邮 票 处
黑龙江省哈尔滨市道外区 ← 收信人地址, 地址要详细, 所以可能要多占两行或者三行								
王某 ← 收信人姓名, 字体稍大一些								
黑龙江省哈尔滨市道外区 ← 寄信人地址、姓名。								
							邮政编码 100053	
							←	

1. HTTP协议简介

1.2 什么是通信协议

2. 互联网中的通信协议

客户端与服务器之间要实现网页内容的传输，则通信的双方必须遵守网页内容的传输协议。

网页内容又叫做超文本，因此网页内容的传输协议又叫做超文本传输协议（HyperText Transfer Protocol），简称 HTTP 协议。

1. HTTP协议简介

1.3 HTTP

1. 什么是HTTP协议

HTTP 协议即超文本传送协议 (HyperText Transfer Protocol)，它规定了客户端与服务器之间进行网页内容传输时，所必须遵守的传输格式。

例如：

- **客户端**要以HTTP协议要求的格式把数据**提交**到**服务器**
- **服务器**要以HTTP协议要求的格式把内容**响应**给**客户端**


1. HTTP协议简介

1.3 HTTP

2. HTTP协议的交互模型

HTTP 协议采用了 **请求/响应** 的交互模型。





录Contents

- ◆ HTTP协议简介
- ◆ HTTP请求消息
- ◆ HTTP响应消息
- ◆ HTTP请求方法
- ◆ HTTP响应状态代码

2. HTTP请求消息

2.1 什么是HTTP请求消息

由于 HTTP 协议属于客户端浏览器和服务器之间的通信协议。因此，客户端发起的请求叫做 **HTTP 请求**，客户端发送到服务器的消息，叫做 **HTTP 请求消息**。

注意：HTTP 请求消息又叫做 HTTP 请求报文。

2. HTTP请求消息

2.2 HTTP请求消息的组成部分

HTTP 请求消息由请求行（request line）、请求头部（header）、空行和请求体 4 个部分组成。



2. HTTP请求消息

2.2 HTTP请求消息的组成部分

1. 请求行

请求行由请求方式、URL 和 HTTP 协议版本 3 个部分组成，他们之间使用空格隔开。



2. HTTP请求消息

2.2 HTTP请求消息的组成部分

2. 请求头部

请求头部用来描述客户端的基本信息，从而把客户端相关的信息告知服务器。比如：[User-Agent](#) 用来说明当前是什么类型的浏览器；[Content-Type](#) 用来描述发送到服务器的数据格式；[Accept](#) 用来描述客户端能够接收什么类型的返回内容；[Accept-Language](#) 用来描述客户端期望接收哪种人类语言的文本内容。

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。

头部字段名称	:(冒号)	值	回车符	换行符
.....				
头部字段名称	:(冒号)	值	回车符	换行符

2. HTTP请求消息

2.2 HTTP请求消息的组成部分

2. 请求头部 – 常见的请求头字段

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

2. HTTP请求消息

2.2 HTTP请求消息的组成部分

2. 请求头部 – 常见的请求头字段

```
▼ Request Headers    view parsed
POST /api/post HTTP/1.1
Host: ajax.frontend.itheima.net:3006
Connection: keep-alive
Content-Length: 14
Accept: */*
Origin: null
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
```

关于更多请求头字段的描述，可以查看 MDN 官方文档：<https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers>

