接口测试必备理论介绍

接口英文全称是Application Interface Program,简称API

1 接口的发展历史

早期前后端不分离时，并没有没有把接口独立出来。

后来随着互联网技术的发展，前端技术和后端技术都得到了广泛的发展。

前端开始专注于数据的展示，例如：Web浏览器、App原生界面、H5页面。

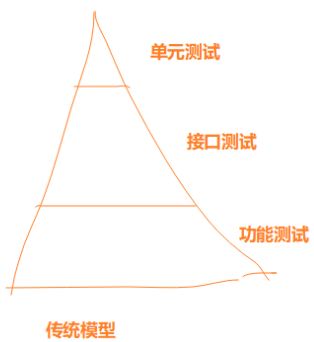
后端专注于数据的逻辑处理。

为了让前端和后端能够顺利的进行数据的传输，后端开发人员专门为前端设计了接口，方便传输数据。

所以我们的接口测试，主要是对后端功能进行测试。而前端主要是使用后端提供的接口，来完成各种页面，面向用户使用。

1.1 接口测试在互联网测试中的地位

在互联网测试中，有一个测试原则：越早介入测试越好。在整个测试流程中，接口测试比功能测试更早介入，所以能让测试工作比功能测试更早展开，更有效的利用测试时间。



单元测试的缺点：

- 对技术要求高：懂代码
- 对项目熟悉程度要求高：熟悉项目中每个模块的作用，才能进行测试

功能测试的缺点：

- 测试时间周期长
- 对后端功能的专项测试点难以验证

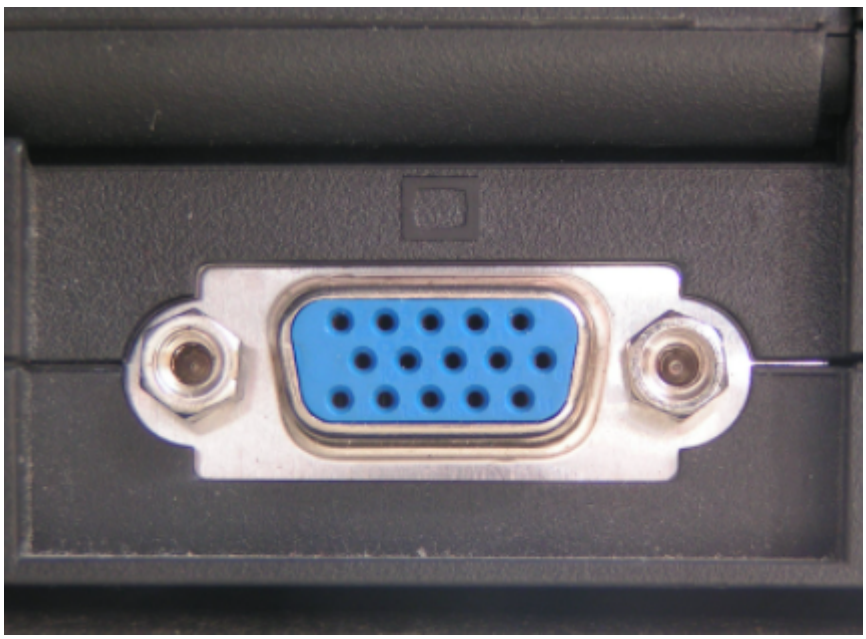
    - 并发测试
    - 性能测试
    - 安全测试
- 难以进行自动化测试

