

服务器的基本概念与初识Ajax

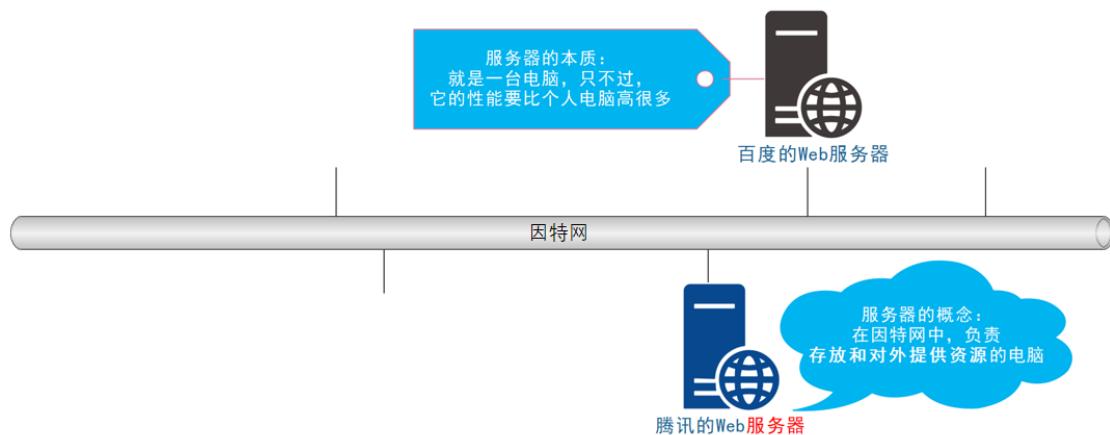
1. 客户端与服务器

1.1 上网的目的

上网的本质目的：通过互联网的形式来获取和消费资源

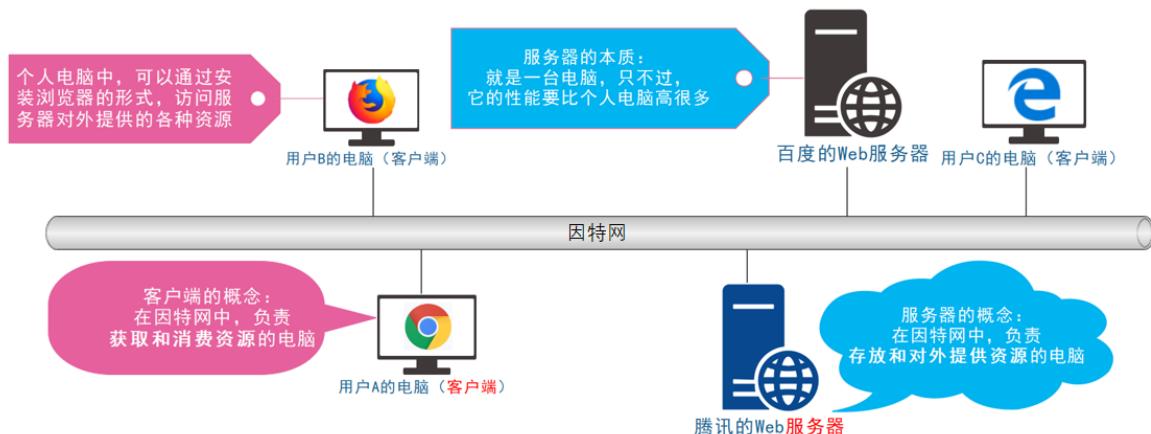
1.2 服务器

上网过程中，负责存放和对外提供资源的电脑，叫做服务器。



1.3 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。



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>

2.2 URL地址的组成部分

URL地址一般由三部组成：

- ① 客户端与服务器之间的通信协议
- 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



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



注意：

1. 客户端与服务器之间的通信过程，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。

3.2 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

Request URL: <http://www.itcast.cn/>

Request Method: GET

Status Code: 200 OK

Remote Address: 124.193.226.228:80

Referrer Policy: no-referrer-when-downgrade

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>传智播客官网-好口碑IT培训机构,一样的教育</title>
6 <meta content="IT培训,Java培训,人工智能培训,Python培训" name="keywords" />
7 <meta content="传智播客专注IT培训,Java培训,人工智能培训,Python培训" name="description" />
8
9   <link rel="stylesheet" href="/2018czgw/css/rule.css?ver=1.0.1" type="text/css" media="all" />
10  <link rel="stylesheet" href="/2018czgw/css/center.css?ver=1.0.1" type="text/css" media="all" />
11  <link rel="stylesheet" href="/2018czgw/css/about.css?ver=1.0.1" type="text/css" media="all" />
12  <link rel="stylesheet" href="/2018czgw/css/join.css?ver=1.0.1" type="text/css" media="all" />
13  <link rel="stylesheet" href="/2018czgw/css/join2.css?ver=1.0.1" type="text/css" media="all" />
14  <script type="text/javascript" src="/2018czgw/js/join.js?ver=1.0.1" type="text/javascript" />
15  <script type="text/javascript" src="http://www.itcast.cn/join2.js?ver=1.0.1" type="text/javascript" />
16  <style>
17    html{width: 100%;height: 520px;overflow-y: scroll;position: relative; font-size: 14px; margin: 0; padding: 0; border: none; background-color: #f5f5f5; font-family: "Microsoft YaHei", "SimHei", "SimSun", sans-serif; color: #333; }
18  </style>
```

4. 服务器对外提供了哪些资源

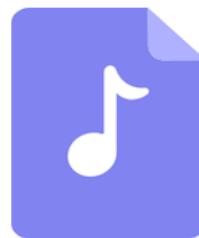
4.1 例举网页中常见的资源



文字内容



Image 图片



Audio 音频



Video 视频

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。

4.3 数据是网页的灵魂

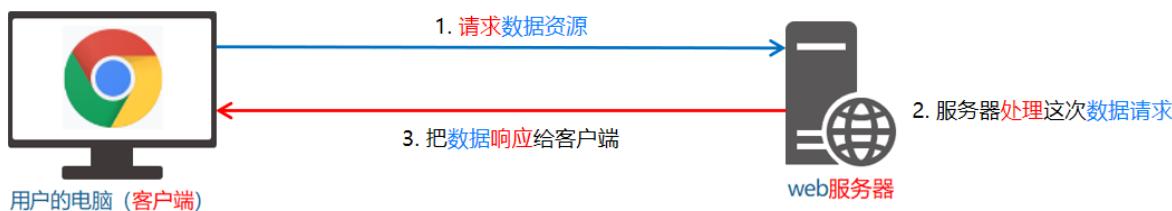
- HTML是网页的骨架
- CSS是网页的颜值
- Javascript是网页的行为
- 数据，则是网页的灵魂

骨架、颜值、行为皆为数据服务

数据，在网页中无处不在

4.4 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()

4.5 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

5. 了解Ajax

5.1 什么是Ajax

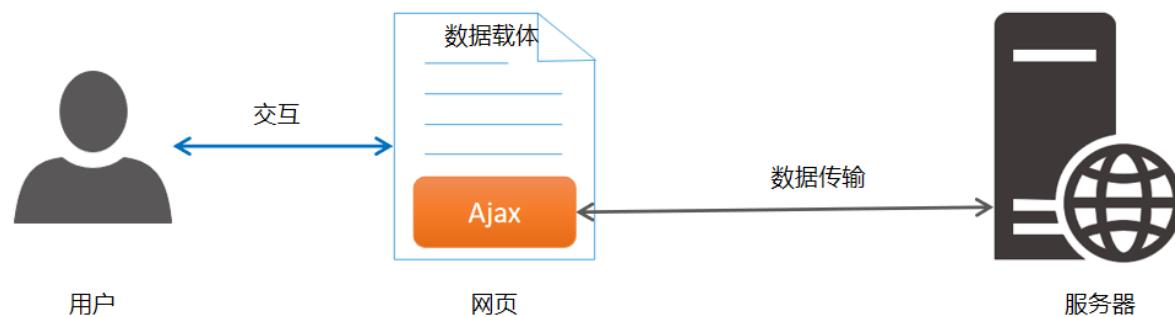
Ajax 的全称是 Asynchronous Javascript And XML (异步 JavaScript 和 XML) 。

通俗的理解：在网页中利用 XMLHttpRequest 对象和服务器进行数据交互的方式，就是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之间的数据交互。



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(注意是占用，这里已经到后端请求数据库里面的数据了)



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



ajax

搜索

ajax请求的五个步骤
ajax同步和异步的区别
ajax是什么
ajax原理

反馈

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 页

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	编辑 删除
2	热门推荐	✓	一级	编辑 删除
3	海外购	✓	一级	编辑 删除
4	苏宁房产	✓	一级	编辑 删除
5	手机相机	✓	一级	编辑 删除

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()`

6.2 \$.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])
```

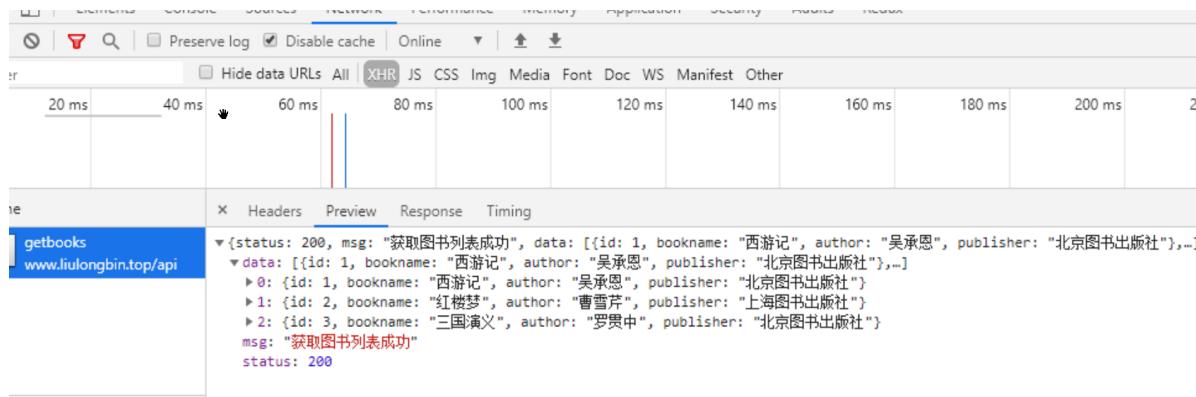
其中，三个参数各自代表的含义如下：

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

\$.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
})
```



\$.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)
})
```



6.3 \$.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])
```

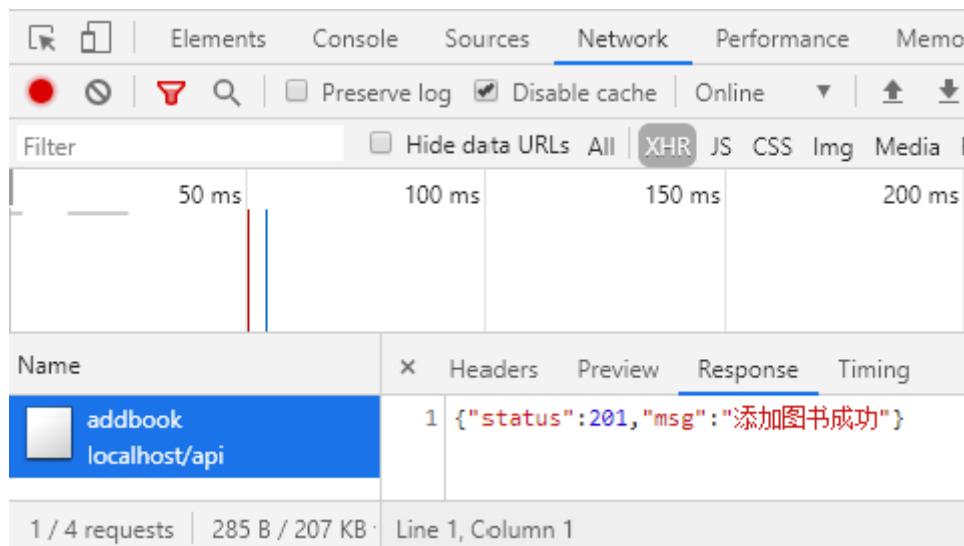
其中，三个参数各自代表的含义如下：

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

\$.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)
  }
)
```



6.4 \$.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
})
```

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)
  }
})
```

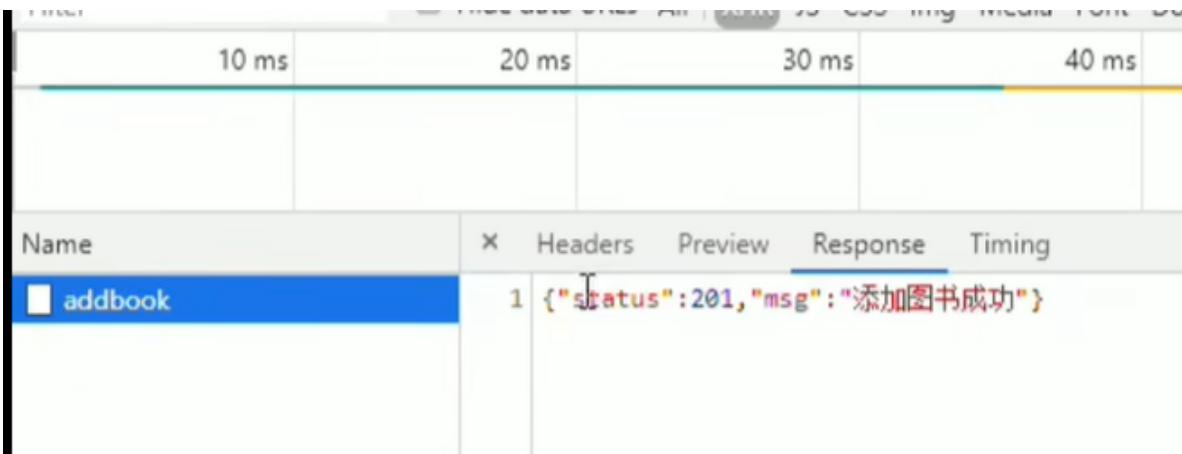


注意看参数拼接 在url后面拼接?id=1

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)
  }
})
```



这里的参数是在请求体里面

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)

7.2 分析接口的请求过程

通过GET方式请求接口的过程



通过POST方式请求接口的过程



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

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。

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

The screenshot shows the PostMan application window. On the left is the 'My Workspace' sidebar with sections for Collections, Environments, and History. The main workspace shows a 'SmartBus_Backend' collection with a 'POST http://localhost:8080/file/upload' request selected. The request details pane shows method 'POST', URL 'http://localhost:9090/file/upload', and a 'Body' tab selected. A file input field is set to 'file' with the value '新建 XLS 工作表.xls'. Below the request details is a 'Response' pane featuring a cartoon character and the text 'Click Send to get a response'.

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

The screenshot shows the Postman application interface. In the top navigation bar, 'Import' is selected. Below it, a collection named 'SmartBus_Backend' is expanded, showing a 'GET' request to 'http://localhost:9090/component/listAll'. The 'Params' tab is active, displaying a single parameter 'Key' with a value 'Value'. A red box highlights the 'Key' column. At the bottom right of the interface, there is a button labeled 'Send' with a red box around it, and a placeholder text 'Click Send to get a response'.

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

The screenshot shows the Postman application interface. In the top navigation bar, 'Import' is selected. Below it, a collection named 'SmartBus_Backend' is expanded, showing a 'POST' request to 'http://localhost:9090/component/save'. The 'Body' tab is active, showing the 'form-data' option selected. Under the 'Body' section, there is a key-value pair 'nameCh' with a value '中文名'. A red box highlights the 'nameCh' key. At the bottom right of the interface, there is a button labeled 'Send' with a red box around it, and a placeholder text 'Click Send to get a response'.

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。

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）：通过对对象的形式，例举服务器返回数据的结构。

3. 接口文档示例

请求的根路径

<http://www.liulongbin.top:3006>

图书列表

- 接口URL： /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：

