目标

- 能够知道和服务器相关的基本概念
- 能够知道客户端和服务器通信的过程
- 能够知道数据也是一种资源
- 能够说出什么是`Ajax`以及应用场景
- 能够使用`jQuery`中的`Ajax`函数请求数据
- 能够知道接口和接口文档的概念

相关概念

客户端与服务器（☆☆☆）

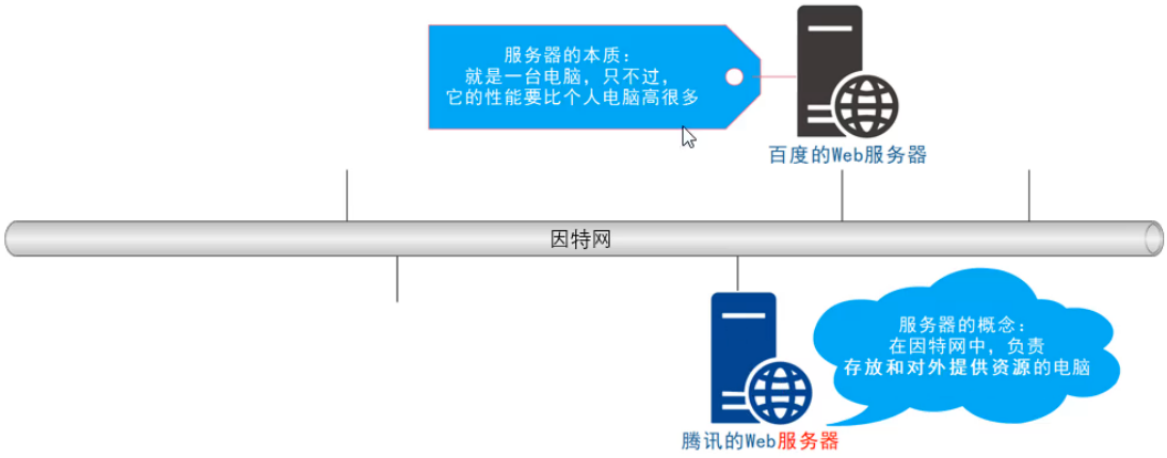
上网的目的

- 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**

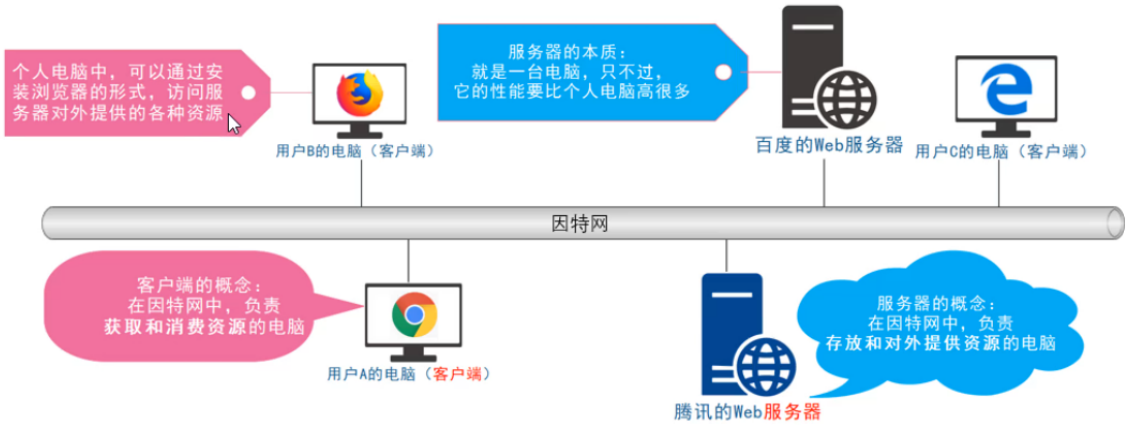
服务器

上网过程中，负责 **存放和对外提供资源** 的电脑，叫做服务器



客户端

在上网过程中，负责 **获取和消费资源** 的电脑，叫做客户端



URL地址的概念&组成

URL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

URL的组成

URL地址一般由三部分组成:

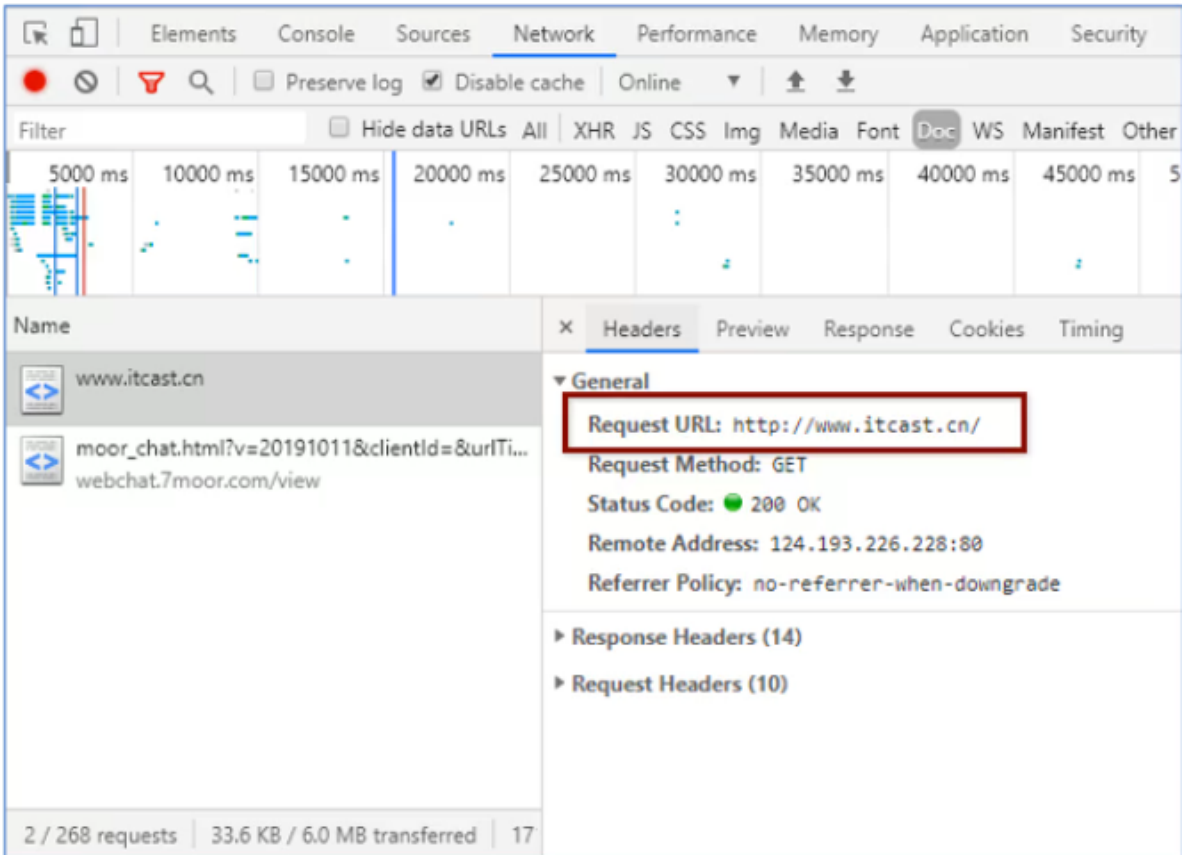
- 客户端与服务器之间的 **通信协议**
- 存有该资源的 **服务器名称**
- 资源在服务器上 **具体的存放位置**

客户端与服务器通讯过程（☆☆☆）

注意:

- 客户端与服务器之间的通讯过程，分为： **请求-处理-响应** 三个步骤
- 网页中每一个资源，都是通过 **请求-处理-响应** 的方式从服务器获取回来的

基于浏览器工具分析通讯过程



步骤：

- 打开 `Chorme` 浏览器
- `Ctrl + Shift + I（F12）` 打开 Chrome 的开发者工具
- 切换到 `Network` 面板
- 选中 `Doc` 页签
- 刷新页面，分析客户端与服务器的通讯过程

服务器对外提供的资源

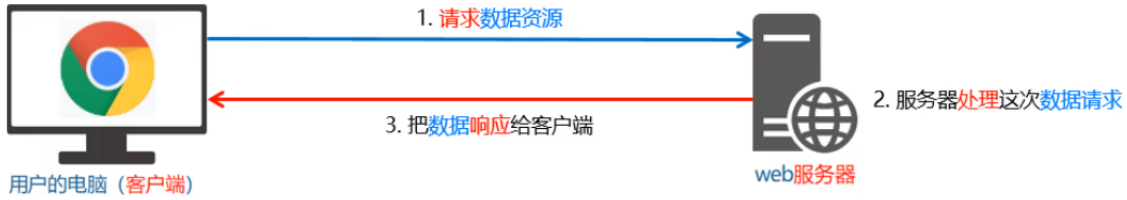
常见资源

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

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，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()`

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是 `get` 和 `post` 请求

- `get 请求`，通常用于 **获取服务器资源**（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**（送资源）