注意：功能测试是必不可少的部分。无论是安全测试还是性能测试，都需要功能测试通过才能进行。

2 接口基础概念

2.1 接口和接口测试的概念

生活中的接口：耳机插头和插孔、电源插座插孔和插头等等、拼图拼接部分



互联网中的接口：是系统之间传输数据的通道。

一个接口主要由输入和输出两个部分组成。

案例：拉勾招聘搜索示例

接口的特点

- 一定的规范要求（协议）
- 能灵活自定义的部分（开发）

2.2 接口的分类

在互联网中，技术层非常多，所以对应的接口类型也有好几种，

按照协议划分：HTTP、FTP、TCP\UDP\IP、Dubbo等协议接口

按照语言划分：Python、Java、C、C++

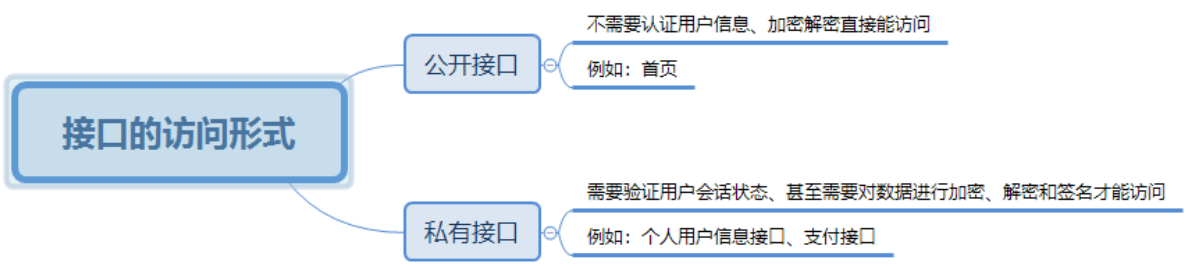
按照内外系统划分：内部系统接口和外部系统接口（又叫第三方接口）

按照技术应用划分： HTTP协议、Web Service、RESTful、RPC远程过程调用型、Web Socket、FTP、Dubbo协议

在实际工作中，我们所说的软件接口测试，主要是针对HTTP、RESTful、Web Service这三种表现形式的接口进行测试。

移动端Api的接口：主流接口测试也是针对Web Api接口进行测试。

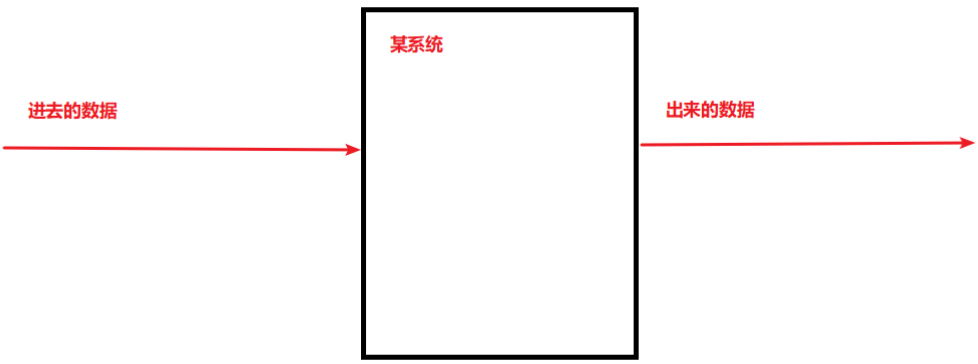
2.3 接口的访问形式



2.4 如何设计一个接口

首先需要理解，在计算机的世界，一切都是数据。

所以，开发们在设计程序时，只需要考虑怎么获取数据、用数据做什么以及返回什么样的数据



在应用层，开发们一般使用HTTP协议来传输数据，所以开发们会基于HTTP协议来设计接口，这也是接口测试的重点测试对象。

下面以flask框架，演示开发设计接口的过程

案例：实现模拟访问html页面和拉勾招聘

```
# -*- encoding=utf-8 -*-
# flask 一个轻量级的web开发框架，与django不同的是，django是重量级的web开发框架
# 安装flask pip install flask
# 导入Flask
from flask import Flask

# 创建Flask对象
app = Flask(__name__)

# 创建接口了，index.html的接口
```

```
@app.route("/index.html")
def index():
    return '''{"status":"1", "msg":"操作成功! "}'''

if __name__ == '__main__':
    # 启动Flask服务器
    app.run()
```