```
JSON
{
  "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" }
  ]
}
```

添加图书

- 接口URL: /api/addbook
- 调用方式: POST
- 参数格式:

参数名称	参数类型	是否必选	参数说明
bookname	String	是	图书名称
author	String	是	作者
publisher	String	是	出版社

- 响应格式:

数据名称	数据类型	说明
status	Number	201 添加成功; 500 添加失败;
msg	String	对 status 字段的详细说明

- 返回示例:

```
JSON
{
    "status": 201,
    "msg": "添加图书成功"
}
```

删除图书

- 接口URL: /api/delbook
- 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	是	图书Id

- 响应格式:

数据名称	数据类型	说明
status	Number	200 删除成功; 500 未指定要删除的图书Id; 501 执行Sql报错; 502 要删除的图书不存在;
msg	String	对 status 字段的详细说明

- 返回示例:

```
JSON
{
    "status": 200,
    "msg": "删除成功！"
}
```

form表单与模板引擎

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。

```

    登录 · 注册

    手机号或邮箱
    密码

     记住我
    登录遇到问题?

    登录

    社交账号登录
       其它

```

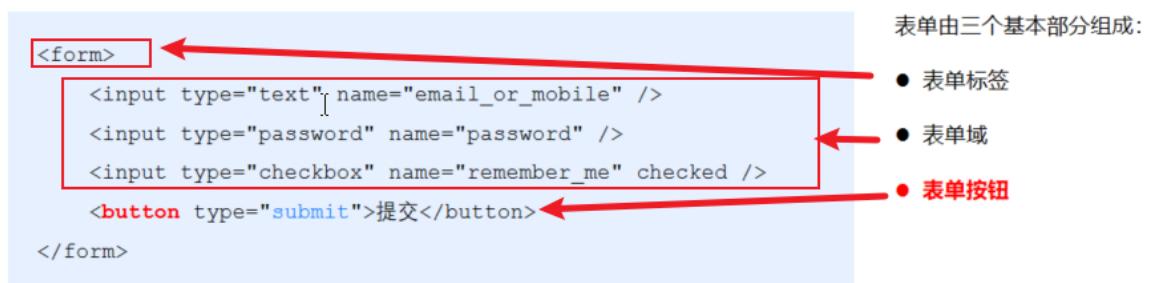
```

<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>

```

1.2 表单的组成部分

注意：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。



1.3 <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 framename	规定在何处打开 action URL

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(所以之后我们要禁用提交跳转)

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

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。

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)

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！

1.4 表单的同步提交及缺点

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。

表单同步提交的缺点

<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。

<form> 表单同步提交后，页面之前的状态和数据会丢失。

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。

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')  
})
```

2.2 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()  
})
```

2.3 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

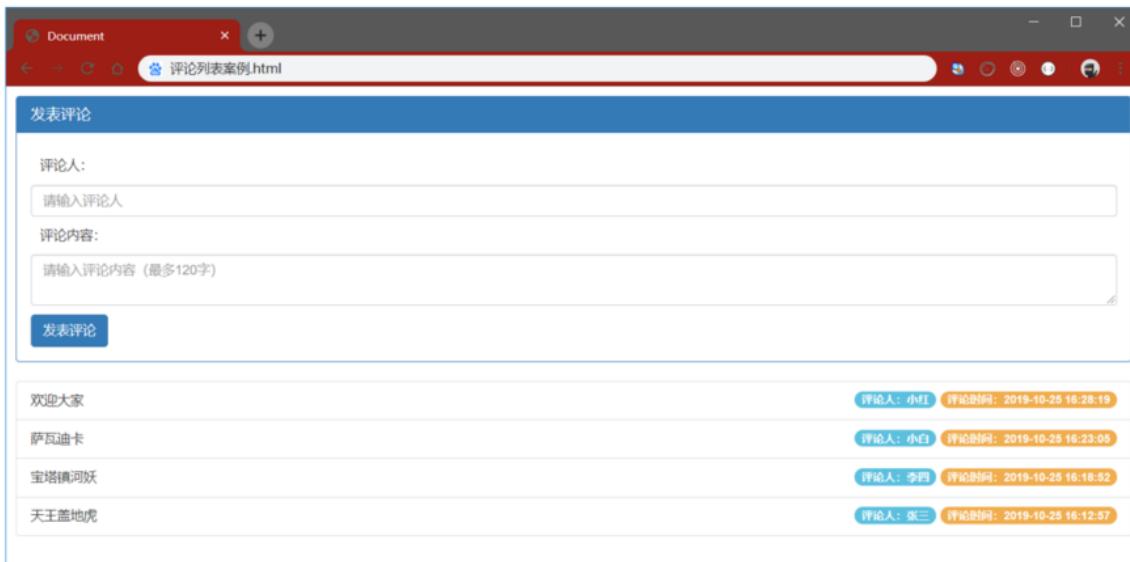
```
<form id="form1">
    <input type="text" name="username" />
    <input type="password" name="password" />
    <button type="submit">提交</button>
</form>
```

```
$('#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！

3. 案例 - 评论列表

3.1 渲染UI结构



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
    })
}

```

3.3 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()
    })
})

```

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。

4.2 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。



4.3 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

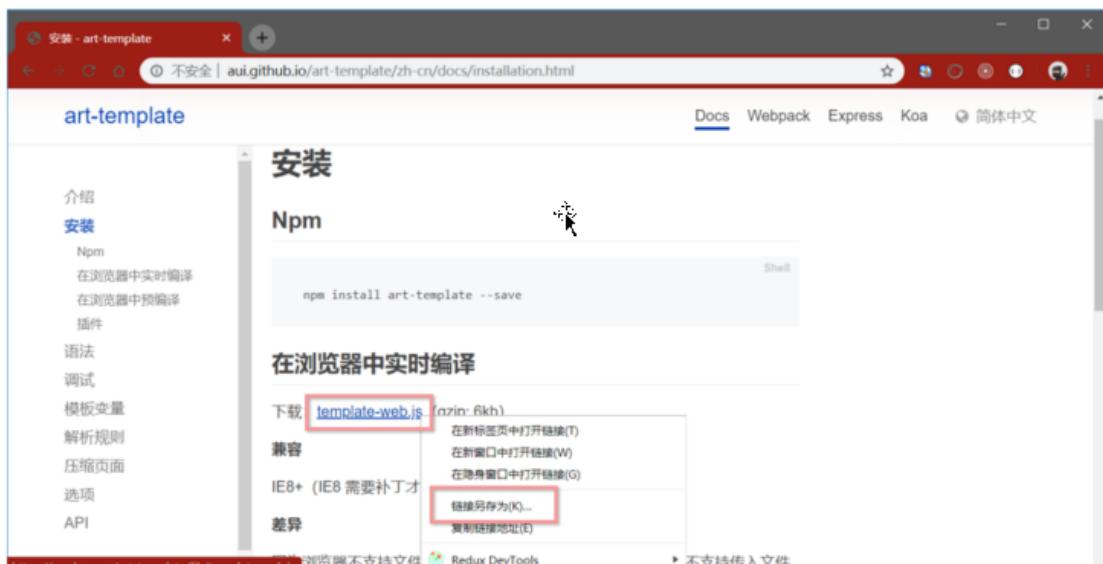
5. art-template模板引擎

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>

5.2 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。



5.3 art-template模板引擎的基本使用

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

```
<!DOCTYPE html>
<html lang="en">

<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>Document</title>
  <!-- 1. 导入模板引擎 -->
  <!-- 在 window 全局，多一个函数，叫做 template('模板的Id', 需要渲染的数据对象) -->
  <script src="./lib/template-web.js"></script>
  <script src="./lib/jquery.js"></script>
</head>

<body>

<div id="container"></div>

<!-- 3. 定义模板 -->
<!-- 3.1 模板的 HTML 结构，必须定义到 script 中 -->
<script type="text/html" id="tpl-user">
  <h1>{{name}} ----- {{age}}</h1>
  <!-- 原文输出 -->
  {{@ test}}


<div>
  {{if flag === 0}}
    flag的值是0
  {{else if flag === 1}}
    flag的值是1
  {{/if}}
</div>

<ul>
  {{each hobby}}
    <li>索引是:{{$index}}, 循环项是:{{$value}}</li>
  {{/each}}
</ul>

<h3>{{regTime | dateFormat}}</h3>
</script>

<script>
  // 定义处理时间的过滤器
  template.defaults.imports.dateFormat = function (date) {
    var y = date.getFullYear()
    var m = date.getMonth() + 1
    var d = date.getDate()

    return y + '-' + m + '-' + d
  }
</script>
```

```

// 2. 定义需要渲染的数据
var data = { name: 'zs', age: 20, test: '<h3>测试原文输出</h3>', flag: 1,
hobby: ['吃饭', '睡觉', '写代码'], regTime: new Date() }

// 4. 调用 template 函数
var htmlStr = template('tpl-user', data)
console.log(htmlStr)
// 5. 渲染HTML结构
$('#container').html(htmlStr)
</script>
</body>

</html>

```

显示效果：

ZS ----- 20

测试原文输出

flag的值是1

- 索引是:0, 循环项是:吃饭
- 索引是:1, 循环项是:睡觉
- 索引是:2, 循环项是:写代码

2023-5-26

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。

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。

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。

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. 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}}
```

6. 标准语法 - 过滤器



过滤器的本质，就是一个 function 处理函数。

```
{@ 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*/}
```

举例：

```
<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
}
```

6. 模板引擎的实现原理

```
<!DOCTYPE html>
<html lang="en">

<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>Document</title>
    <script src="./js/template.js"></script>
</head>

<body>

    <div id="user-box"></div>

    <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>

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

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

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

</html>
```

```

function template(id, data) {
    var str = document.getElementById(id).innerHTML
    var pattern = /{{\s*([a-zA-Z]+)\s*}}/

    var pattResult = null
    while (pattResult = pattern.exec(str)) {
        str = str.replace(pattResult[0], data[pattResult[1]])
    }

    return str
}

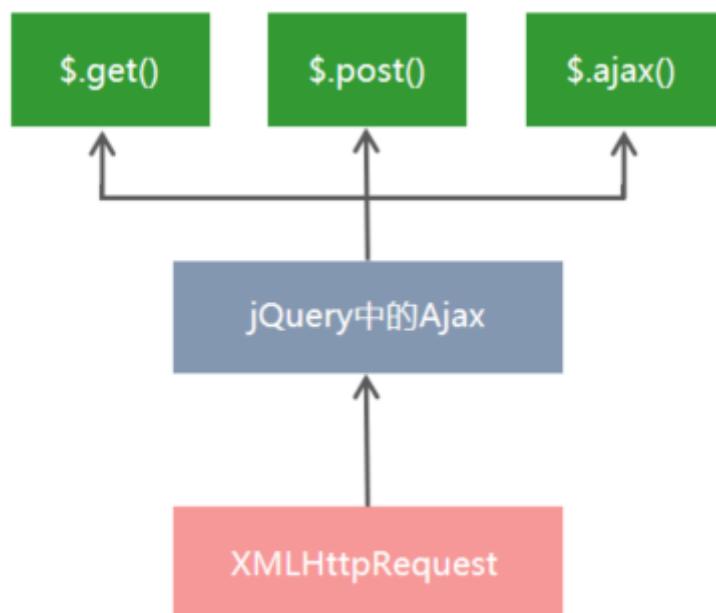
```

Ajax加强

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。



1.2 使用xhr发起无参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. 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
    //这一行是一个固定写法...这个200是浏览器的响应码，跟404那个一样
    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)
    }
}

```

1.3 了解xhr对象的readyState属性

XMLHttpRequest 对的 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。

1.4 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。

1.5 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

```

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() {} })
```

1.6 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(现在是可以出现中文字符了, 但是还是建议将中文进行编码)。

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
```

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
```

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

1.7 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

注意第三步，Post请求要比Get请求多这一步

```
// 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)
  }
}
```

The screenshot shows the Network tab in Chrome DevTools. A POST request to 'http://www.liulongbin.top:3006/api/addbook' is selected. The Headers panel shows:

- Request URL: http://www.liulongbin.top:3006/api/addbook
- Request Method: POST
- Status Code: 502 Bad Gateway
- Referrer Policy: strict-origin-when-cross-origin

The Payload panel shows the form data:

- bookname: 水浒传
- author: 施耐庵
- publisher: 天津图书出版社

2. 数据交换格式

2.1 什么是数据交换

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。



2.2 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

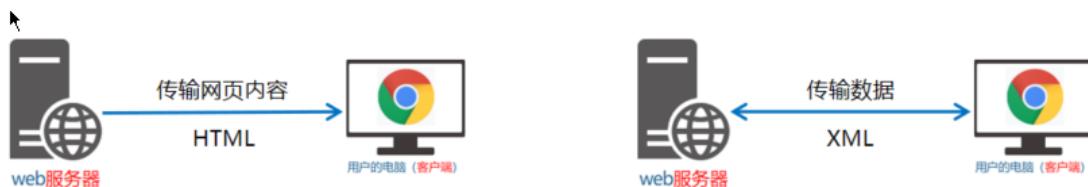
```
<note>
  <to>Tom</to>
  <from>Zhang</from>
  <heading>通知</heading>
  <body>晚上开会</body>
</note>
```

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



3. XML的缺点

XML 格式臃肿，和数据无关的代码多（标签比数据还多），体积大，传输效率低

在 Javascript 中解析 XML 比较麻烦

```
<note>
  <to>瑶瑶</to>
  <from>昊昊</from>
  <heading>通知</heading>
  <body>晚上开会</body>
</note>
```

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. 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。