  例如：登录时，向服务器 **提交登录信息**、注册时向服务器 **提交注册信息**、添加用户时向服务器 **提交用户信息**等各种 **数据提交操作**

了解Ajax

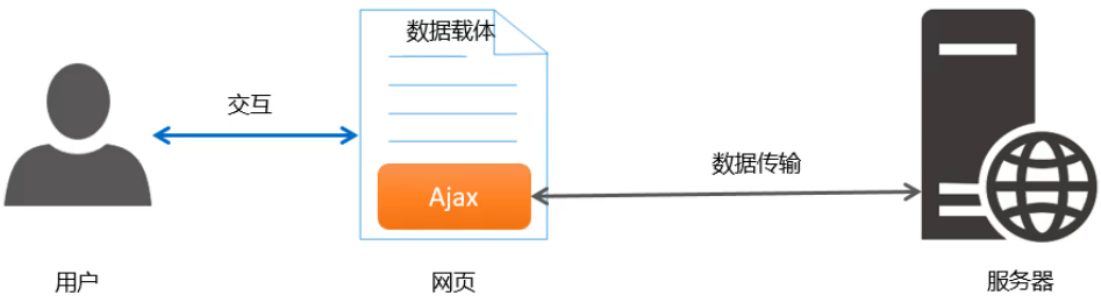
什么是Ajax（☆☆☆）

`Ajax` 的全称是 `Asynchronous JavaScript And XML`（异步 `JavaScript` 和 `xml`）

通俗理解：在网页中利用 `XMLHttpRequest` 对象和服务器进行数据交互的方式，就是 `Ajax`

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，但还是，`Ajax` 能让我们轻松实现 **网页** 与 **服务器**之间的 **数据交互**



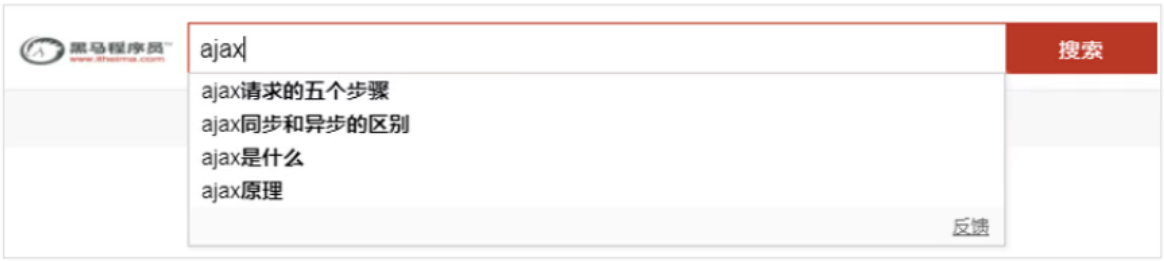
Ajax应该用场景（☆☆☆）

场景一：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**

场景二：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**



场景三：数据分页显示

当点击页码值得时候，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

场景四：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

`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：

- `$.get()` get方式请求，用于获取数据
- `$.post()` post方式请求，用于提交数据
- `$.ajax()` 比较综合，既可以获取数据，又可以提交数据

$.get() 函数介绍（☆☆）

`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

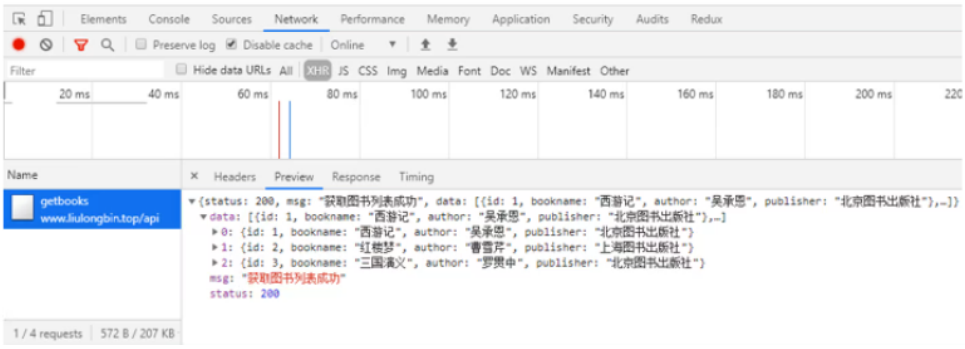
```
$.get(url,[data],[callback])
```

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

$.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，直接提供给 **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