3 HTTP协议

在互联网软件测试中，大部分接口使用HTTP协议进行接口开发。

3.1 HTTP协议的概念

HTTP协议是超文本传输协议，它主要规定了在互联网中传输数据时的标准。

特点：

- 支持客户端/服务器模式
- 简单快速
- 灵活
- 无连接
- 无状态

3.2 HTTP协议的组成部分

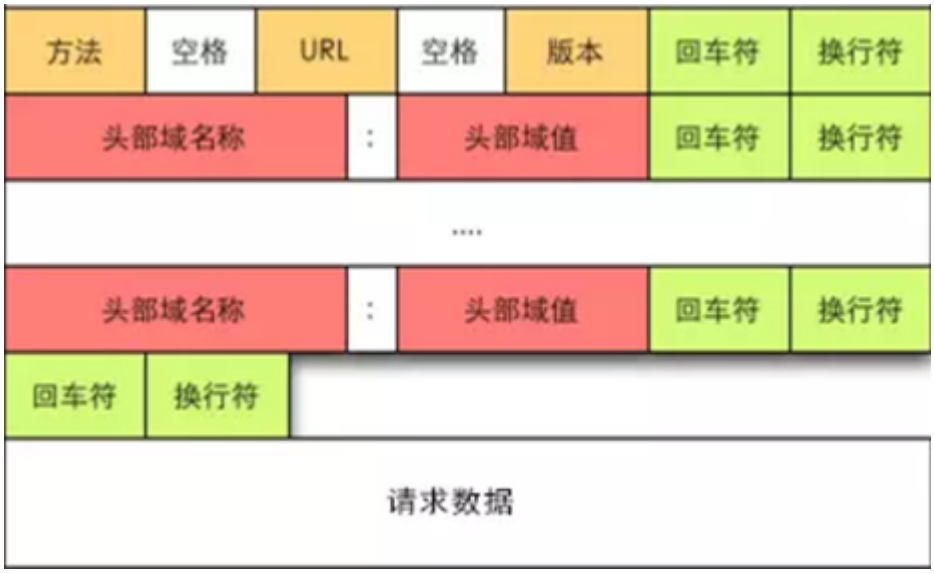
按照HTTP协议规定，传输客户端请求报文时，数据包括三个部分：

- 请求行
- 请求头
- 请求体

传输服务器响应报文时，数据包括三个部分：

- 状态行
- 响应头
- 响应正文

3.3 HTTP请求



请求行：从一个HTTP请求报文里，开始第一行的内容都是请求行

请求头：第一行下面，空行之前都是请求头；

请求体：空行之后，都是请求体；

请求行、请求头、请求体都可以传输具体数据

请求数据示例报文：

```
POST http://localhost/index.php?m=Home&c=User&a=do_login&t=0.005897075... HTTP/1.1
Host: localhost
Connection: keep-alive
Content-Length: 53
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/88.0.4324.182 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/index.php/Home/user/login.html
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
```

```
Cookie: province_id=1; city_id=2; district_id=3;
parent_region=%5B%7B%22id%22%3A3%2C%22name%22%3A%22%u4E1C%u57CE%u533A%22%7D%2C%7B%22id%22
%3A14%2C%22name%22%3A%22%u897F%u57CE%u533A%22%7D%2C%7B%22id%22%3A22%2C%22name%22%3A%22%u5
D07%u6587%u533A%22%7D%2C%7B%22id%22%3A30%2C%22name%22%3A%22%u5BA3%u6B66%u533A%22%7D%2C%7B
%22id%22%3A39%2C%22name%22%3A%22%u671D%u9633%u533A%22%7D%2C%7B%22id%22%3A83%2C%22name%22%
3A%22%u4E30%u53F0%u533A%22%7D%2C%7B%22id%22%3A105%2C%22name%22%3A%22%u77F3%u666F%u5C71%u5
33A%22%7D%2C%7B%22id%22%3A115%2C%22name%22%3A%22%u6D77%u6DC0%u533A%22%7D%2C%7B%22id%22%3A
145%2C%22name%22%3A%22%u95E8%u5934%u6C9F%u533A%22%7D%2C%7B%22id%22%3A159%2C%22name%22%3A%
22%u623F%u5C71%u533A%22%7D%2C%7B%22id%22%3A188%2C%22name%22%3A%22%u901A%u5DDE%u533A%22%7D
%2C%7B%22id%22%3A204%2C%22name%22%3A%22%u987A%u4E49%u533A%22%7D%2C%7B%22id%22%3A227%2C%22
name%22%3A%22%u660C%u5E73%u533A%22%7D%2C%7B%22id%22%3A245%2C%22name%22%3A%22%u5927%u5174%
u533A%22%7D%2C%7B%22id%22%3A264%2C%22name%22%3A%22%u6000%u67D4%u533A%22%7D%2C%7B%22id%22%
3A281%2C%22name%22%3A%22%u5E73%u8C37%u533A%22%7D%5D; is_mobile=0; CNZZDATA009=30037667-
1536735; is_distribut=0; PHPSESSID=krl4miah115o2nmr6n5nobmh24; cn=0

username=13800138006&password=123456&verify_code=TViw
```

其中【parent_region=%5B%7B%22id%22%3A3...】中的符号，是一些特殊字符经过encoded之后的数据

需要encoded的原因是URL中，不支持传输中文数据

URL

又称为统一资源定位符，帮助定位互联网网络资源的地址。

网络资源：包括服务器资源、图片资源、数据等等

http://localhost:80/index.php?m=Home&c=User&a=do_login&t=0.005897075...