1. 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 必须是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。

```
//这个是不对的
{
  name: "zs",
  'age': 20,
  "gender": '男',
  "address": undefined,
  "hobby": ["吃饭", "睡觉", '打豆豆'],
  say: function() {}
}
```

```
//修改后
{
  "name": "zs",
  "age": 20,
  "gender": "男",
  "address": null,
  "hobby": ["吃饭", "睡觉", "打豆豆"]
}
```

2. 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 ] ]
```

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

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"}'
```

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"}'
```

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。

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>
```

3.2 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

3.3 实现

测试页面：

```
<!DOCTYPE html>
<html lang="en">

    <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>Document</title>
        <script src="./js/itheima.js"></script>
    </head>

    <body>
        <script>
            // itheima({
            //     method: 'GET',
            //     url: 'http://www.liulongbin.top:3006/api/getbooks',
            //     data: {
            //         id: 1
            //     },
            //     success: function (res) {
            //         console.log(res)
            //     }
            // })
        </script>
    </body>
</html>
```

```

itheima({
  method: 'post',
  url: 'http://www.liulongbin.top:3006/api/addbook',
  data: {
    bookname: '水浒传',
    author: '施耐庵',
    publisher: '北京图书出版社'
  },
  success: function (res) {
    console.log(res)
  }
})
</script>
</body>

</html>

```

itheima.js:

```

function resolveData(data) {
  var arr = []
  for (var k in data) {
    var str = k + '=' + data[k]
    arr.push(str)
  }

  return arr.join('&')
}

// var res = resolveData({ name: 'zs', age: 20 })
// console.log(res)

function itheima(options) {
  var xhr = new XMLHttpRequest()

  // 把外界传递过来的参数对象，转换为 查询字符串
  var qs = resolveData(options.data)

  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)
  }

  xhr.onreadystatechange = function () {
    if (xhr.readyState === 4 && xhr.status === 200) {
      var result = JSON.parse(xhr.responseText)
    }
  }
}

```

```
        options.success(result)
    }
}
}
```

4. XMLHttpRequest Level2的新特性

4.1 认识XMLHttpRequest Level2

1. 旧版XMLHttpRequest的缺点

只支持文本数据的传输，无法用来读取和上传文件

传送和接收数据时，没有进度信息，只能提示有没有完成

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

4.2 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！')
}
```

4.3 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)
//下面可以绑定响应成功的事件，这里就不写了

```

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) {
    //禁用自动提交跳转
    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)
    //发送ajax请求
    var xhr = new XMLHttpRequest()
    xhr.open('POST', 'http://www.liulongbin.top:3006/api/formdata')
    xhr.send(fd)
    //还是监听响应成功的回调函数，这里就不写了
    xhr.onreadystatechange = function() {}
})

```

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

```

<!DOCTYPE html>
<html lang="en">

<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>Document</title>
</head>

```

```

<body>
    <!-- 1. 文件选择框 -->
    <input type="file" id="file1" />
    <!-- 2. 上传文件的按钮 -->
    <button id="btnUpload">上传文件</button>
    <br />
    <!-- 3. img 标签, 来显示上传成功以后的图片 -->
    <img src="" alt="" id="img" width="800" />

    <script>
        // 1. 获取到文件上传按钮
        var btnUpload = document.querySelector('#btnUpload')
        // 2. 为按钮绑定单击事件处理函数
        btnUpload.addEventListener('click', function () {
            // 3. 获取到用户选择的文件列表
            var files = document.querySelector('#file1').files
            if (files.length <= 0) {
                return alert('请选择要上传的文件!')
            }
            var fd = new FormData()
            // 将用户选择的文件, 添加到 FormData 中
            fd.append('avatar', files[0])

            var xhr = new XMLHttpRequest()
            xhr.open('POST', 'http://www.liulongbin.top:3006/api/upload/avatar')
            xhr.send(fd)

            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    var data = JSON.parse(xhr.responseText)
                    if (data.status === 200) {
                        // 上传成功
                        document.querySelector('#img').src =
                            'http://www.liulongbin.top:3006' + data.url
                    } else {
                        // 上传失败
                        console.log('图片上传失败! ' + data.message)
                    }
                }
            }
        })
    </script>
</body>

</html>

```

4.5 显示文件上传进度

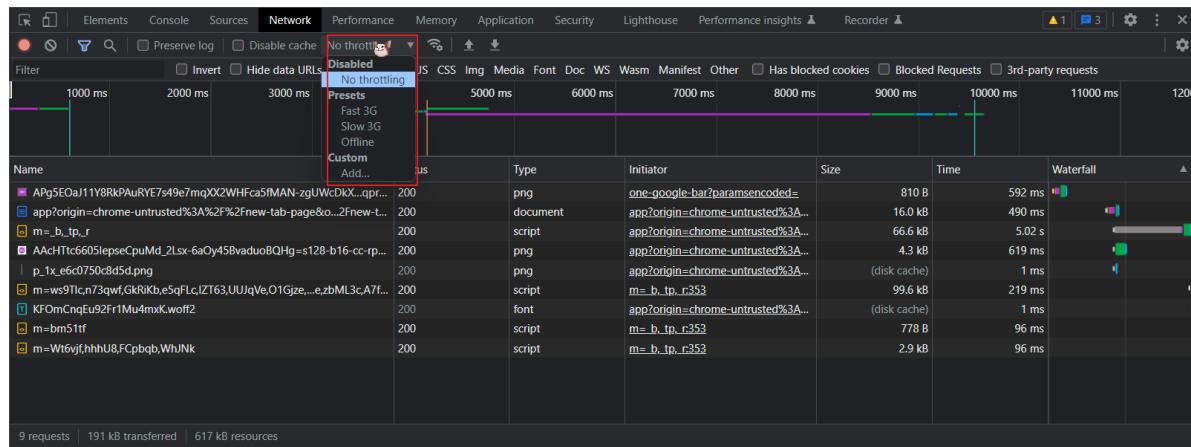
新版本的 XMLHttpRequest 对象中, 可以通过监听 `xhr.upload.onprogress` 事件, 来获取到文件的上传进度。语法格式如下:

```

// 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)
    }
}

```

修改上传的网速



1. 导入需要的库

```

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

```

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>

```

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. 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')
}
```

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>
```

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！')
    }
})
```

3. 向FormData中追加文件

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

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)
  }
})
```

5.2 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。

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()
})
```

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。

6.2 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(result)
})
```

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)
})
```

6.4 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, 请求体中的数据 */ },
    params: { /* GET参数, url中的查询参数 */ }
}) .then(callback)
```

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)
})
```

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)
})
```

6.5 使用await简化

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>

  <button id="btnPost">发起POST请求</button>
  <button id="btnGet">发起GET请求</button>

  <script src="./lib/axios.js"></script>
  <script>
    document.querySelector('#btnPost').addEventListener('click', async function () {
      // 如果调用某个方法的返回值是 Promise 实例，则前面可以添加 await!
      // await 只能用在被 async “修饰”的方法中
      const res = await axios({
        method: 'POST',
        url: 'http://www.liulongbin.top:3006/api/post',
      })
      console.log(res)
    })
  </script>
</body>
```

```

        data: {
            name: 'zs',
            age: 20
        }
    })

    //这个res就是我们之前得到的.then(res)里面的对象，这两个是一样的，只是不一种写法而已
    console.log(res)
}

</script>
</body>

</html>

```

还可以进行结构赋值

```

<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>

<body>

<button id="btnPost">发起POST请求</button>
<script src="./lib/axios.js"></script>
<script>
    document.querySelector('#btnPost').addEventListener('click', async function
() {
    //进行结构赋值
    const { data } = await axios({
        method: 'POST',
        url: 'http://www.liulongbin.top:3006/api/post',
        data: {
            name: 'zs',
            age: 20
        }
    })

    console.log(data)
})
//结构赋值并改
document.querySelector('#btnGet').addEventListener('click', async function
() {
    // 解构赋值的时候，使用 : 进行重命名
    // 1. 调用 axios 之后，使用 async/await 进行简化
    // 2. 使用解构赋值，从 axios 封装的大对象中，把 data 属性解构出来
    // 3. 把解构出来的 data 属性，使用 冒号 进行重命名，一般都重命名为 { data: res }
    const { data: res } = await axios({
        method: 'GET',

```

```
        url: 'http://www.liulongbin.top:3006/api/getbooks'
    })

    console.log(res.data)
})
</script>
</body>

</html>
```

跨域与JSONP

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。

下面的接口正常来说会请求失败，因为他们两个是非同源的

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。

1. 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。
2. 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。

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。

2.2 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。

2.3 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>
```

2.4 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。

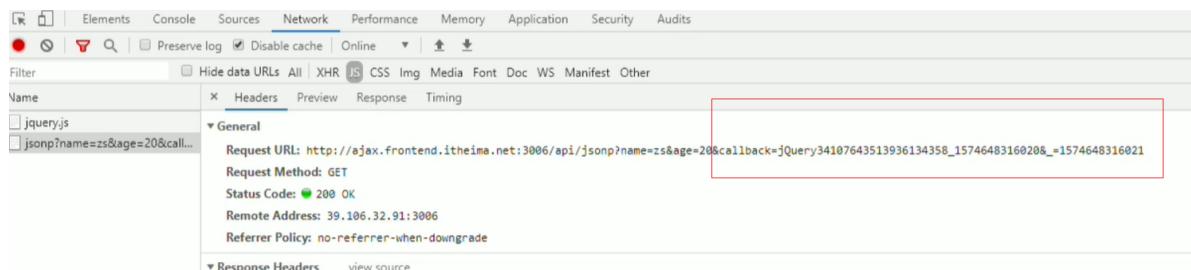
注意：JSONP 和 Ajax 之间没有任何关系，不能把 JSONP 请求数据的方式叫做 Ajax，因为 JSONP 没有用到 XMLHttpRequest 这个对象。

2.5 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({
    //这个跟上面的区别是参数，参数里面没有携带回调方法名称
    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',
    //这里是回调函数
    success: function(res) {
        console.log(res)
    }
})
```

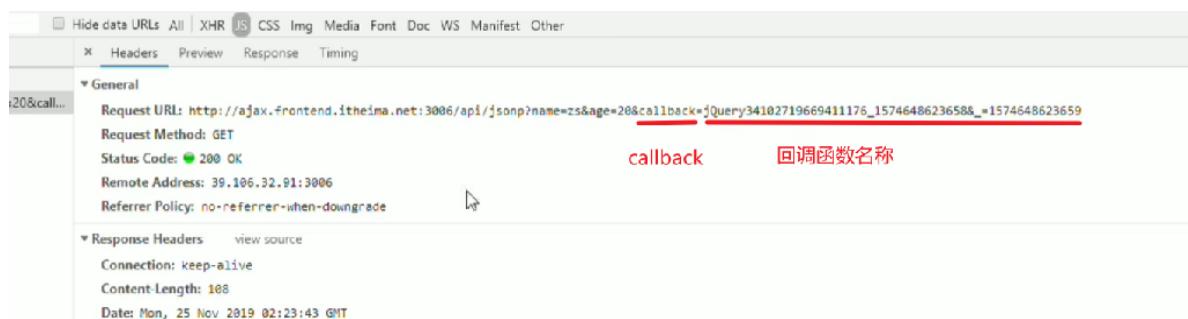
默认情况下，使用 jQuery 发起（注意这里是发起，发起的时候会自动拼接） JSONP 请求，会自动携带一个 `callback=jQueryxxx` 的参数，`jQueryxxx` 是随机生成的一个回调函数名称。



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)
    }
})
```



2.7 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；

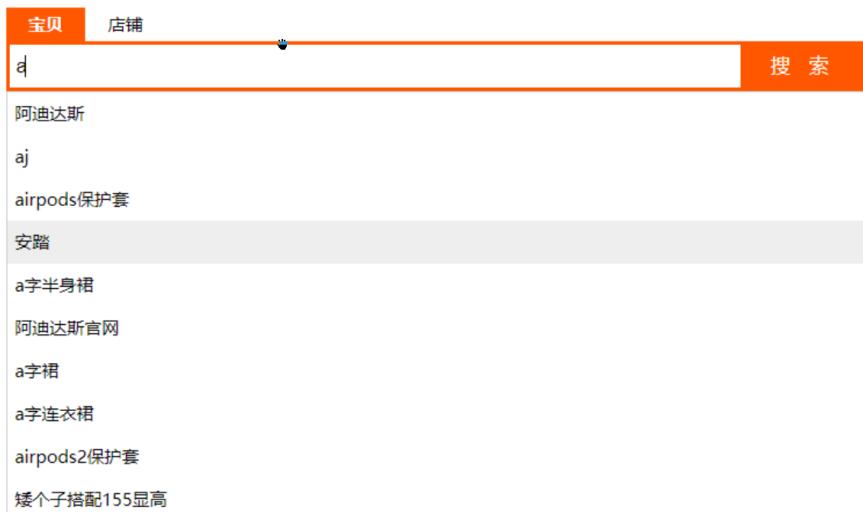
在 JSONP 请求成功以后，动态从 `<header>` 中移除刚才 append 进去的 `<script>` 标签；

这个好神奇，学会了，可以实现这种效果比如说点击按钮动态添加删除标签，

3. 案例 - 淘宝搜索

3.1 要实现的UI效果

淘宝网



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
})
```

3.3 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 -->
<script type="text/html" id="tpl-suggestList">
    {{each result}}
        <!--value是一个对象，他里面的第一个值就是我们要的数据-->
        <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
    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

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

3.5 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。

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. 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)
})
```

3.6 缓存搜索的建议列表

1. 定义全局缓存对象

```
// 缓存对象
var cacheObj = {}
```

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. 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。

就像射子弹, 需要一定时间的缓冲, 就算你按了多次只能触发一次

4.2 节流的应用场景

应用的目的就是降低事件的触发频率, 来达到优化性能的目的

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;

4.3 节流案例 - 鼠标跟随效果

1. 渲染UI结构并美化样式

```
<!-- UI 结构 -->

```

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')
    })
})

```

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. 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)
    })
})

```

4.4 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！

HTTP协议加强

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

现实生活中的通信

案例：张三要把自己考上郑州轻工业大学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；

通信的内容是考上郑州轻工业大学；

通信的方式是写信；

互联网中的通信

案例：服务器把郑州轻工业大学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；

通信的内容是郑州轻工业大学的简介；

通信的方式是响应；

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. 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。