2. HTTP请求消息

2.2 HTTP请求消息的组成部分

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**请求头部**与**请求体**。



2. HTTP请求消息

2.2 HTTP请求消息的组成部分

4. 请求体

请求体中存放的，是要通过 **POST 方式**提交到服务器的数据。

头部字段名称	:(冒号)	值	回车符	换行符
空行(回车符 或 换行符)				
请求体				

注意：只有 POST 请求才有请求体，GET 请求没有请求体！

2. HTTP请求消息

2.2 HTTP请求消息的组成部分

5. 总结



目录

Contents

- ◆ HTTP协议简介
- ◆ HTTP请求消息
- ◆ HTTP响应消息
- ◆ HTTP请求方法
- ◆ HTTP响应状态代码

3. HTTP响应消息

3.1 什么是HTTP响应消息

响应消息就是服务器响应给客户端的消息内容，也叫作响应报文。

3. HTTP响应消息

3.2 HTTP响应消息的组成部分

HTTP响应消息由**状态行**、**响应头部**、**空行**和**响应体**4个部分组成，如下图所示：



3. HTTP响应消息

3.2 HTTP响应消息的组成部分

1. 状态行

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；



▼ Response Headers [view parsed](#)

HTTP/1.1 200 OK

X-Powered-By: Express

Access-Control-Allow-Origin: *

Content-Type: application/json; charset=utf-8

Content-Length: 68

ETag: W/"44-nT/y6yOFj7H40EVW1DWB1MG+Pq0"

Date: Wed, 27 Nov 2019 01:48:57 GMT

Connection: keep-alive

3. HTTP响应消息

3.2 HTTP响应消息的组成部分

2. 响应头部

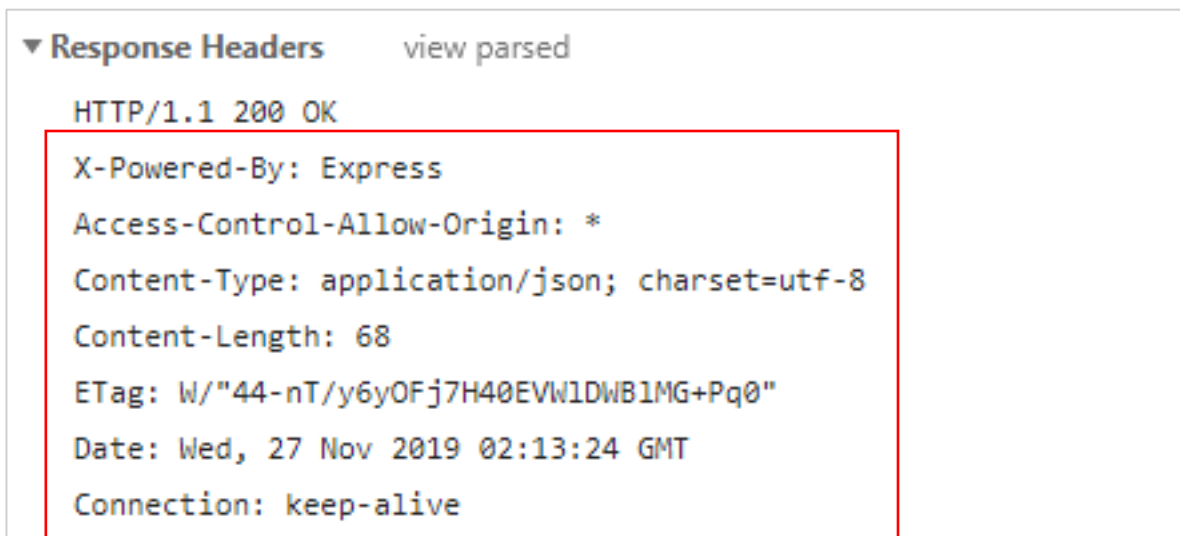
响应头部用来描述**服务器的基本信息**。响应头部由多行 **键/值对** 组成，每行的键和值之间用英文的冒号分隔。

头部字段名称	:(冒号)	值	回车符	换行符
.....				
头部字段名称	:(冒号)	值	回车符	换行符

3. HTTP响应消息

3.2 HTTP响应消息的组成部分

2. 响应头部 – 常见的响应头字段



关于更多响应头字段的描述，可以查看 MDN 官方文档：<https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers>