URL由5个部分组成：

- 协议部分：http
  规定传输数据的协议是什么，常见的包括：http、ftp、https等等
- 域名部分：localhost
  决定了要访问的服务器网络地址，这个域名会被DNS服务器解析成IP地址，通过IP来定位服务器资源地址。
- 端口部分：http协议默认80端口（默认：没有填写端口时，就采用默认的端口）
  端口部分就是服务器内部的应用的端口。（https的默认端口是：443端口，ftp的默认端口是21端口）
- 资源路径：/index.php
  定位到服务器具体代码的路径
- 查询参数：m=Home&c=User&a=do_login&t=0.005897075..
  是具体传递的数据，

快递地址：国家、省份、城市、街道、楼房，楼房中的单元号就是端口

资源路径相当于某个单元房间中内部的厨房、卧室等内容

查询参数：相当于传递的数据

3.3.1 请求行

请求行：

```
POST http://localhost/index.php?m=Home&c=User&a=do_login&t=0.0058970756... HTTP/1.1
```

- 请求行的每个数据之间，用空格做分隔符
- 依次分别是：请求方法、URL、协议/版本

请求方法：描述对资源操作的动作

在HTTP1.1中，常见请求方法包括：

- GET：获取和查询数据
- POST：新增数据
- PUT：修改数据
- DELETE：删除数据

其他请求方法还有：OPTIONS、HEAD、PATCH等等

URL（Uniform Resource Locator）：

又称为统一资源定位符，在互联网中，都是用URL来帮助定位**网络资源。**

网络资源： 图片、网页、数据、多媒体等等

HTTP协议没有对URL长度做出限制，但是浏览器对URL长度有限制

ie：2182个字符

为了兼容性，URL长度不要超过IE规定的长度2182个字符

URL 由5部分组成：

```
http://localhost:80/index.php?m=Home&c=User&a=do_login&t=0.005897075610153069
```

协议部分：http；其他协议还有mysql、ftp等等

域名部分：localhost，域名会被解析成IP地址来访问指定的资源。

端口部分：80，http协议默认80端口，https默认443端口。不填写端口时使用默认的端口。

资源路径：/index.php；资源路径相当于服务器开放的文件系统，我们可以通过资源路径访问服务器中开放的资源。

查询参数：m=Home&c=User&a=do_login&t=0.005897075610153069

传递给服务器的数据。多组查询参数用"&"作分隔符。

如果把URL比喻成快递，那么协议部分决定了用顺丰快递还是申通快递，它是指选择不同的传输方式。

域名部分就是传到哪个具体位置的楼房：北京海淀区海置创投大厦

端口部分就是指这栋楼的那一层：二楼

资源路径：指这层楼中的哪一个单元房间

查询参数：传递给这个单元房间的物品。

3.3.2 请求头

请求头主要用于存放一些不常更新的数据，例如：客户端请求的浏览器说明，客户端域名，Cookie等等

请求头主要由键值对组成，每行一对

结构： 键名：属性值

```
Host: localhost
Connection: keep-alive
Content-Length: 53
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/88.0.4324.182 Safari/537.36
```

在开发时，开发可以自定义请求头的键和属性值。

也可以对HTTP协议规定的固定的键值对进行二次开发。

常用的请求头有：

- User-Agent：描述客户端的浏览器信息
- Content-Type：描述请求体的内容格式

  Content-Type的值包括很多种，常见的有：

  - text/plain 文本格式
  - text/xml xml格式
  - x-www-form-urlencoded 表单格式；不能上传文件、图片、音视频
  - multipart/form-data 二进制表单格式，能用来上传文件、图片、音视频
  - application/json

3.3.3 请求体

请求体用来传输数据。和URL中的查询参数不一样的是，URL的数据直接可以在浏览器地址栏看到，而请求体不能直接看到。并且，请求体能传输的数据类型、数据大小都比URL要多

实际测试中，我们测试人员主要是按照开发设计，对请求体的数据内容进行自由设计，然后测试服务器返回的相应数据与预期是否一致。

例如：开发设计了一个添加课程的接口，要求输入

```
课程标题title  字符串 不超过50个字符

课程备注remark 字符串 不超过500个字符

课程价格price 字符串  保留两位小数 一共10位

课程标签tag 字符串 不超过10个字符
```

那么我们就需要**对输入数据进行用例设计，测试服务器的响应数据与需求规定的预期是否一致**。

同样的一组数据，在不同的Content-Type属性值时，会有不同的编写方式，具体的编写方式，其中的内容是由开发指定，但是结构是固定的：

- Content-Type:applicaiton/json

  ```
  {"title":"测试课程", "remark":"干货满满，学完就能就业", "price":"5.20", "tag":"测试"}
  ```

- Content-Type：text/xml

  ```
  <?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <addCourse xmlns="http://WebXml.com.cn/">
        <title>测试课程</title>
        <remark>干货满满，学完就能就业</remark>
        <price>5.20</price>
        <tag>测试</tag>
      </addCourse>
    </soap:Body>
  </soap:Envelope>
  ```