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。

1.3 HTTP

1. 什么是HTTP协议

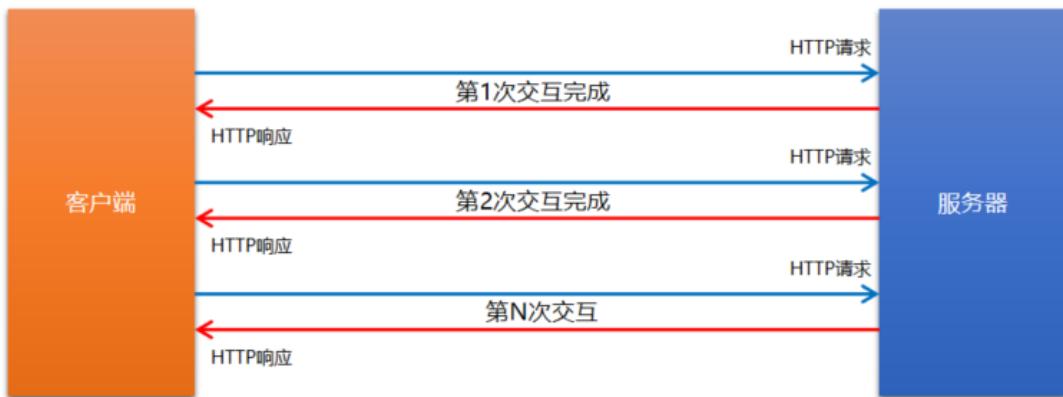
HTTP 协议即超文本传送协议 (HyperText Transfer Protocol)，它规定了客户端与服务器之间进行网页内容传输时，所必须遵守的传输格式。

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

2. HTTP协议的交互模型

HTTP 协议采用了 请求/响应 的交互模型。



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。

2.2 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。



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。



Headers Preview Response Timing

▼ Request Headers view parsed

POST /api/post HTTP/1.1

Host: ajax.frontend.itheima.net:3006

Proxy-Connection: keep-alive

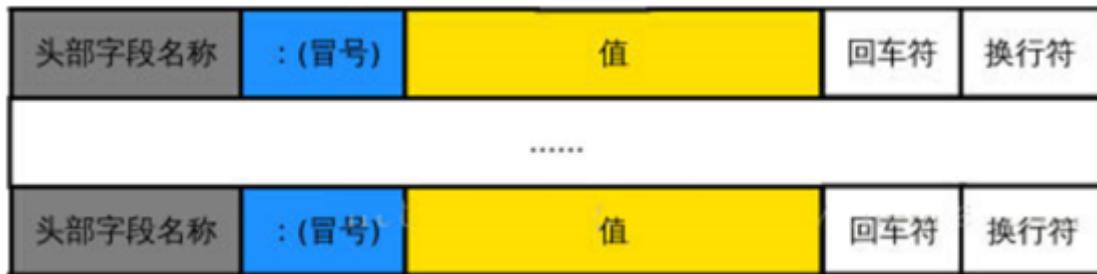
Content-Length: 14

Accept: */*

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。



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

```

▼ 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

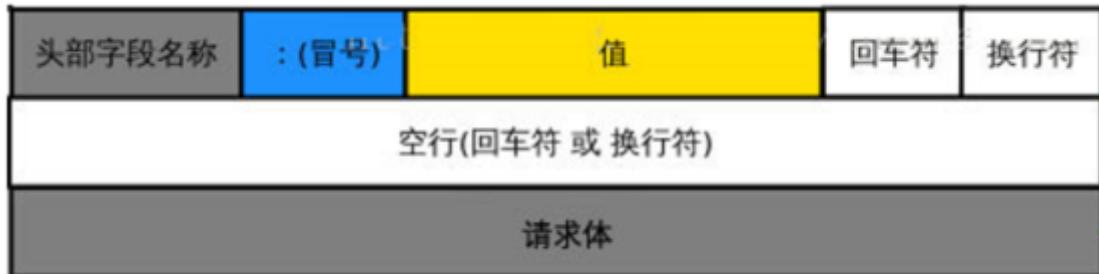
```

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>

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。



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。



这里的空行被省略了

请求体的最原始格式

注意：只有 POST 请求才有请求体，GET 请求没有请求体！

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。

3.2 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：



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；

协议版本	空格	状态码	空格	状态码描述	回车符	换行符
------	----	-----	----	-------	-----	-----

▼ 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/y6yOFj7H40EVN1DWB1MG+Pq0"
Date: Wed, 27 Nov 2019 01:48:57 GMT
Connection: keep-alive
```

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

常见的响应头字段

▼ Response Headers [view parsed](#)

HTTP/1.1 200 OK

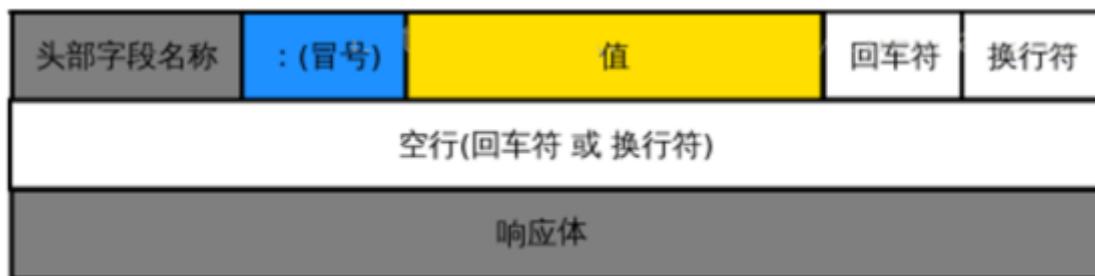
```
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 68
ETag: W/"44-nT/y6y0Fj7H40EVWlDWB1MG+Pq0"
Date: Wed, 27 Nov 2019 02:13:24 GMT
Connection: keep-alive
```

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. 空行

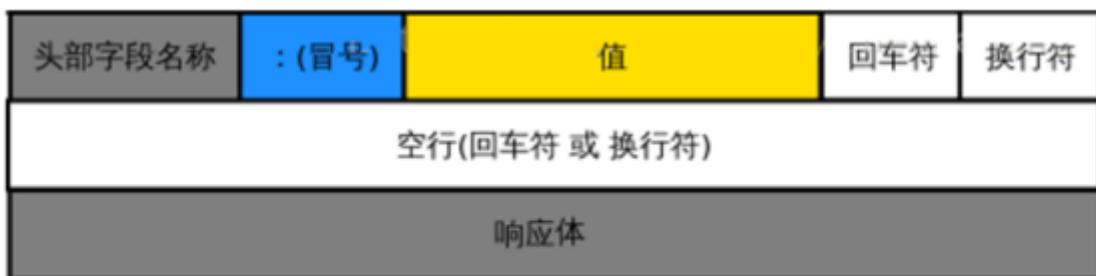
在最后一个响应头部字段结束之后，会紧跟一个空行，用来通知客户端响应头部至此结束。

响应消息中的空行，用来分隔响应头部与响应体。



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。



Headers Preview Response Timing

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

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。

4.2 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。

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
```

5.2 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>

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

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。

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。

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。

Git

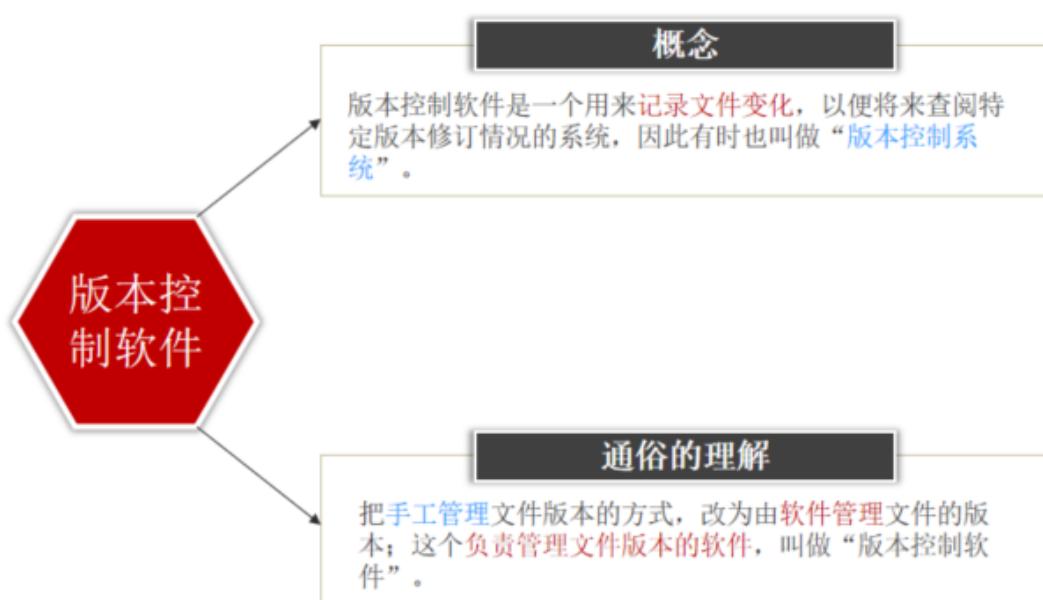
起步 - 关于版本控制

1. 文件的版本

这个同样适用于文档，比如语雀就有自己的版本库，可以恢复之前的版本的文档



2. 版本控制软件



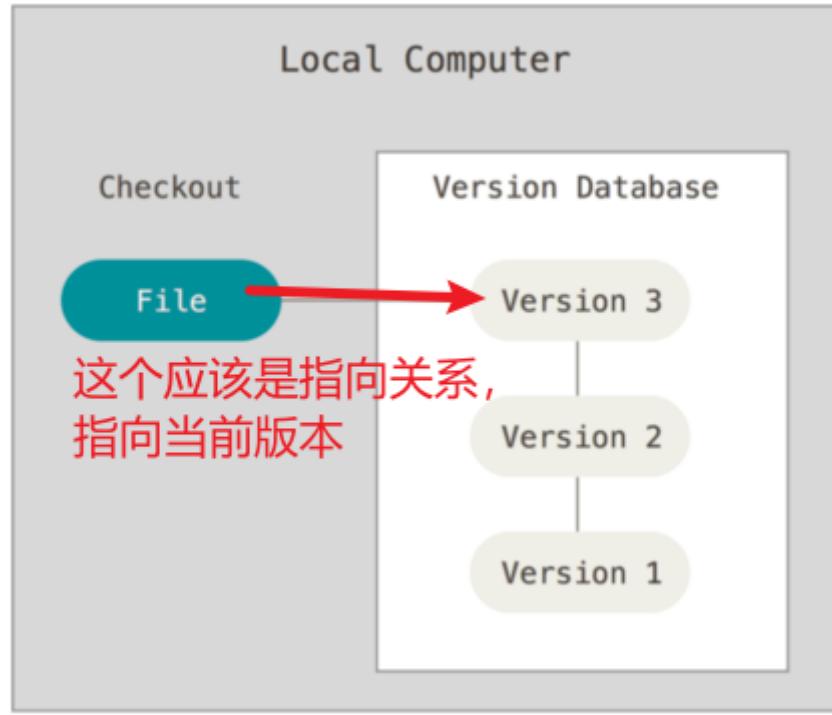
3. 使用版本控制软件的好处

操作简便	只需识记几组简单的终端命令，即可快速上手常见的版本控制软件
易于对比	基于版本控制软件提供的功能，能够方便地比较文件的变化细节，从而查找出导致问题的原因
易于回溯	可以将选定的文件回溯到之前的状态，甚至将整个项目都退回到过去某个时间点的状态
不易丢失	在版本控制软件中，被用户误删除的文件，可以轻松的恢复回来
协作方便	基于版本控制软件提供的分支功能，可以轻松实现多人协作开发时的代码合并操作