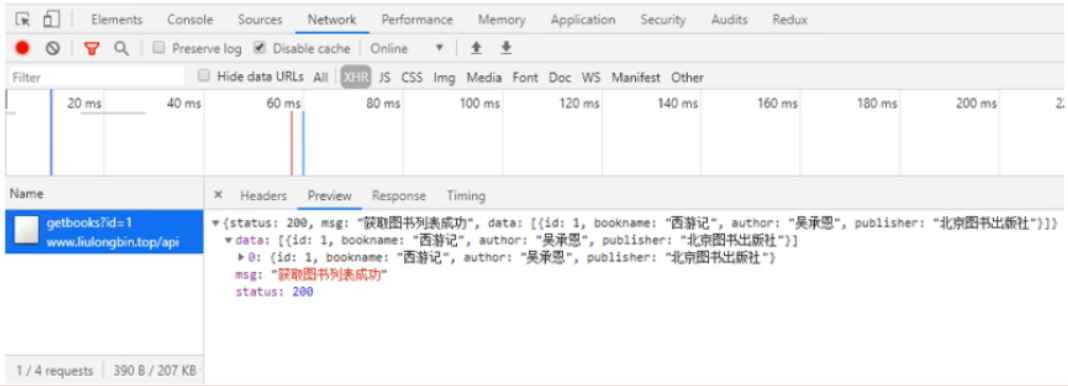
```
$.get('http://www.liulongbin.top:3006/api/getbooks', function(res) {
    console.log(res) // 这里的 res 是服务器返回的数据
})
```



$.get()发起携带参数的请求

使用 `$.get()` 发起携带参数的请求，那么携带的参数应该写在第二个参数的位置，示例代码如下：

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



$.post() 函数介绍（☆☆）

`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])
```

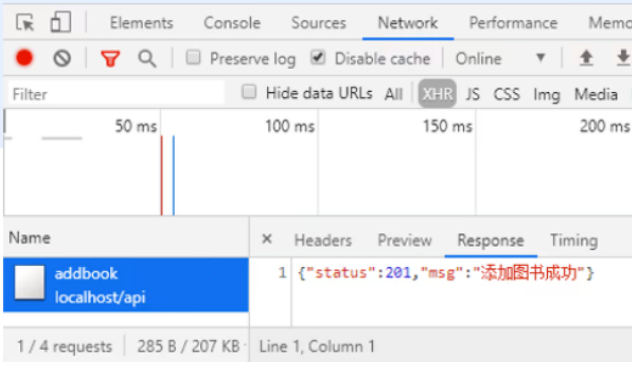
参数各自代表的含义如下：

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

$.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)
    }
)
```



`$.ajax()` 函数介绍（☆☆☆）

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
})
```

`$.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)
    }
})
```