- Content-Type：x-www-form-urlencoded

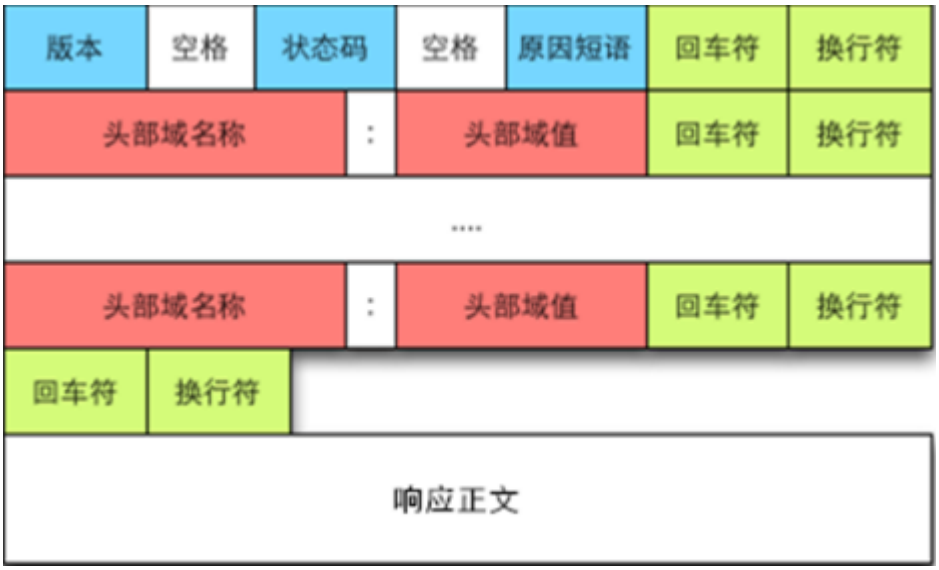
  ```
  title=测试课程&remark=干货满满，进阶提升找工作&price=5.20&tag=测试
  ```

案例：

使用Fiddler抓取拉勾教育操作过程产生的网络报文，也就是接口的请求和响应数据。

3.4 HTTP响应

HTTP响应包括状态行、响应头和响应体



状态行：响应报文中，第一行是状态行

响应头：状态行下面，空行之前都是响应头

响应正文：空行之后，都是响应正文

响应数据示例报文：

```
HTTP/1.1 200 OK
Server: nginx/1.11.5
Date: Fri, 09 Apr 2021 08:28:01 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.6.27
Set-Cookie: is_mobile=0; expires=Fri, 09-Apr-2021 09:28:00 GMT; Max-Age=3600; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: user_id=8; path=/
Set-Cookie: is_distribut=0; path=/
Set-Cookie: uname=summer; path=/
Set-Cookie: cn=0; expires=Fri, 09-Apr-2021 07:28:01 GMT; Max-Age=-3600; path=/
Content-Length: 999
```

```
{"status":1,"msg":"\u767b\u9646\u6210\u529f","result":
{"user_id":8,"email":"","password":"519475228fe35ad067744465c42a19b2","paypwd":"519475228
fe35ad067744465c42a19b2","sex":0,"birthday":0,"user_money":"99861.00","frozen_money":"0.0
0","distribut_money":"0.00","underling_number":0,"pay_points":100000,"address_id":0,"reg_
time":1523857661,"last_login":1617956749,"last_ip":"","qq":"","mobile":"13800138006","mob
ile_validated":1,"oauth":"","openid":null,"unionid":null,"head_pic":"http:\/\/thirdwx.qlo
go.cn\/mmopen\/vi_32\/c58Iiaib1aPodvKHMMGR9ZYmq7XGFUgppvhxgQKrJxdlZTAauZ8dTucEguiamsncVDR
3h32TMO4YzppDmSuHIGI9w\/132","province":0,"city":0,"district":0,"email_validated":0,"nick
name":"summer","level":2,"discount":"1.00","total_amount":"605.00","is_lock":0,"is_distri
but":0,"first_leader":3,"second_leader":0,"third_leader":0,"token":"","message_mask":63,"
push_id":"190e35f7e07c8658ec6","distribut_level":0,"is_vip":0,"xcx_qrcode":null,"poster":
null,"level_name":"\u5014\u5f3a\u9752\u94dc"},"url":""}
```


3.4.1 状态行

状态行的作用是：描述服务器处理客户端请求的结果

```
HTTP/1.1 200 OK
```

状态行也由3个部分组成：协议/版本 状态码 状态消息

协议/版本： 该响应报文的所使用的协议和版本号

状态码： 描述服务器处理请求的结果；状态码由3位数字组成，可以分成5类

- 1XX：代表客户端请求已经被接收了，继续处理
- 2XX：代表客户端请求被服务器按照内部逻辑正确处理成功了
- 3XX：资源已经失效，重定向到新的地址
- 4XX：客户端错误
- 5XX：服务器错误