4. 版本控制系统的分类



4.1 本地版本控制系统



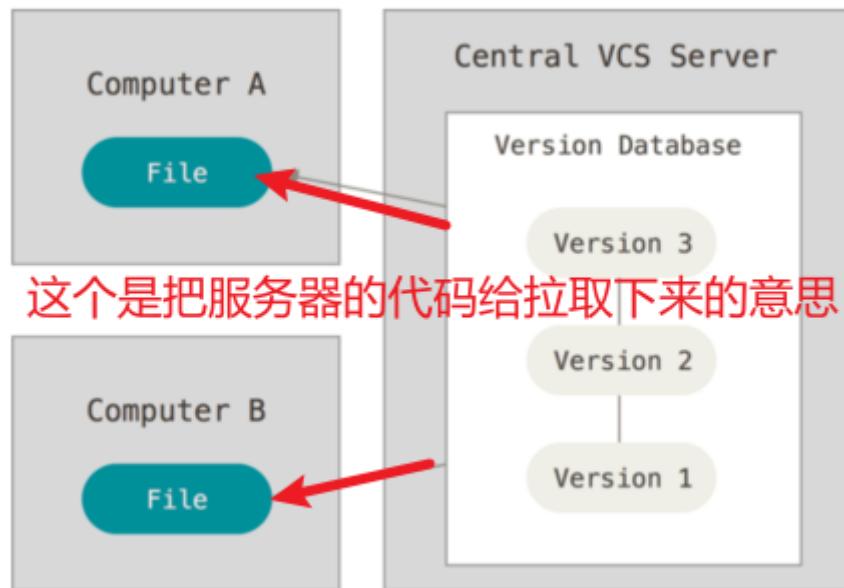
特点：

使用软件来记录文件的不同版本，提高了工作效率，降低了手动维护版本的出错率

缺点：

- 单机运行，不支持多人协作开发
- 版本数据库故障后，所有历史更新记录会丢失

4.2 集中化的版本控制系统



特点：基于服务器、客户端的运行模式

- 服务器保存文件的所有更新记录
- 客户端只保留最新的文件版本

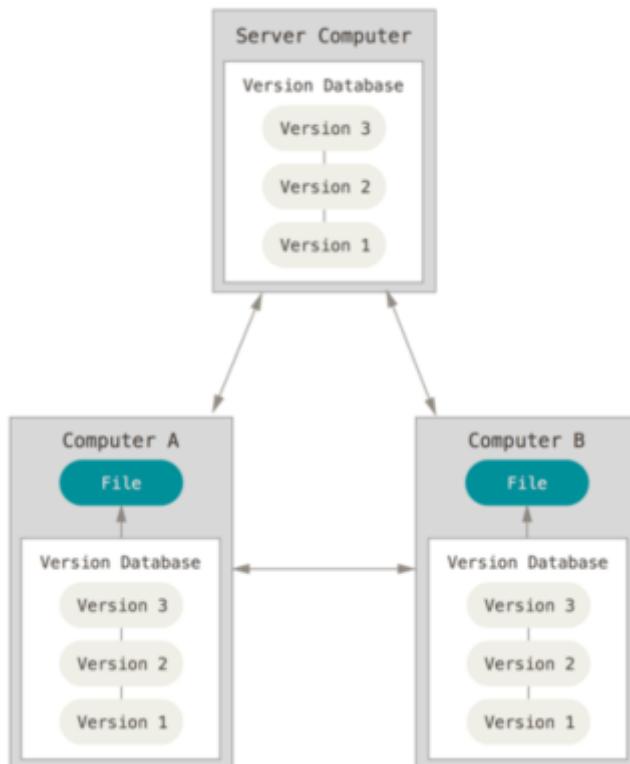
优点：联网运行，支持多人协作开发

缺点：

- 不支持离线提交版本更新
- 中心服务器崩溃后，所有人无法正常工作
- 版本数据库故障后，所有历史更新记录会丢失

典型代表：SVN

4.3 分布式版本控制系统



特点：基于服务器、客户端的运行模式

1. 服务器保存文件的所有更新版本
2. 客户端有服务器的完整(全部版本的)备份信息，并不是只保留文件的最新版本

优点：

1. 联网运行，支持多人协作开发
2. 客户端断网后支持离线本地提交版本更新
3. 服务器故障或损坏后，可使用任何一个客户端的备份进行恢复

典型代表：Git

起步 - Git 基础概念

1. 什么是 Git

Git 是一个开源的分布式版本控制系统，是目前世界上最先进、最流行的版本控制系统。可以快速高效地处理从很小到非常大的项目版本管理。

特点：项目越大越复杂，协同开发者越多，越能体现出 Git 的高性能和高可用性！

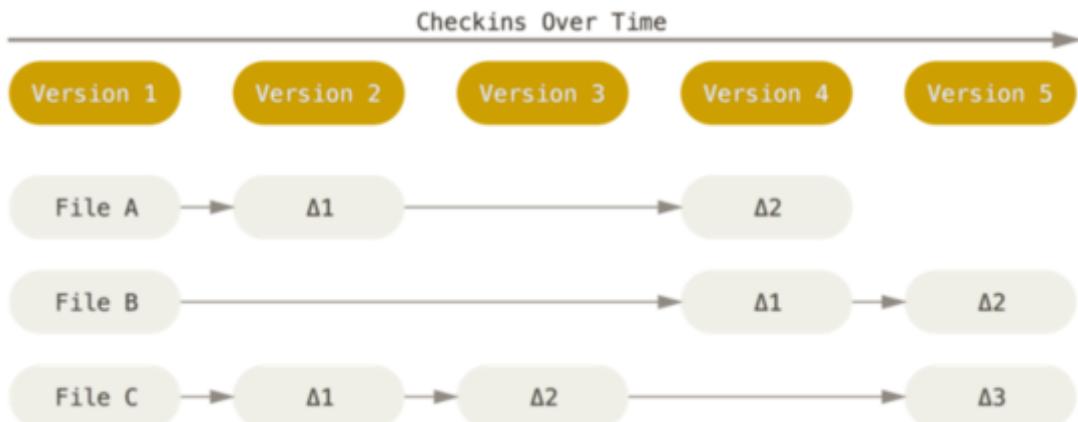
2. Git 的特性

Git 之所以快速和高效，主要依赖于它的如下两个特性：

1. 直接记录快照，而非差异比较
2. 近乎所有操作都是本地执行

2.1 SVN 的差异比较

传统的版本控制系统（例如 SVN）是基于差异的版本控制，它们存储的是一组基本文件和每个文件随时间逐步累积的差异。



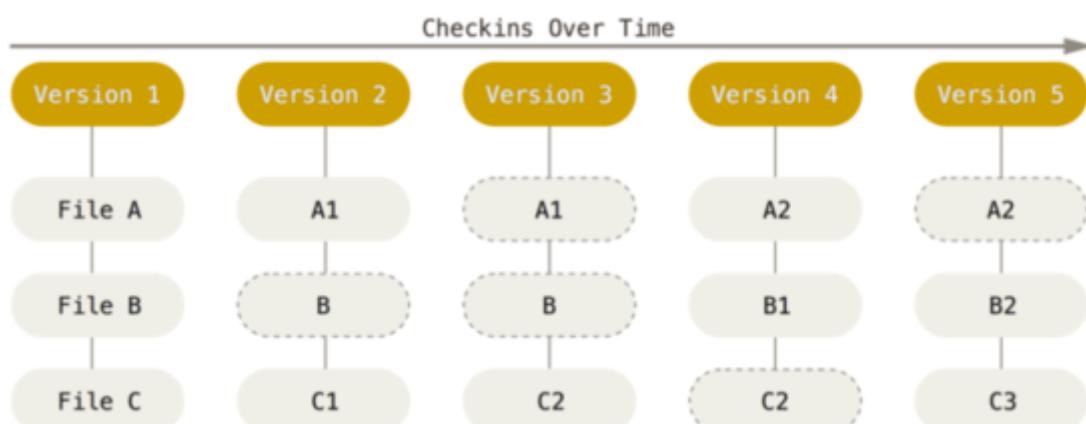
好处：节省磁盘空间

缺点：耗时、效率低

在每次切换版本的时候，都需要在基本文件的基础上，应用每个差异，从而生成目标版本对应的文件。

2.2 Git 的记录快照

Git 快照是在原有文件版本的基础上重新生成一份新的文件，类似于备份。为了效率，如果文件没有修改，Git 不再重新存储该文件（这样就节省了磁盘空间），而是只保留一个链接指向之前存储的文件。



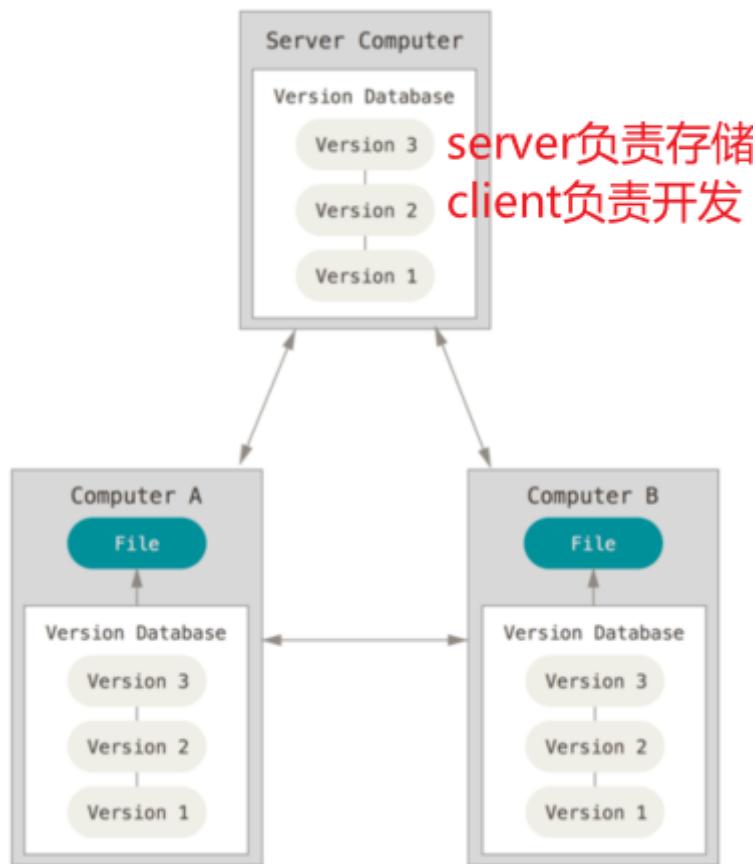
缺点：占用磁盘空间较大

优点：版本切换时非常快，因为每个版本都是完整的文件快照，切换版本时直接恢复目标版本的快照即可。

特点：用空间换时间

2.3 近乎所有操作都是本地执行

在 Git 中的绝大多数操作都只需要访问本地文件和资源，一般不需要来自网络上其它计算机的信息。



特性：

1. 断网后依旧可以在本地对项目进行版本管理
2. 联网后，把本地修改的记录同步到云端服务器即可

3. Git 中的三个区域

使用 Git 管理的项目，拥有三个区域，分别是工作区、暂存区、Git 仓库。



4. Git 中的三种状态



已修改	已暂存	已提交
表示修改了文件，但还没将修改的结果放到暂存区	表示对已修改文件的当前版本做了标记，使之包含在下次提交的列表中	表示文件已经安全地保存在本地的 Git 仓库中

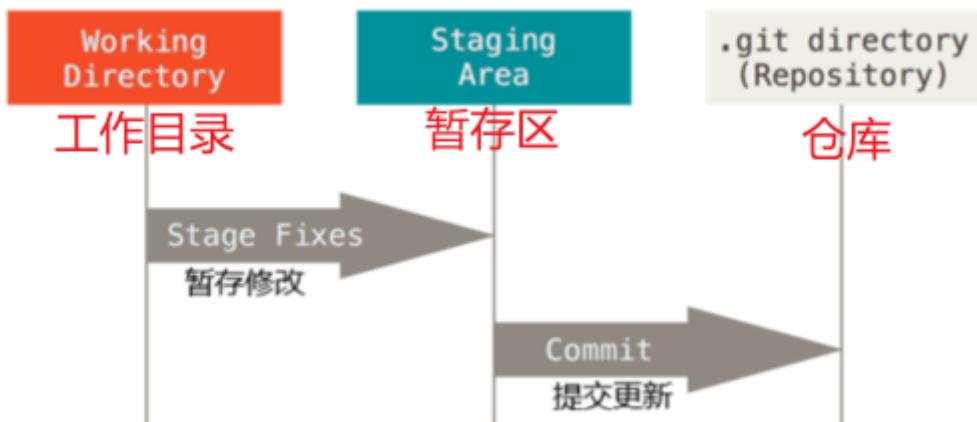
注意：

- 工作区的文件被修改了，但还没有放到暂存区，就是已修改状态。
- 如果文件已修改并放入暂存区，就属于已暂存状态。
- 如果 Git 仓库中保存着特定版本的文件，就属于已提交状态。

5. 基本的 Git 工作流程

基本的 Git 工作流程如下：

1. 在工作区中修改文件
2. 将你想要下次提交的更改进行暂存
3. 提交更新，找到暂存区的文件，将快照永久性存储到 Git 仓库



Git 基础 - 安装并配置 Git

1. 在 Windows 中下载并安装 Git

在开始使用 Git 管理项目的版本之前，需要将它安装到计算机上。可以使用浏览器访问如下的网址，根据自己的操作系统，选择下载对应的 Git 安装包：<https://git-scm.com/downloads>

2. 配置用户信息

安装完 Git 之后，要做的第一件事就是设置自己的用户名和邮件地址。因为通过 Git 对项目进行版本管理的时候，Git 需要使用这些基本信息，来记录是谁对项目进行了操作：

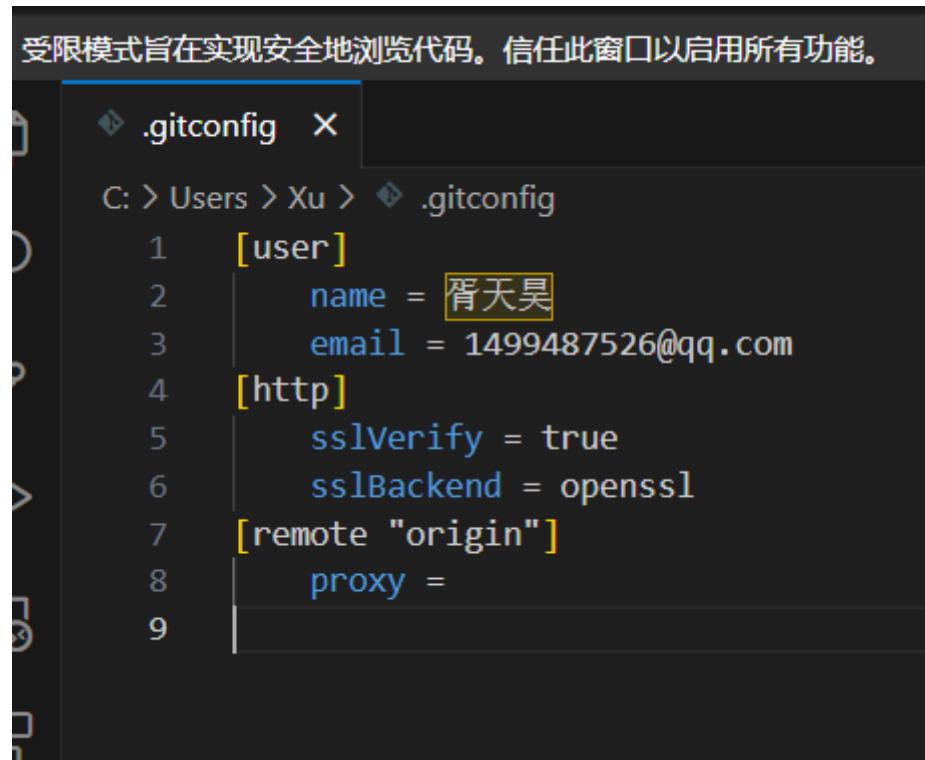
```
git config --global user.name "胥天昊"
git config --global user.email "xiaobaicai350@163.com"
```

注意：如果使用了 --global 选项，那么该命令只需要运行一次，即可永久生效。

3.Git 的全局配置文件

通过 git config --global user.name 和 git config --global user.email 配置的用户名和邮箱地址，会被写入到 C:/Users/用户名文件夹/.gitconfig 文件中。这个文件是 Git 的全局配置文件，配置一次即可永久生效。

可以使用记事本打开此文件，从而查看自己曾经对 Git 做了哪些全局性的配置。



受限模式旨在实现安全地浏览代码。信任此窗口以启用所有功能。

```
C: > Users > Xu > .gitconfig

1 [user]
2   name = 胥天昊
3   email = 1499487526@qq.com
4 [http]
5   sslVerify = true
6   sslBackend = openssl
7 [remote "origin"]
8   proxy =
```

4. 检查配置信息

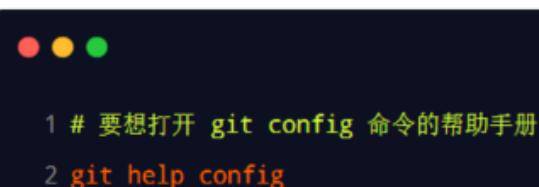
除了使用记事本查看全局的配置信息之外，还可以运行如下的终端命令，快速的查看 Git 的全局配置信息：

```
#查看所有的全局配置项
git config --list --global
```

```
#查看指定的全局配置项
#只想查看用户名
git config user.name
#只想查看用户的邮箱
git config user.email
```

5. 获取帮助信息

可以使用 git help <verb> 命令，无需联网即可在浏览器中打开帮助手册，例如：



```
1 # 要想打开 git config 命令的帮助手册
2 git help config
```

如果不想查看完整的手册，那么可以用 -h 选项获得更简明的“help”输出：

```
● ● ●  
1 # 想要获取 git config 命令的快速参考  
2 git config -h
```