`$.ajax` 发起 post 请求

使用 `$.ajax()` 发起 `post` 请求，只需要把 `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)
    }
})
```


接口

接口的概念（☆☆☆）

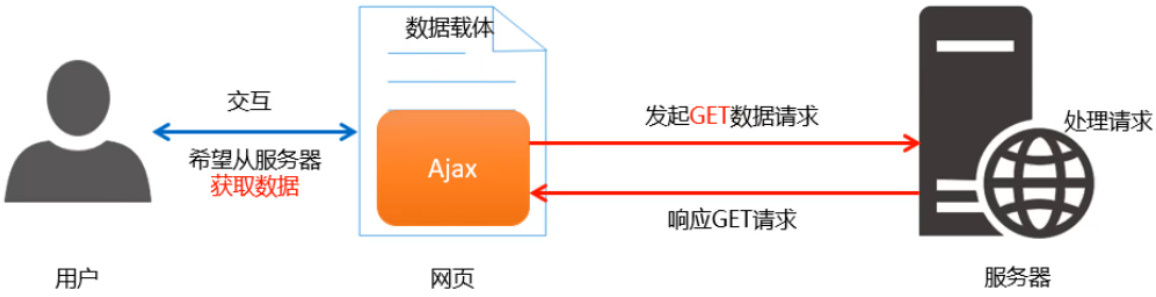
使用 `Ajax` 请求数据时，被请求的 `URL` 地址，就叫做 数据接口（简称**接口**）。同时，每个接口必须有请求方式。

例如：

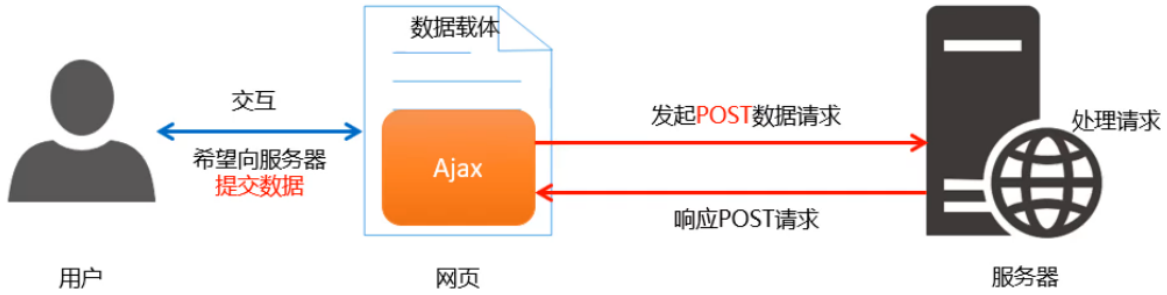
```
http://www.liulongbin.top:3006/api/getbooks 获取图书列表的接口（get请求）
http://www.liulongbin.top:3006/api/addbook  添加图书的接口（post请求）
```

接口的请求过程

GET方式请求接口的过程



POST方式请求接口的过程



接口测试工具

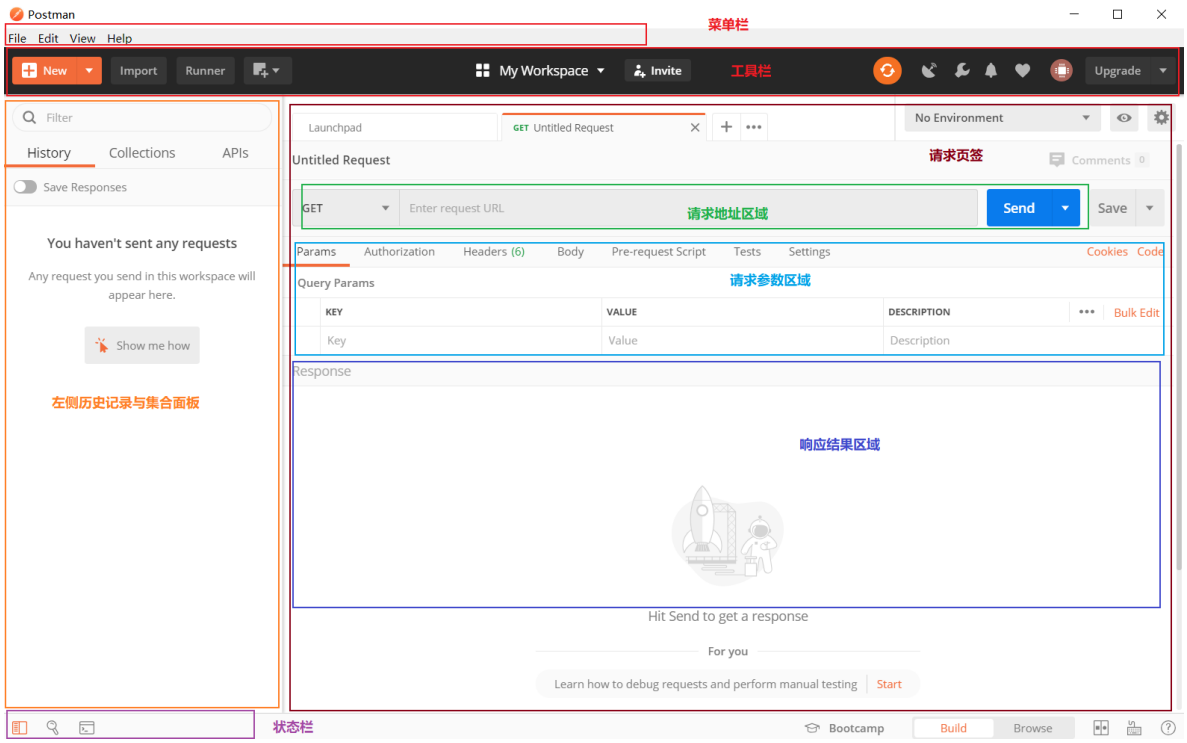
什么是接口测试工具

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

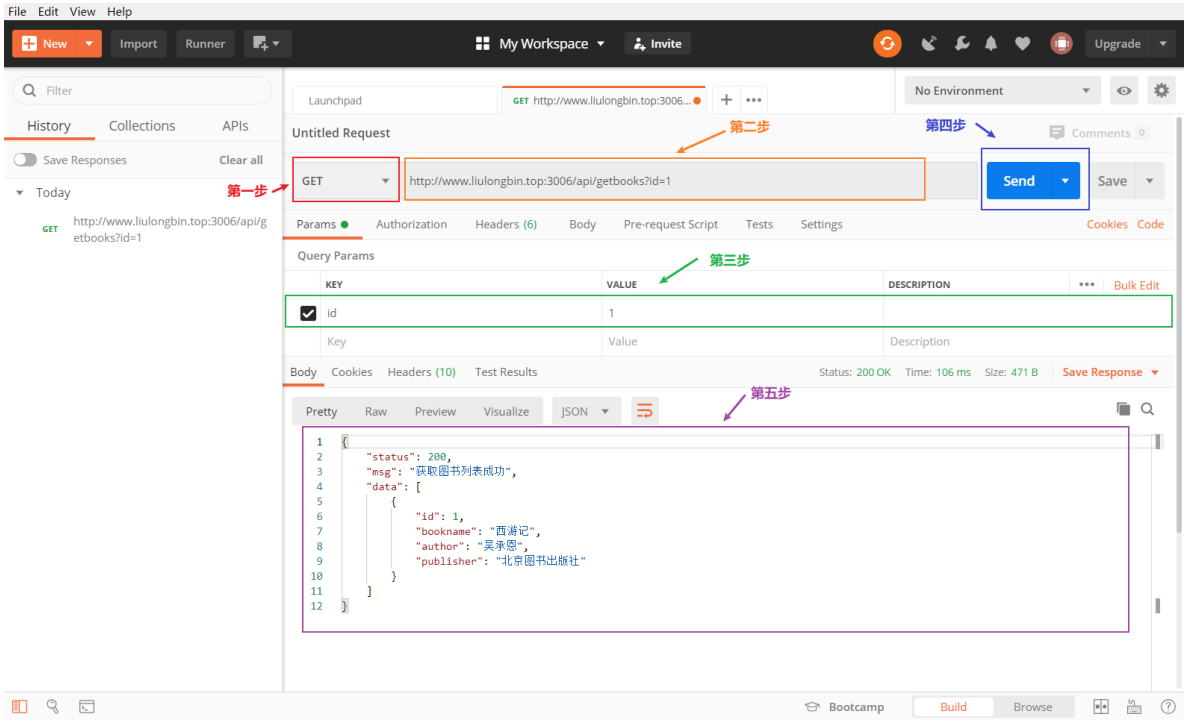
好处：接口测试工具能让我们在 **不写任何代码** 的情况下，对接口进行 **调用** 和 **测试**

常用的就是：PostMan

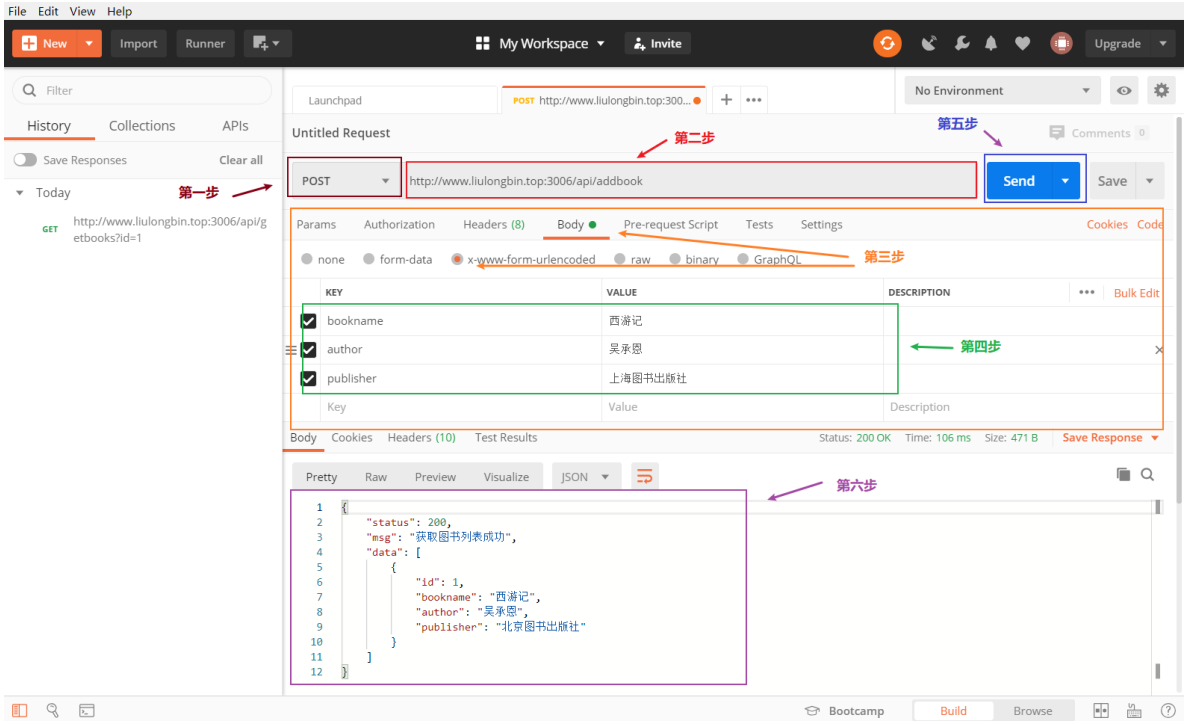
了解 Postman 界面结构



使用PostMan测试GET接口



使用 PostMan 测试POST接口



接口文档

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

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：

- **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
- **接口URL：**接口的调用地址