3. HTTP响应消息

3.2 HTTP响应消息的组成部分

3. 空行

在最后一个响应头部字段结束之后，会紧跟一个**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**。

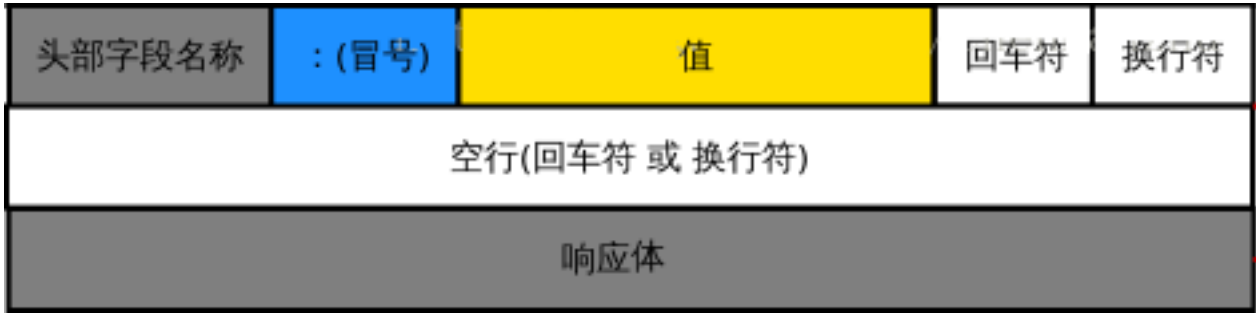
头部字段名称	:(冒号)	值	回车符	换行符
空行(回车符 或 换行符)				
响应体				

3. HTTP响应消息

3.2 HTTP响应消息的组成部分

4. 响应体

响应体中存放的，是服务器响应给客户端的资源内容。



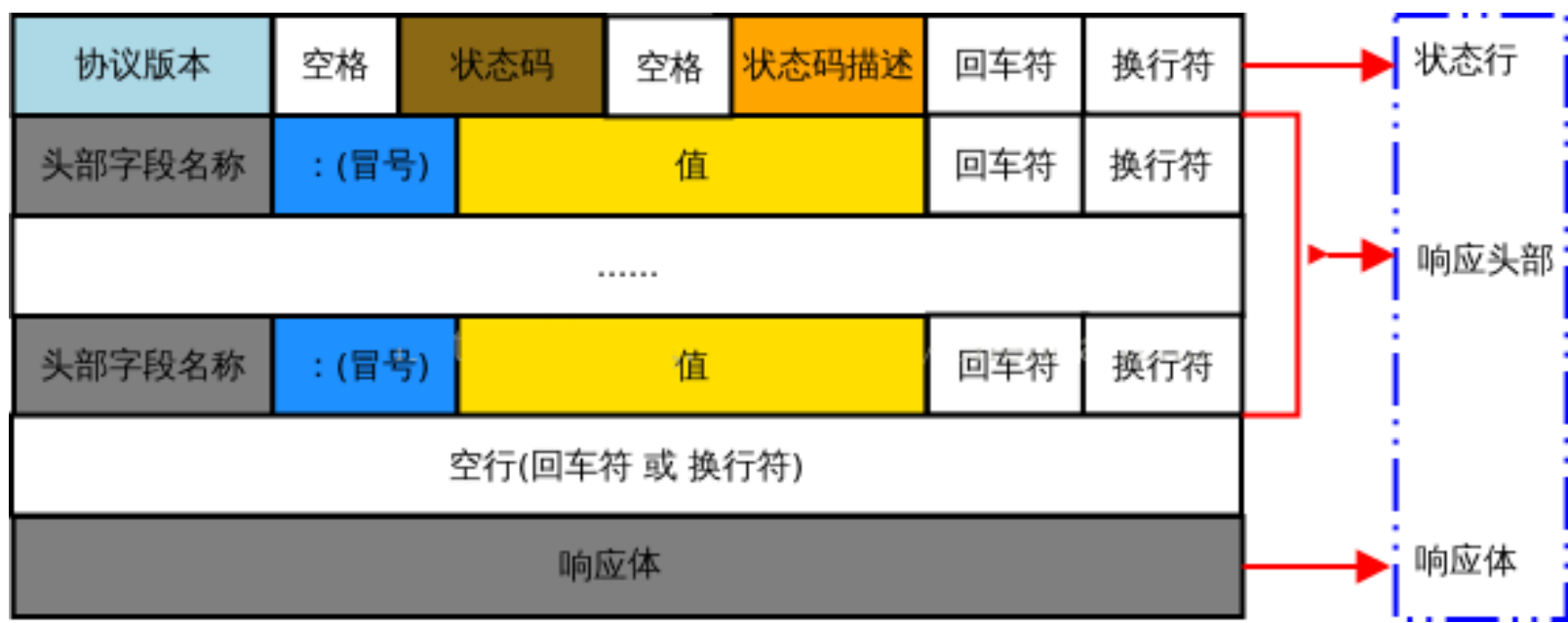
HeadersPreviewResponseTiming

```
{"message": "POST请求测试成功", "data": {"name": "zs", "age": "20"}}
```