Git 基础 - Git 的基本操作

1. 获取 Git 仓库的两种方式

1. 将尚未进行版本控制的本地目录转换为 Git 仓库
2. 从其它服务器克隆一个已存在的 Git 仓库（这个还没讲）

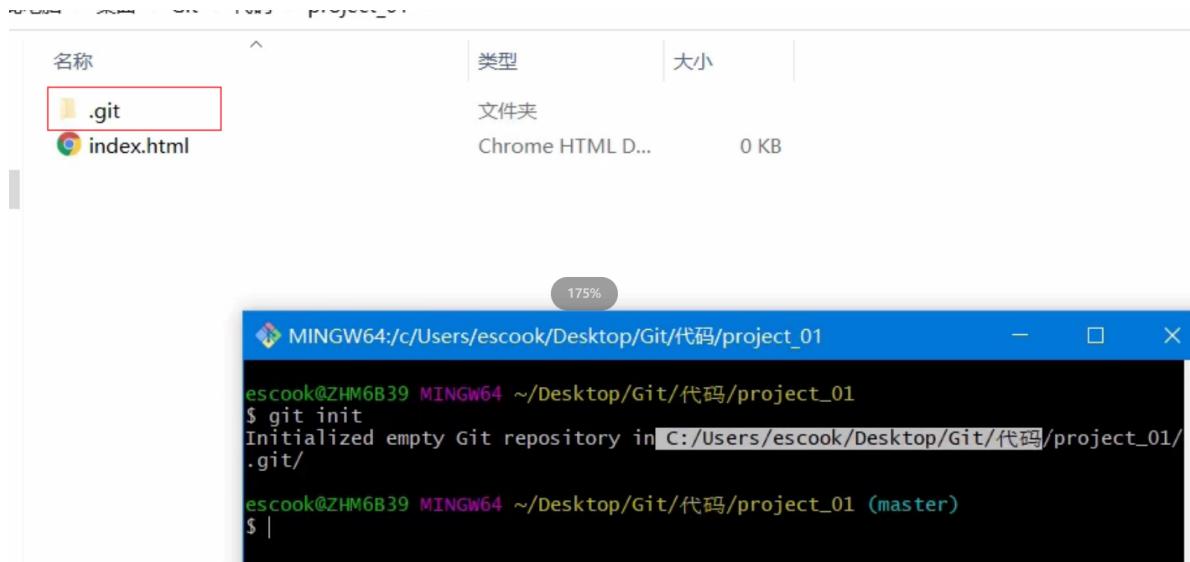
以上两种方式都能够在自己的电脑上得到一个可用的 Git 仓库

2. 在现有目录中初始化仓库

如果自己有一个尚未进行版本控制的项目目录，想要用 Git 来控制它，需要执行如下两个步骤：

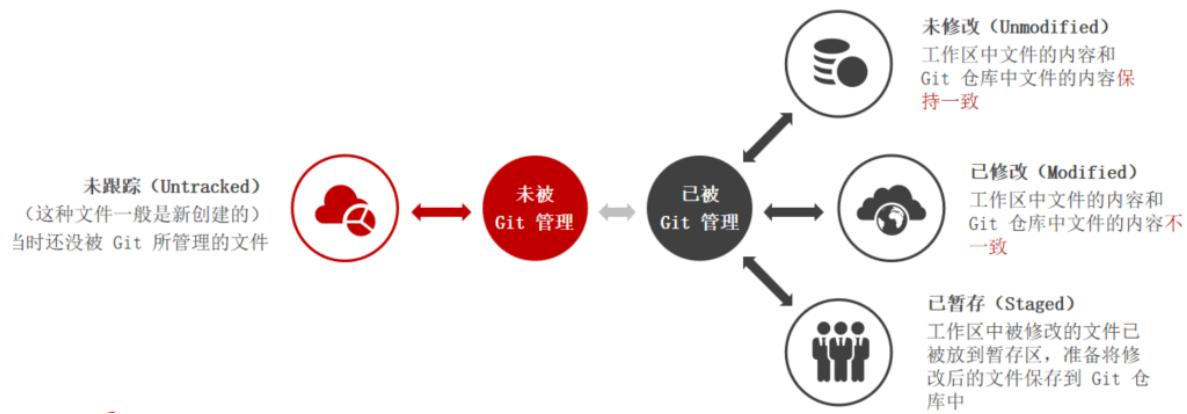
1. 在项目目录中，通过鼠标右键打开“Git Bash”
2. 执行 git init 命令将当前的目录转化为 Git 仓库

git init 命令会创建一个名为 .git 的隐藏目录，这个 .git 目录就是当前项目的 Git 仓库，里面包含了初始的必要文件，这些文件是 Git 仓库的必要组成部分。



3. 工作区中文件的 4 种状态

工作区中的每一个文件可能有 4 种状态，这四种状态共分为两大类，如图所示：



Git 操作的终极结果：让工作区中的文件都处于“未修改”的状态。

4. 检查文件的状态

可以使用 git status 命令查看文件处于什么状态，例如：

```
C:\Windows\System32\cmd.exe
E:\code>git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    index.html

nothing added to commit but untracked files present (use "git add" to track)
```

在状态报告中可以看到新建的 index.html 文件出现在 Untracked files (未跟踪的文件) 下面。

未跟踪的文件意味着 Git 在之前的快照 (提交) 中没有这些文件；Git 不会自动将之纳入跟踪范围，除非明确地告诉它“我需要使用 Git 跟踪管理该文件”。

5. 以精简的方式显示文件状态

使用 git status 输出的状态报告很详细，但有些繁琐。如果希望以精简的方式显示文件的状态，可以使用如下两条完全等价的命令，其中 -s 是 --short 的简写形式：

```
● ● ●
1 # 以精简的方式显示文件状态
2 git status -s      这两个命令是一样的哦
3 git status --short
```

未跟踪文件前面有红色的 ?? 标记（好像那个乱码...），例如：

```
选择C:\Windows\System32\cmd.exe  
E:\code>git status -s  
?? index.html
```

6. 跟踪新文件

使用命令 git add 开始跟踪一个文件。所以，要跟踪 index.html 文件，运行如下的命令即可：

```
git add index.html
```

此时再运行 git status 命令，会看到 index.html 文件在 Changes to be committed 这行的下面，说明已被跟踪，并处于暂存状态：

```
C:\Windows\System32\cmd.exe  
E:\code>git status  
On branch master  
  
No commits yet  
  
Changes to be committed:  
(use "git rm --cached <file>..." to unstage)  
  
    new file:   index.html
```

以精简的方式显示文件的状态：新添加到暂存区中的文件前面有绿色的 A 标记

```
C:\Windows\System32\cmd.exe  
E:\code>git status -s  
A  index.html
```

7. 提交更新

现在暂存区中有一个 index.html 文件等待被提交到 Git 仓库中进行保存。可以执行 git commit 命令进行提交，其中 -m 选项后面是本次的提交消息，用来对提交的内容做进一步的描述：

```
git commit -m "新建了index.html文件"
```

这个-m 'xxx信息' 是必须的参数，必须要一段提示信息

提交成功之后，会显示如下的信息：

```
C:\Windows\System32\cmd.exe

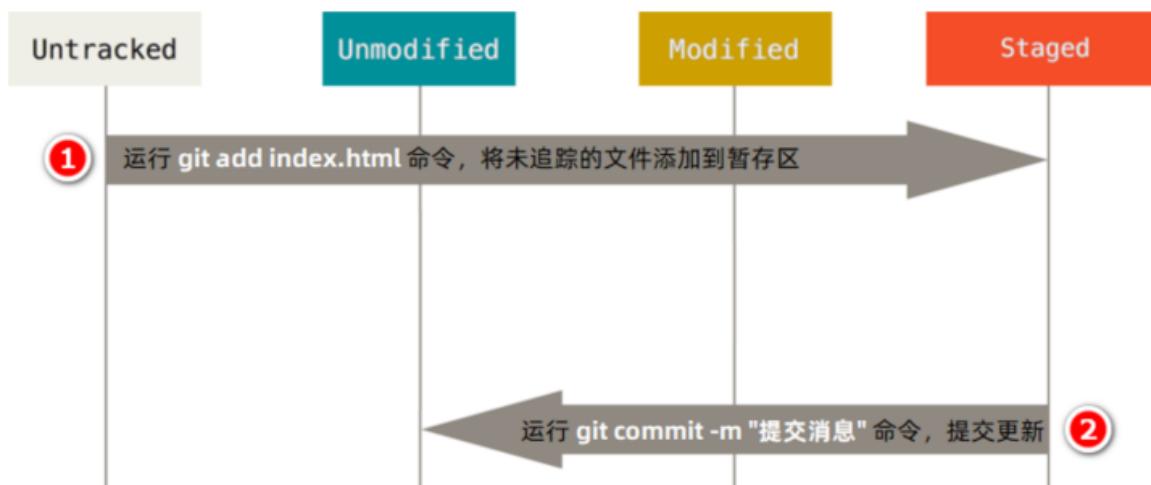
E:\code>git commit -m "新建了index.html文件"
[master (root-commit) 270b1f3] 新建了index.html文件
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
```

提交成功之后，再次检查文件的状态，得到提示如下：

```
C:\Windows\System32\cmd.exe

E:\code>git status
On branch master
nothing to commit, working tree clean
```

证明工作区中所有的文件都处于“未修改”的状态，没有任何文件需要被提交。



8. 对已提交的文件进行修改

目前，index.html 文件已经被 Git 跟踪，并且工作区和 Git 仓库中的 index.html 文件内容保持一致。当我们修改了(注意是修改过了之后)工作区中 index.html 的内容之后，再次运行 git status 和 git status -s 命令，会看到如下的内容：

```
C:\Windows\System32\cmd.exe

E:\code>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

E:\code>git status -s
 M index.html
```

文件 index.html 出现在 Changes not staged for commit 这行下面，说明已跟踪文件的内容发生了变化，但还没有放到暂存区。

注意：修改过的、没有放入暂存区的文件前面有红色的 M 标记。

9. 暂存已修改的文件

目前，工作区中的 index.html 文件已被修改，如果要暂存这次修改，需要再次运行 git add 命令，这个命令是个多功能的命令，主要有如下 3 个功效：

1. 可以用它开始跟踪新文件
2. 把已跟踪的、且已修改的文件放到暂存区(我们上面提到的修改过后的代码，就需要这个将他放到暂存区)
3. 把有冲突的文件标记为已解决状态

```
C:\Windows\System32\cmd.exe
E:\code>git add index.html 把已修改的文件放入暂存区
E:\code>git status 查看详细的文件状态报告
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   index.html

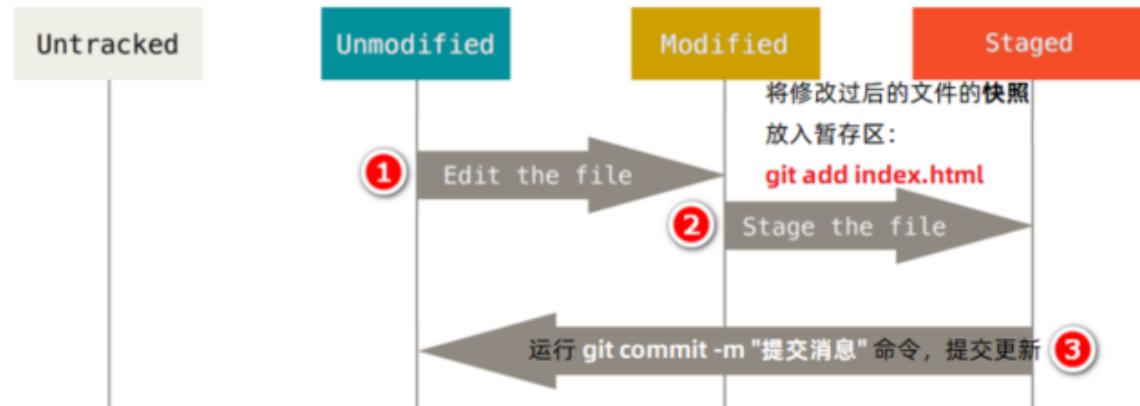
E:\code>git status -s 查看精简的文件状态报告,
M index.html 绿色的 M 表示文件已修改且已放入暂存区
```

10. 提交已暂存的文件

再次运行 git commit -m "提交消息" 命令，即可将暂存区中记录的 index.html 的快照，提交到 Git 仓库中进行保存：

```
C:\Windows\System32\cmd.exe
E:\code>git commit -m "初始化了index.html中的内容" 将暂存区中的文件提交到 Git 仓库
[master 554647a] 初始化了index.html中的内容
 1 file changed, 14 insertions(+)

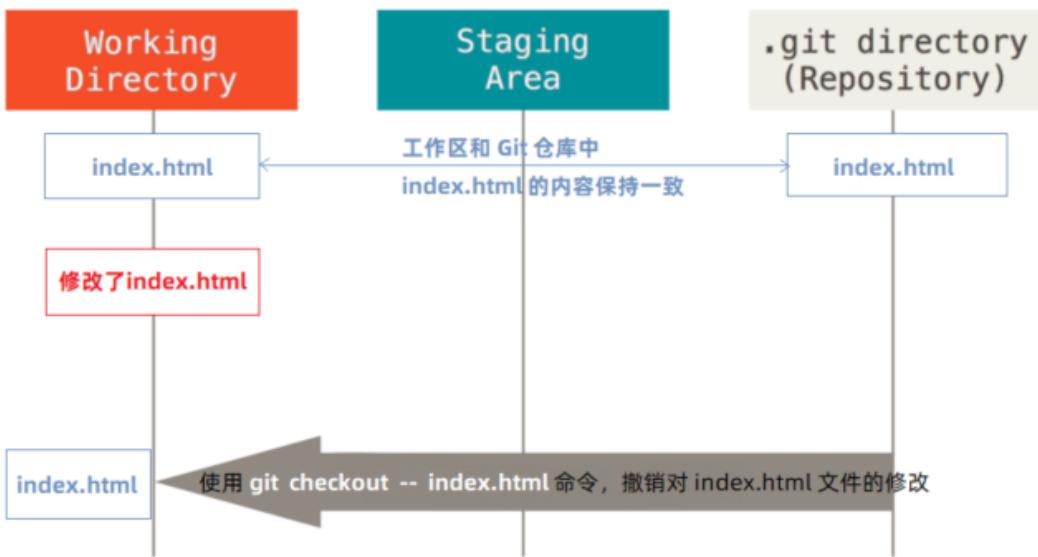
E:\code>git status 检查工作区中文件的状态
On branch master
nothing to commit, working tree clean
```



11. 撤销对文件的修改

撤销对文件的修改指的是：把对工作区中对应文件的修改，还原成 Git 仓库中所保存的版本。

操作的结果：所有的修改会丢失，且无法恢复！危险性比较高，请慎重操作！



撤销操作的本质：用 Git 仓库中保存的文件，覆盖工作区中指定的文件。

12. 向暂存区中一次性添加多个文件

如果需要被暂存的文件个数比较多，可以使用如下的命令，一次性将所有的新增(也就是还未被git管理的)和修改过的文件加入暂存区：

```
git add .
```