- **调用方式：**接口的调用方式，如 **GET** 或者 **POST**
- **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
- **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
- **返回示例（可选）：**通过对象的形式，列举服务器返回数据的结构

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

- 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

- 返回示例：

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

案例

图书管理

项目效果

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
6	钉钉	的撒大	的撒	删除
7	abc	abc	abc	删除
8	天才在左,疯子在右	高铭	人民出版社	删除

UI界面搭建

需要使用到的库和插件

- 用到的 `css` 库 `bootstrap.css`
- 用到的 `javascript` 库 `jquery.js`
- 用到 `vs code` 插件 `Bootstrap 3 Snippets`

搭建步骤

- Panel面板搭建
  - 创建panel板 （快捷键：`bs3-panel:primary`）
  - 在 `panel-body` 里面，创建3个对应的输入框 （快捷键：`bs3-input:addon:text`），对应修改标题
  - 在 `panel-body` 最后面，创建 `button` 按钮 （快捷键：`bs3-button:primary`）,修改内容
- 图书的表格
  - 创建 table（快捷键：`bs3-table:bordered`）
  - 在里面创建对应5个`td`，填写里面内容

获取图书列表数据

步骤：

- 查阅资料中的接口文档，找到获取图书列表的接口
- 定义 script 标签，创建入口函数
- 利用 $.get() 方法，传入相应的`url`，和成功之后的回调函数
- 在回调函数中获取到请求成功的数据