3. HTTP响应消息

3.2 HTTP响应消息的组成部分

5. 总结



目录

Contents

- ◆ HTTP协议简介
- ◆ HTTP请求消息
- ◆ HTTP响应消息
- ◆ HTTP请求方法
- ◆ HTTP响应状态代码

4. HTTP请求方法

4.1 什么是HTTP请求方法

HTTP 请求方法，属于 HTTP 协议中的一部分，请求方法的作用是：用来表明**要对服务器上的资源执行的操作**。最常用的请求方法是 GET 和 POST。

4. HTTP请求方法

4.2 HTTP的请求方法

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。

目录

Contents

- ◆ HTTP协议简介
- ◆ HTTP请求消息
- ◆ HTTP响应消息
- ◆ HTTP请求方法
- ◆ HTTP响应状态代码

5. HTTP响应状态码

5.1 什么是HTTP响应状态码

HTTP 响应状态码（HTTP Status Code），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。

▼ Response Headers view parsed

HTTP/1.1 **200** OK

X-Powered-By: Express

Access-Control-Allow-Origin: *

Content-Type: application/json; charset=utf-8

Content-Length: 68

ETag: W/"44-nT/y6yOFj7H40EVW1DWB1MG+Pq0"

Date: Wed, 27 Nov 2019 02:13:24 GMT

Connection: keep-alive

5. HTTP响应状态码

5.2 HTTP响应状态码的组成及分类

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。

HTTP 状态码共分为 5 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

完整的 HTTP 响应状态码，可以参考 MDN 官方文档 <https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status>

5. HTTP响应状态码

5.3 常见的HTTP响应状态码

1. 2** 成功相关的响应状态码

2** 范围的状态码，表示服务器已成功接收到请求并进行处理。常见的 2** 类型的状态码如下：

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

5. HTTP响应状态码

5.3 常见的HTTP响应状态码

2. 3** 重定向相关的响应状态码

3** 范围的状态码，表示表示服务器要求客户端重定向，需要客户端进一步的操作以完成资源的请求。常见的 3** 类型的状态码如下：

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。

5. HTTP响应状态码

5.3 常见的HTTP响应状态码

3. 4** 客户端错误相关的响应状态码

4** 范围的状态码，表示客户端的请求有非法内容，从而导致这次请求失败。常见的 4** 类型的状态码如下：

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。

5. HTTP响应状态码

5.3 常见的HTTP响应状态码

4. 5** 服务端错误相关的响应状态码

5** 范围的状态码，表示服务器未能正常处理客户端的请求而出现意外错误。常见的 5** 类型的状态码如下：

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。



传智播客旗下高端IT教育品牌