今后在项目开发中，会经常使用这个命令，将新增和修改过后的文件加入暂存区。

13. 取消暂存的文件

如果需要从暂存区中移除对应的文件，可以使用如下的命令：

```
git reset HEAD 要移除的文件名称
```

比如：

```
git reset HEAD index.html  
git reset HEAD .
```

14. 跳过使用暂存区域

Git 标准的工作流程是工作区 → 暂存区 → Git 仓库，但有时候这么做略显繁琐，此时可以跳过暂存区，直接将工作区中的修改（也就是把git add .和git commit -m "" 合为一个指令了）提交到 Git 仓库，这时候 Git 工作的流程简化为了工作区 → Git 仓库。

Git 提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤：

```
git commit -a -m "描述信息"
```

15. 移除文件

从 Git 仓库中移除文件的方式有两种：

1. 从 Git 仓库和工作区中同时移除对应的文件

2. 只从 Git 仓库中移除指定的文件，但保留工作区中对应的文件

```
#从Git仓库和工作区中同时移除index.js文件  
git rm -f index.js  
#只从Git仓库中移除index.css 但保存工作区中的index.css文件  
git rm --cached index.css
```

第一个指令会直接把本地目录下的文件给删除，千万要注意这两个指令的区别，要不然真的可能会欲哭无泪。。。

16. 忽略文件

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。在这种情况下，我们可以创建一个名为 `.gitignore` 的配置文件，列出要忽略的文件的匹配模式。

文件 `.gitignore` 的格式规范如下：

1. 以 `#` 开头的是注释
2. 以 `/` 结尾的是目录
3. 以 `/` 开头防止递归
4. 以 `!` 开头表示取反
5. 可以使用 glob 模式进行文件和文件夹的匹配 (glob 指简化了的正则表达式)

```
D: > train2023 > shixun > .gitignore
1  HELP.md
2  target/
3  !.mvn/wrapper/maven-wrapper.jar
4  !**/src/main/**/target/
5  !**/src/test/**/target/
6
7  ### STS ###
8  .apt_generated
9  .classpath
10 .factorypath
11 .project
12 .settings
13 .springBeans
14 .sts4-cache
15
16 ### IntelliJ IDEA ###
17 .idea
18 *.iws
19 *.iml
20 *.ipr
21
22 ### NetBeans ###
23 /nbproject/private/
24 /nbbuild/
25 /dist/
26 /nbdist/
27 /.nb-gradle/
28 build/
29 !**/src/main/**/build/
30 !**/src/test/**/build/
31
32 ### VS Code ###
33 .vscode/
```

17. glob 模式

所谓的 glob 模式是指简化了的正则表达式：

1. 星号 * 匹配零个或多个任意字符
2. [abc] 匹配任何一个列在方括号中的字符（此案例匹配一个 a 或匹配一个 b 或匹配一个 c）
3. 问号 ? 只匹配一个任意字符

4. 在方括号中使用短划线分隔两个字符，表示所有在这两个字符范围内的都可以匹配（比如 [0-9] 表示匹配所有 0 到 9 的数字）
5. 两个星号 ** 表示匹配任意中间目录（比如 a/**/z 可以匹配 a/z、a/b/z 或 a/b/c/z 等）

18. .gitignore 文件的例子

```
1 # 忽略所有的 .a 文件
2 *.a
3
4 # 但跟踪所有的 lib.a, 即便你在前面忽略了 .a 文件
5 !lib.a
6
7 # 只忽略当前目录下的 TODO 文件, 而不忽略 subdir/TODO
8 /TODO
9
10 # 忽略任何目录下名为 build 的文件夹
11 build/
12
13 # 忽略 doc/notes.txt, 但不忽略 doc/server/arch.txt
14 doc/*.txt
15
16 # 忽略 doc/ 目录及其所有子目录下的 .pdf 文件
17 doc/**/*.pdf
```

19. 查看提交历史

如果希望回顾项目的提交历史，可以使用 git log 这个简单且有效的命令。

```
1 # 按时间先后顺序列出所有的提交历史, 最近的提交排在最上面
2 git log
3
4 # 只展示最新的两条提交历史, 数字可以按需进行填写
5 git log -2
6
7 # 在一行上展示最近两条提交历史的信息
8 git log -2 --pretty=oneline
9
10 # 在一行上展示最近两条提交历史的信息, 并自定义输出的格式
11 # %h 提交的简写哈希值    %an作者名字    %ar作者修订日期, 按多久以前的方式显示    %s提交说明
12 git log -2 --pretty=format:"%h | %an | %ar | %s"
```

最后一个指令的效果：

```
escook@ZHM6B39 MINGW64 ~/Desktop/Git/代码/project_01 (master)
$ git log -2 --pretty=format:"%h | %an | %ar | %s"
28dd9c5 | itheima | 37 minutes ago | 创建了git忽略文件
805dd1c | itheima | 4 hours ago | 移除了index.css和index.js
```

20. 回退到指定的版本

```
1 # 在一行上展示所有的提交历史
2 git log --pretty=oneline
3
4 # 使用 git reset --hard 命令，根据指定的提交 ID 回退到指定版本
5 git reset --hard <CommitID>
6
7 # 在旧版本中使用 git reflog --pretty=oneline 命令，查看命令操作的历史
8 git reflog --pretty=oneline
9
10 # 再次根据最新的提交 ID，跳转到最新的版本
11 git reset --hard <CommitID>
```

第三条是回滚之后查看全部历史，回滚之后运行第一条命令只能查看当前版本之前的历史记录

21. 小结

初始化 Git 仓库的命令

```
git init
```

查看文件状态的命令

```
git status 或 git status -s
```

一次性将文件加入暂存区的命令

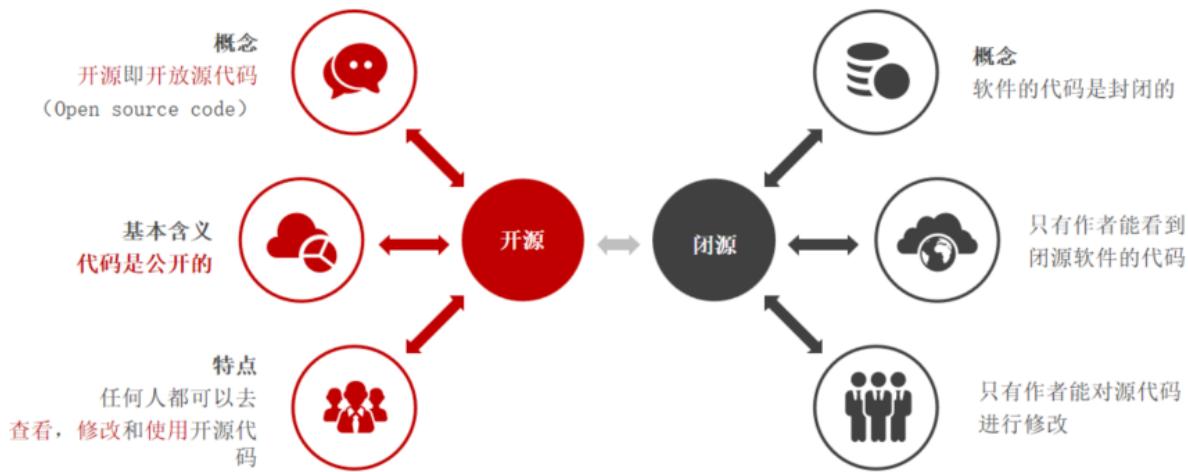
```
git add .
```

将暂存区的文件提交到 Git 仓库的命令

```
git commit -m "提交消息"
```

Github - 了解开源相关的概念

1. 什么是开源



通俗的理解：

开源是指不仅提供程序还提供程序的源代码

闭源是只提供程序，不提供源代码

2. 什么是开源许可协议

开源并不意味着完全没有限制，为了限制使用者的使用范围和保护作者的权利，每个开源项目都应该遵守开源许可协议（Open Source License）。

3. 常见的 5 种开源许可协议

1. BSD (Berkeley Software Distribution)
2. Apache Licence 2.0
3. GPL (GNU General Public License)
 1. 具有传染性的一种开源协议，不允许修改后和衍生的代码做为闭源的商业软件发布和销售
 2. 使用 GPL 的最著名的软件项目是：Linux
4. LGPL (GNU Lesser General Public License)
5. MIT (Massachusetts Institute of Technology, MIT)
 1. 是目前限制最少的协议，唯一的条件：在修改后的代码或者发行包中，必须包含原作者的许可信息
 2. 使用 MIT 的软件项目有：jquery、Node.js

关于更多开源许可协议的介绍，可以参考博客 <https://www.runoob.com/w3cnote/open-source-licenses.html>

4. 为什么要拥抱开源

开源的核心思想是“我为人人，人人为我”，人们越来越喜欢开源大致是出于以下 3 个原因：

1. 开源给使用者更多的控制权
2. 开源让学习变得容易
3. 开源才有真正的安全

开源是软件开发领域的大趋势，拥抱开源就像站在了巨人的肩膀上，不用自己重复造轮子，让开发越来越容易。

5. 开源项目托管平台

专门用于免费存放开源项目源代码的网站，叫做开源项目托管平台。目前世界上比较出名的开源项目托管平台主要有以下 3 个：

- Github（全球最牛的开源项目托管平台，没有之一）
- Gitlab（对代码私有性支持较好，因此企业用户较多）
- Gitee（又叫做码云，是国产的开源项目托管平台。访问速度快、纯中文界面、使用友好）

注意：以上 3 个开源项目托管平台，只能托管以 Git 管理的项目源代码，因此，它们的名字都以 Git 开头。

6. 什么是 Github

Github 是全球最大的开源项目托管平台。因为只支持 Git 作为唯一的版本控制工具，故名 GitHub。

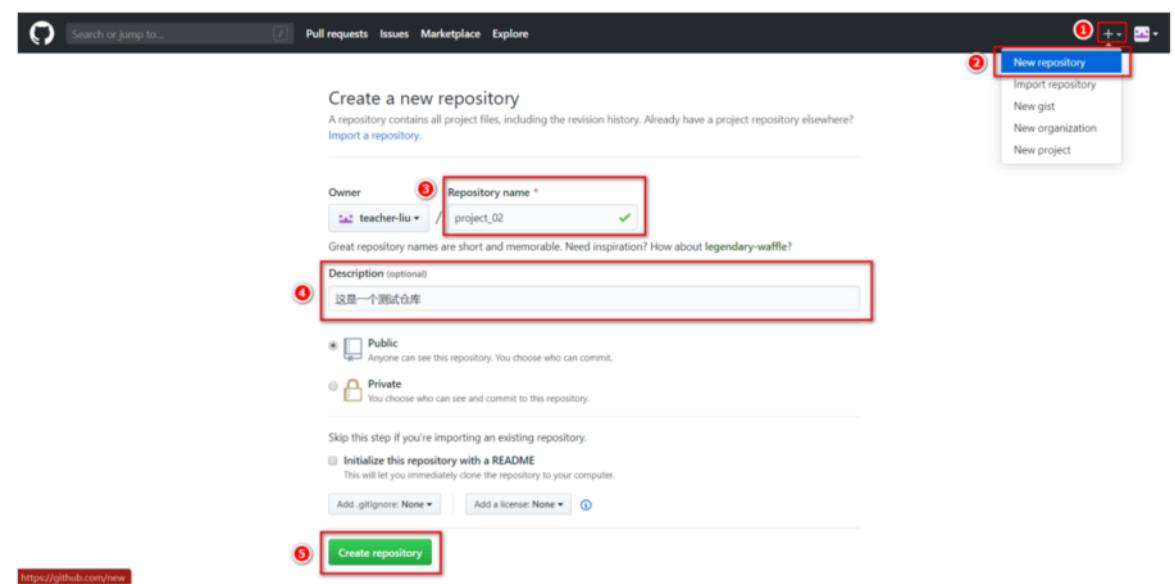
在 Github 中，你可以：

1. 关注自己喜欢的开源项目，为其点赞打 call
2. 为自己喜欢的开源项目做贡献（Pull Request）
3. 和开源项目的作者讨论 Bug 和提需求（Issues）
4. 把喜欢的项目复制一份作为自己的项目进行修改（Fork）
5. 创建属于自己的开源项目
6. etc...

So, Github ≠ Git

Github - 远程仓库的使用

1. 新建空白远程仓库



2. 新建空白远程仓库成功

The screenshot shows a GitHub repository page for 'teacher-liu / project_02'. At the top, there's a search bar and navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header, there's a 'Code' tab and several other tabs like 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. A 'Quick setup' section is displayed, with 'HTTPS' being the selected option. It provides instructions for creating a new repository via command line or pushing an existing one. It also includes a link to 'Set up in Desktop'.

3. 远程仓库的两种访问方式

Github 上的远程仓库，有两种访问方式，分别是 HTTPS 和 SSH。它们的区别是：

1. HTTPS：零配置；但是每次访问仓库时，需要重复输入 Github 的账号和密码才能访问成功
2. SSH：需要进行额外的配置；但是配置成功后，每次访问仓库时，不需重复输入 Github 的账号和密码

注意：在实际开发中，推荐使用 SSH 的方式访问远程仓库。

4. 基于 HTTPS 将本地仓库上传到 Github

这种方法需要账号和密码，但是目前的github好像是基于token的。不需要了

This screenshot shows the 'Quick setup' section for GitHub's HTTPS repository creation. It highlights the command-line steps for creating a new repository:

- 1. 使用终端命令创建 README.md 文档，并写入初始内容为 # project_02**
- 2. 初始化本地 Git 仓库，并将文件的修改提交到本地的 Git 仓库中**
- 3. 将本地仓库和远程仓库进行关联，并把远程仓库命名为 origin**
- 4. 将本地仓库中的内容推送到远程的 origin 仓库中**

Below this, another section for existing repositories is shown, with a similar set of steps highlighted.