```
// 获取图书列表数据
function getBookList() {
    $.get('http://www.liulongbin.top:3006/api/getbooks', function(res) {
        console.log(res)
    })
}
```

渲染图书列表

步骤：

- 根据返回状态码来判断是否成功请求到数据
- 创建数组，用来存放**行**数据（`rows`）
- 遍历服务器返回的数组，每遍历一次，利用数组（`rows`）去`push` `<tr></tr>`
- 每一行 `tr` 里面包含了5个`td`
- 给每一个 `td` 设置对应内容即可
- 遍历循环完毕之后，找到内容容器，先清空当前内容（以免有重复数据），然后添加 `rows`

```
var rows = []
$.each(res.data, function(i, item) {
     rows.push('<tr><td>' + item.id + '</td><td>' + item.bookname + '</td><td>'
+ item.author + '</td><td>' + item.publisher + '</td><td><a href="javascript:;"
class="del" data-id="' + item.id + '">删除</a></td></tr>')
})
$('#tb').empty().append(rows.join(''))
```

删除功能实现

删除链接绑定单击事件处理函数

- 利用 `tbody` 容器，通过**事件委派**的方式，给动态创建的`a`标签绑定事件
- 删除图书需要通过`id`删除，所以我们需要得到对应的`id`，我们利用自定义属性的方式，传递过来相应的`id`

删除功能实现

- 查阅**删除**的接口文档
- 在a标签点击事件处理函数里面利用`$.get()`方法，请求服务器，传入要删除的对应`id`
- 删除成功之后，调用 `getBookList()` 刷新页面

```
$('tbody').on('click', '.del', function() {
  var id = $(this).attr('data-id')
  $.get('http://www.liulongbin.top:3006/api/delbook', {
    id: id
  }, function(res) {
    if (res.status !== 200) return alert('删除图书失败！')
    getBookList()
   })
})
```

添加功能实现

添加按钮绑定点击事件

- 获取三个输入框的内容
- 判断三个输入框是否输入了内容，如果没有进行提示

实现图书添加功能

- 查阅接口文档
- 注意接口文档需要提交的参数名，我们需要保持一致

- 调用 `$.post()` 方法，传入请求路径，然后组拼需要提交的参数
- 在成功回调里面判断返回值是否是201，如果是201代表成功，反之没有成功，进行提示
- 请求成功之后，调用 `gettBookList()` 方法刷新页面，同时把输入框里面值清空