5. SSH key

SSH key 的作用：实现本地仓库和 Github 之间免登录的加密数据传输。

SSH key 的好处：免登录身份认证、数据加密传输。

SSH key 由两部分组成，分别是：

1. id_rsa (私钥文件，存放于客户端的电脑中即可)

2. id_rsa.pub (公钥文件, 需要配置到 Github 中)

6. 生成 SSH key

打开 Git Bash

粘贴如下的命令, 并将 your_email@example.com 替换为注册 Github 账号时填写的邮箱:

- ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

连续敲击 3 次回车, 即可在 C:\Users\用户名文件夹.ssh 目录中生成 id_rsa 和 id_rsa.pub 两个文件

7. 配置 SSH key

使用记事本打开 id_rsa.pub 文件, 复制里面的文本内容

在浏览器中登录 Github, 点击头像 -> Settings -> SSH and GPG Keys -> New SSH key

将 id_rsa.pub 文件中的内容, 粘贴到 Key 对应的文本框中

在 Title 文本框中任意填写一个名称, 来标识这个 Key 从何而来

8. 检测 Github 的 SSH key 是否配置成功

打开 Git Bash, 输入如下的命令并回车执行:

```
1 ssh -T git@github.com
```

上述的命令执行成功后, 可能会看到如下的提示消息:

```
1 The authenticity of host 'github.com (IP ADDRESS)' can't be established.  
2 RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.  
3 Are you sure you want to continue connecting (yes/no)?
```

输入 yes 之后, 如果能看到类似于下面的提示消息, 证明 SSH key 已经配置成功了:

```
1 Hi username! You've successfully authenticated, but GitHub does not  
2 provide shell access.
```

9. 基于 SSH 将本地仓库上传到 Github

Quick setup — if you've **①**one this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com:teacher-liu/project_03.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# project_03" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin git@github.com:teacher-liu/project_03.git  
git push -u origin master
```

将本地现成的仓库推送到 Github

...or push an existing repository from the command line

② `git remote add origin git@github.com:teacher-liu/project_03.git` 1. 将本地仓库和远程仓库进行关联，并把远程仓库命名为 origin
`git push -u origin master` 2. 将本地仓库中的内容推送到远程的 origin 仓库中

10. 将远程仓库克隆到本地

打开 Git Bash，输入如下的命令并回车执行：

```
1 git clone 远程仓库的地址
```

Git 分支 - 本地分支操作

1. 分支的概念

分支就是科幻电影里面的平行宇宙，当你正在电脑前努力学习Git的时候，另一个你正在另一个平行宇宙里努力学习SVN。

如果两个平行宇宙互不干扰，那对现在的你也没啥影响。

不过，在某个时间点，两个平行宇宙合并了，结果，你既学会了Git又学会了SVN！

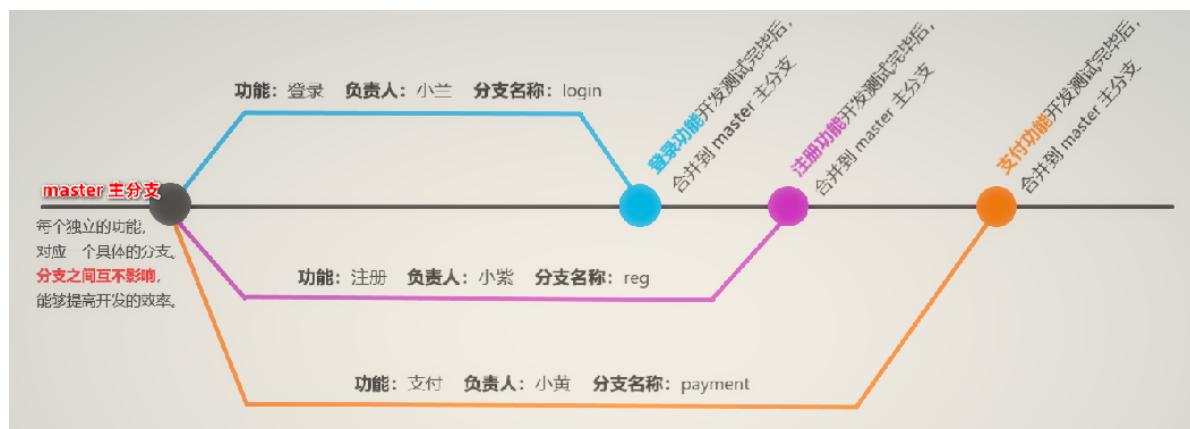
2. 分支在实际开发中的作用

在进行多人协作开发的时候，为了防止互相干扰，提高协同开发的体验，建议每个开发者都基于分支进行项目功能的开发，例如：



3. master 主分支

在初始化本地 Git 仓库的时候, Git 默认已经帮我们创建了一个名字叫做 master 的分支。通常我们把这个 master 分支叫做主分支。



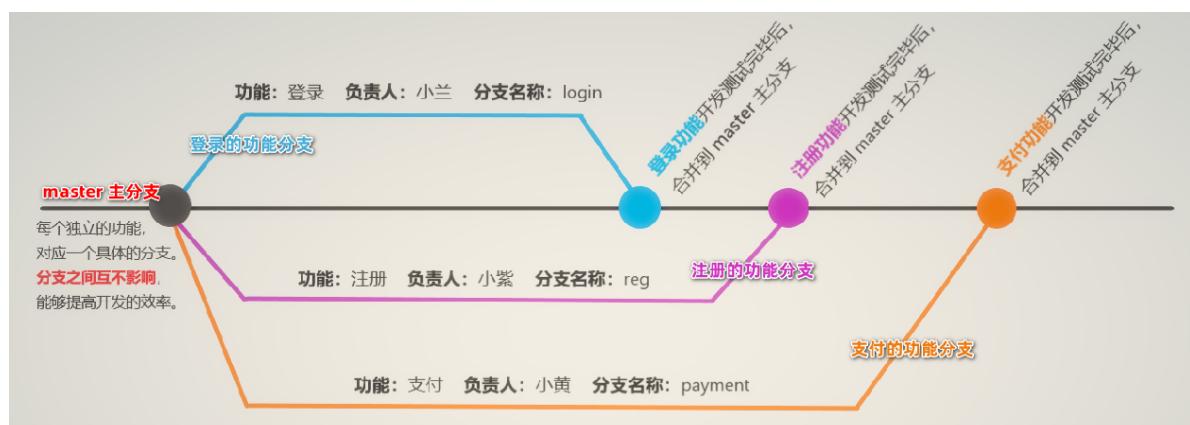
在实际工作中, master 主分支的作用是: 用来保存和记录整个项目已完成的功能代码。

因此, 不允许程序员直接在 master 分支上修改代码, 因为这样做的风险太高, 容易导致整个项目崩溃。

4. 功能分支

由于程序员不能直接在 master 分支上进行功能的开发, 所以就有了功能分支的概念。

功能分支指的是专门用来开发新功能的分支, 它是临时从 master 主分支上分叉出来的, 当新功能开发且测试完毕后, 最终需要合并到 master 主分支上, 如图所示:



5. 查看分支列表

使用如下的命令，可以查看当前 Git 仓库中所有的分支列表：

```
git branch
```

运行的结果如下所示：

```
C:\Windows\System32\cmd.exe  
E:\code2>git branch  
* master  
E:\code2>
```

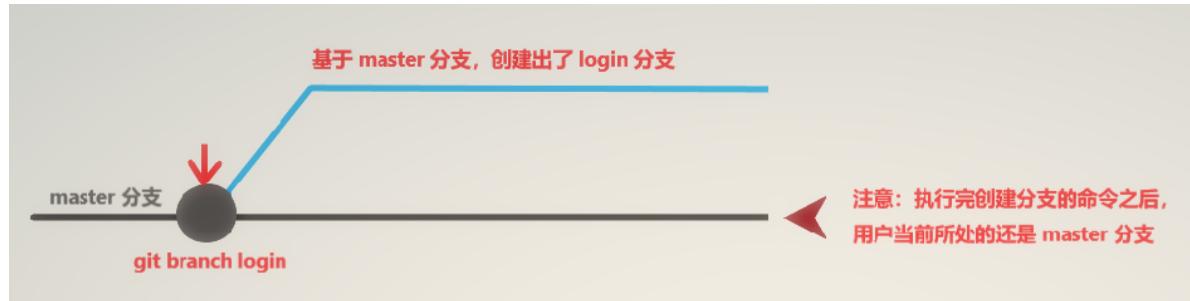
注意：分支名字前面的 * 号表示当前所处的分支。

6. 新建分支

使用如下的命令，可以基于当前所处分支，创建一个新的分支，此时，新分支中的代码和当前分支完全一样（执行完这行命令不会自动切换分支，还是处于当前分支）：

```
1 git branch 分支名称
```

图示如下：



7. 切换分支

使用如下的命令，可以切换到指定的分支上进行开发：

```
git checkout login
```

图示如下：



8. 分支的快速创建和切换

使用如下的命令，可以创建指定名称的新分支，并立即切换到新分支上：



图示如下：



注意：

"`git checkout -b 分支名称`" 是下面两条命令的简写形式：

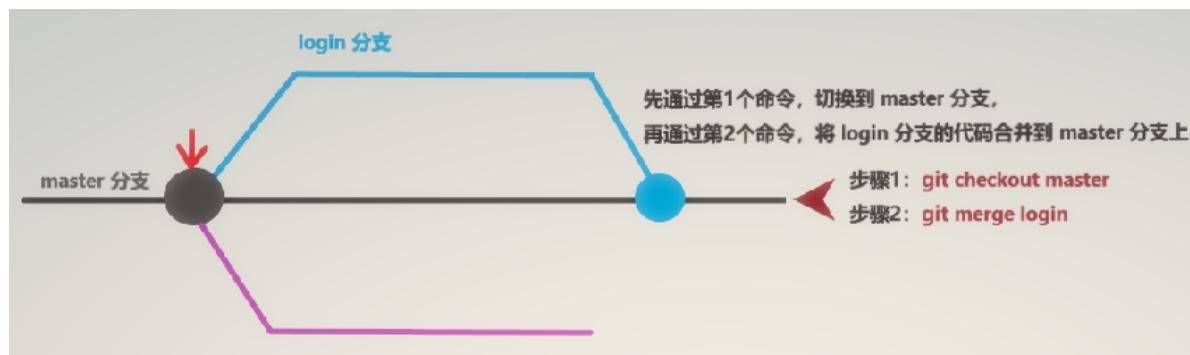
1. `git branch 分支名称`
2. `git checkout 分支名称`

9. 合并分支

功能分支的代码开发测试完毕之后，可以使用如下的命令，将完成后的代码合并到 `master` 主分支上：



图示如下：



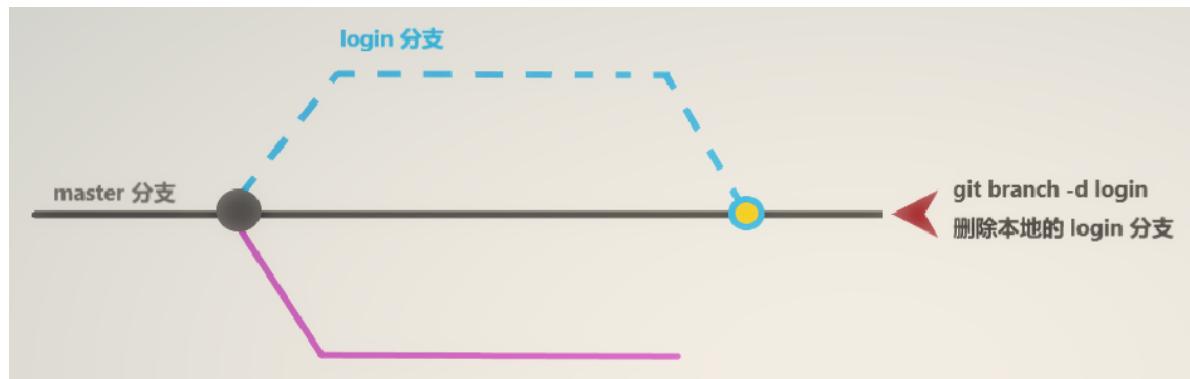
合并分支时的注意点：假设要把 C 分支的代码合并到 A 分支，则必须先切换到 A 分支上，再运行 git merge 命令，来合并 C 分支！

10. 删除分支

当把功能分支的代码合并到 master 主分支上以后，就可以使用如下的命令，删除对应的功能分支（注意，如果你要删除某个分支，那你目前所处的分支不能是那个你要删除的分支）：

```
1 git branch -d 分支名称
```

图示如下：



11. 遇到冲突时的分支合并

如果在两个不同的分支中，对同一个文件进行了不同的修改，Git 就没法干净的合并它们。此时，我们需要打开这些包含冲突的文件然后手动解决冲突。

```
1 # 假设：在把 reg 分支合并到 master 分支期间，代码发生了冲突
2 git checkout master
3 git merge reg
4
5 # 打开包含冲突的文件，手动解决冲突之后，再执行如下的命令
6 git add .
7 git commit -m "解决了分支合并冲突的问题"
```

这里需要在master分支上打开文件，然后修改就可以了

Git 分支 - 远程分支操作

1. 将本地分支推送到远程仓库

如果是第一次将本地分支推送到远程仓库，需要运行如下的命令：

(这里的origin好像是默认的，就是远程仓库的名字)

```
1 # -u 表示把本地分支和远程分支进行关联，只在第一次推送的时候需要带 -u 参数
2 git push -u 远程仓库的别名 本地分支名称:远程分支名称
3
4 # 实际案例:
5 git push -u origin payment:pay
6
7 # 如果希望远程分支的名称和本地分支名称保持一致，可以对命令进行简化:
8 git push -u origin payment
```

注意：第一次推送分支需要带 -u 参数，此后可以直接使用 git push 推送代码到远程分支。

2. 查看远程仓库中所有的分支列表

通过如下的命令，可以查看远程仓库中，所有的分支列表的信息：

```
1 git remote show 远程仓库名称
```

git remote show origin(这个命令基本上是固定的，因为远程仓库名称是固定的origin)

3. 跟踪分支

跟踪分支指的是：从远程仓库中，把远程分支下载到本地仓库中。需要运行的命令如下：

```
1 # 从远程仓库中，把对应的远程分支下载到本地仓库，保持本地分支和远程分支名称相同
2 git checkout 远程分支的名称
3 # 示例：
4 git checkout pay
5
6 # 从远程仓库中，把对应的远程分支下载到本地仓库，并把下载的本地分支进行重命名
7 git checkout -b 本地分支名称 远程仓库名称/远程分支名称
8 # 示例：
9 git checkout -b payment origin/pay
```

4. 拉取远程分支的最新的代码

可以使用如下的命令，把远程分支最新的代码下载到本地对应的分支中：

```
1 # 从远程仓库，拉取当前分支最新的代码，保持当前分支的代码和远程分支部代码一致
2 git pull
```

5. 删除远程分支

可以使用如下的命令，删除远程仓库中指定的分支：

```
1 # 删除远程仓库中，指定名称的远程分支
2 git push 远程仓库名称 --delete 远程分支名称
3 # 示例：
4 git push origin --delete pay
```

小结

1. 能够掌握 Git 中基本命令的使用
 1. git init
 2. git add .
 3. git commit -m "提交消息"
 4. git status 和 git status -s
2. 能够使用 Github 创建和维护远程仓库
 1. 能够配置 Github 的 SSH 访问
 2. 能够将本地仓库上传到 Github
3. 能够掌握 Git 分支的基本使用
 1. git checkout -b 新分支名称

2. git push -u origin 新分支名称
3. git checkout 分支名称
4. git branch