```
$('#btnAdd').on('click', function() {
     var bookname = $('#iptBookname').val().trim()
     var author = $('#iptAuthor').val().trim()
     var publisher = $('#iptPublisher').val().trim()
     if (bookname.length <= 0 || author.length <= 0 || publisher.length <= 0) {
         return alert('请填写完整的图书信息！')
     }

     $.post('http://www.liulongbin.top:3006/api/addbook', {
         bookname: bookname,
         author: author,
         publisher: publisher
     }, function(res) {
         if (res.status !== 201) return alert('添加图书失败！')
         getBookList()
         $('#iptBookname').val('')
         $('#iptAuthor').val('')
         $('#iptPublisher').val('')
     })
})
```

聊天机器人

效果

实现功能点

- 梳理案例代码结构
- 将用户输入的内容渲染到聊天窗口
- 发起请求获取聊天消息
- 将机器人的聊天内容转为语音
- 通过 播放语音
- 使用回车发送消息

梳理案例的代码结构

- `UI`结构梳理

- 业务代码抽离
- `resetui()` 函数作用-让聊天框区域自动滚动到底部

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
- 在点击事件函数里面判断一下用户输入内容是否为空，注意：如果为空，我们清除一下输入框内容
- 获取到对应的 `ul` 容器，调用 `append` 函数来追加 `li`，注意：追加 `li` 的类名叫做 `right_word`
- 清除文本输入框的值
- 最后调用一下 `resetui()`，让聊天框区域自动滚动到底部

```
// 为发送按钮绑定鼠标点击事件
$('#btnSend').on('click', function () {
    var text = $('#ipt').val().trim()
    if (text.length <= 0) {
      return $('#ipt').val('')
    }
    // 如果用户输入了聊天内容，则将聊天内容追加到页面上显示
    $('#talk_list').append('<li class="right_word"><img src="img/person02.png"
/> <span>' + text + '</span></li>')
    $('#ipt').val('')
    // 重置滚动条的位置
    resetui()
})
```

发起请求获取聊天信息

- 定义一个函数 `getMsg()` 接收一个参数，参数就是用户发送的信息
- 利用 `$.ajax()` 发送一个 GET 方式请求，传入请求地址 `http://ajax.frontend.itheima.net:3006/api/robot`
- 定义请求数据 `spoken: value`
- 定义 `success` 成功的回调，在回调函数里面判断返回数据的 `message` 是否等于 `success`
- 给容器动态添加返回的内容

```
// 获取聊天机器人发送回来的消息
function getMsg(text) {
  $.ajax({
    method: 'GET',
    url: 'http://ajax.frontend.itheima.net:3006/api/robot',
    data: {
      spoken: text
```

```
      },
      success: function (res) {
        // console.log(res)
        if (res.message === 'success') {
          // 接收聊天消息
          var msg = res.data.info.text
          $('#talk_list').append('<li class="left_word"><img
src="img/person01.png" /> <span>' + msg + '</span></li>')
          // 重置滚动条的位置
          resetui()
        }
      }
    })
  }
```

将机器人聊天内容转成语音

- 封装函数 `getVoice()` 接收一个参数，机器人的聊天信息
- 利用 `$.ajax()` 发送一个 GET 方式请求，传入请求地址 `http://ajax.frontend.itheima.net:3006/api/synthesize`
- 定义请求数据 `text: value`
- 定义 `success` 成功的回调，判断返回的状态码是否是200，如果是代表成功
- 在页面上定义 `audio` 标签，设置隐藏，等数据返回之后，利用这个 `audio` 来进行播放。设置 `autoplay` 属性来进行自动播放

```
// 把文字转化为语音进行播放
  function getVoice(text) {
    $.ajax({
      method: 'GET',
      url: 'http://ajax.frontend.itheima.net:3006/api/synthesize',
      data: {
        text: text
      },
      success: function (res) {
        // console.log(res)
        if (res.status === 200) {
          // 播放语音
          $('#voice').attr('src', res.voiceUrl)
        }
      }
    })
  }
```

通过回车键发送消息

- 给文本输入框注册 `keyup` 事件，按键弹起的事件监听
- 在事件函数里面，通过 `keycode` 来获取对应的按键的 机器码
- 判断 `keycode` 是否等于 13（不需要去记忆，开发时候打印调试一下就行了），如果是，代表是回车键
- 如果是回车键，模拟用户点击： `$('#btnSend').click()`

```
// 为文本框绑定 keyup 事件
 $('#ipt').on('keyup', function (e) {
   // console.log(e.keyCode)
   if (e.keyCode === 13) {
     // console.log('用户弹起了回车键')
     $('#btnSend').click()
   }
 })
```