注意：200代表服务器处理成功，但是并不代表业务成功。

例如：登陆成功和密码错误服务器都会返回200的状态码，可是在用户看来一个是登陆成功，一个是登陆失败。

状态消息： 对状态码的解释说明

常见状态码

状态码	状态消息	请求方法	说明
200	OK	GET	服务器成功返回用户请求的数据
201	created	POST/PUT	用户新建或修改数据成功
204	NO CONTENT	DELETE	用户删除数据成功
400	Bad Request	POST/PUT	客户端请求有语法错误，不能被服务器所理解
401	Unauthorized	*	表示用户没有权限（令牌、用户名、密码错误）
403	Forbidden	*	禁止访问
404	Not Found	*	请求资源不存在
500	Internal Server Error	*	服务器发生错误，客户端无法判断发乎的请求是否成功
503	Server Unavailable	*	服务器当前不能处理客户端请求，一段时间后可能恢复正常

3.4.2 响应头

响应头和请求头结构和内容都一样，不同的是响应头是对响应数据的描述。

例如，响应头中的Content-Type，描述的是响应正文的内容格式类型

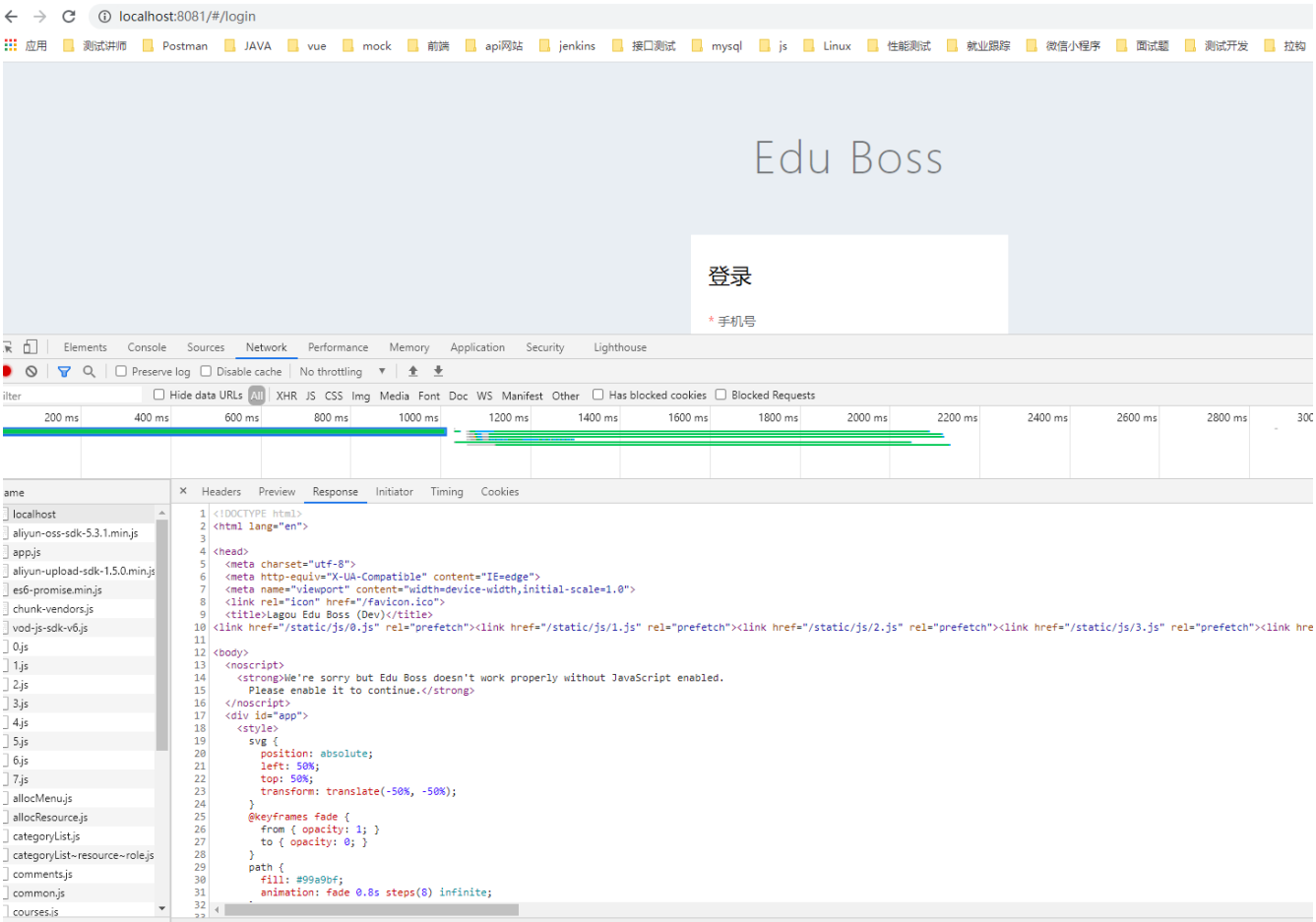
3.4.3 响应正文

经过服务器处理后的响应数据。

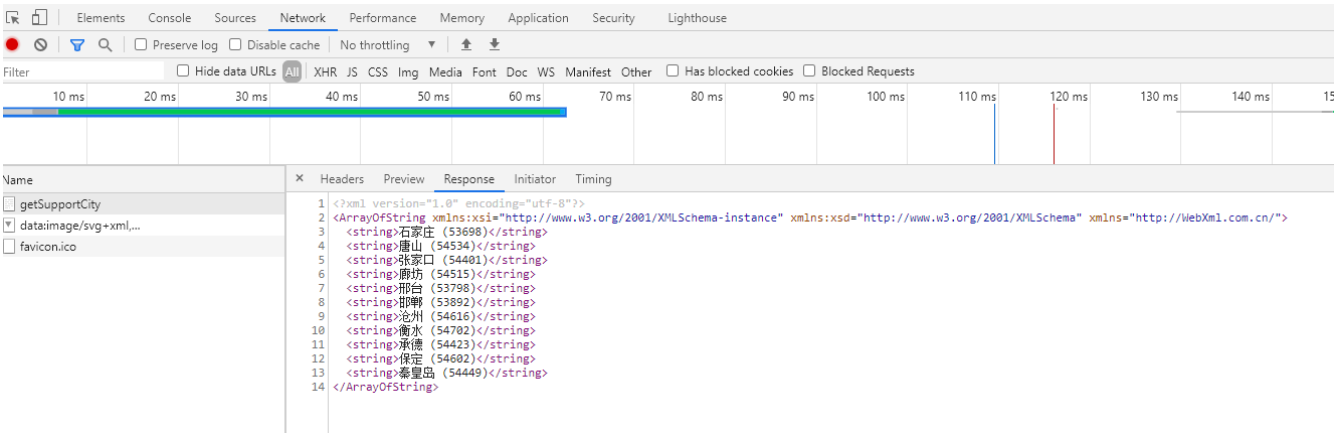
响应数据类型包括：html、xml、text、json、图片等等

案例

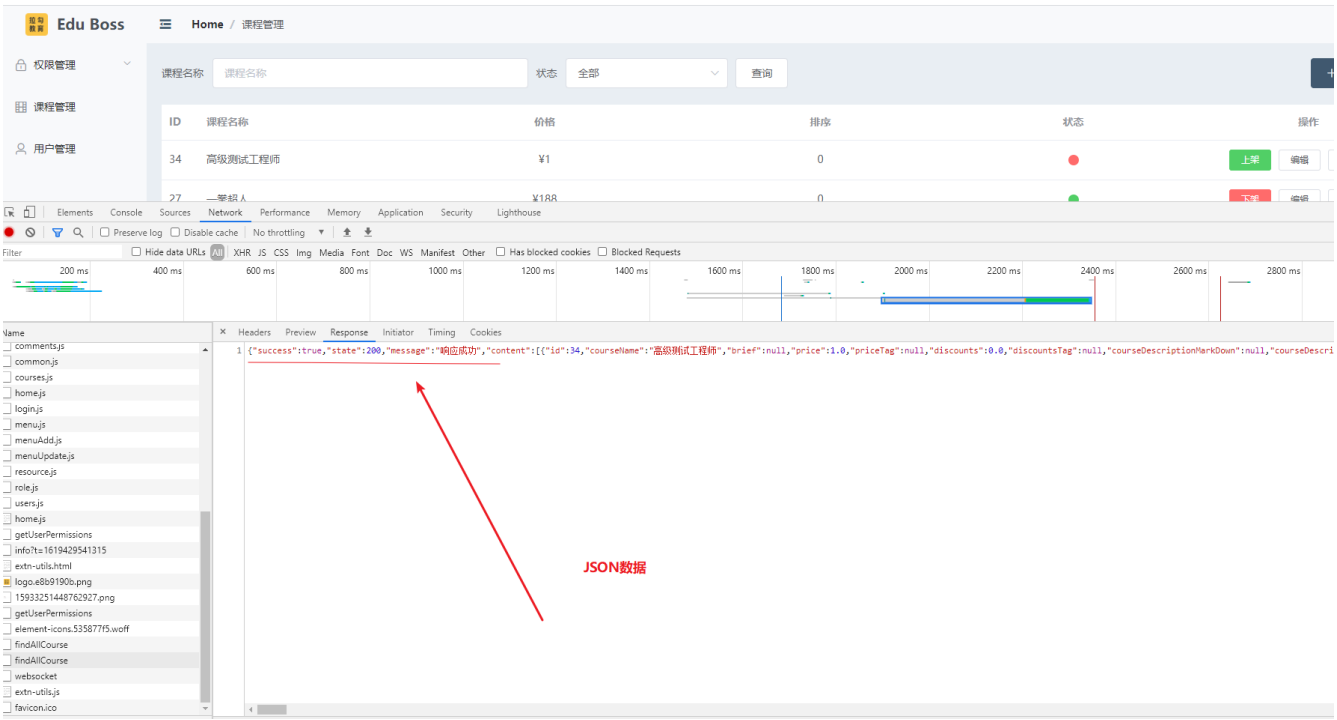
查看HTML响应



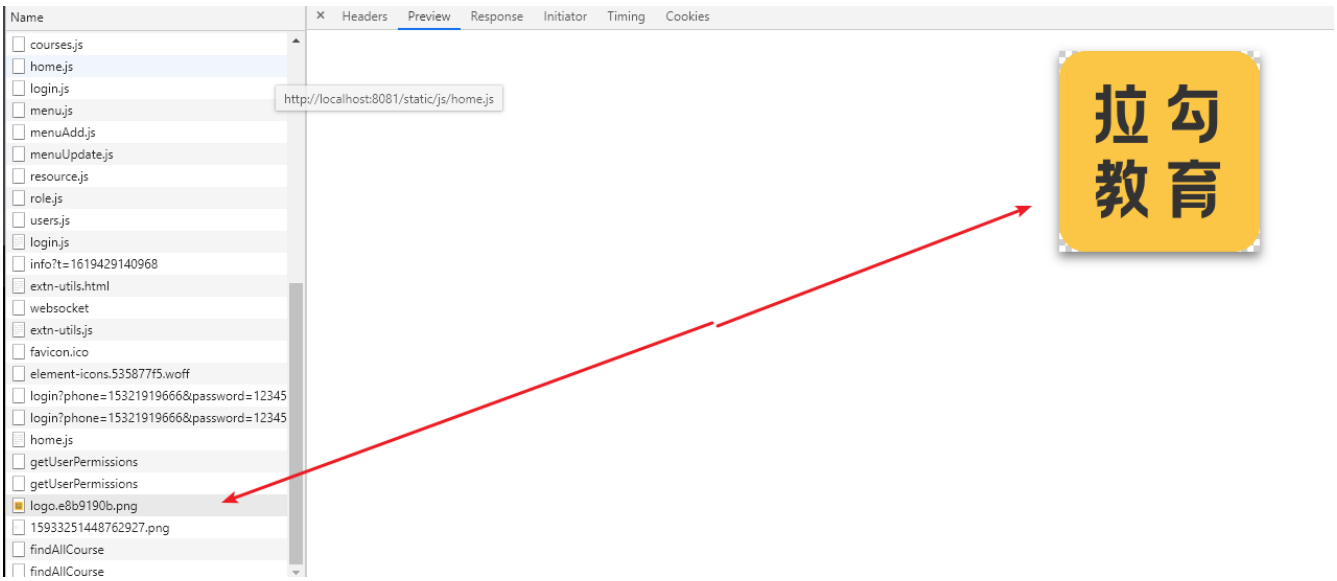
查看XML响应



查看JSON响应



查看图片响应



3.5 HTTPS

HTTPS是HTTP协议+SSL/TLS（Secure Sockets Layer/Transport Layer Security ）认证，是一个基于安全套接字的超文本传输协议

3.5.1 HTTPS的作用

HTTPS主要用于对网络传输的数据进行加密，保证数据的安全性，完整性，一致性。

在**测试参与的接口测试工作中，一般测试环境都用HTTP协议**，生产环境使用HTTPS协议

3.5.2 HTTP与HTTPS的区别

安全性：
HTTP 是明文传输，数据在传输过程中是未加密的，容易被中间人窃听、篡改或劫持。
HTTPS 使用 SSL/TLS 协议对传输的数据进行加密，提供了安全性保障，确保数据在传输过程中不被窃听、篡改或劫持。
数据传输：
HTTP 数据传输是明文的，没有加密处理，因此传输速度相对较快。
HTTPS 数据传输是经过加密的，因此会增加一定的传输延迟和计算负载，相对于 HTTP 来说传输速度略慢一些。
端口号：
HTTP 默认使用端口号80进行通信。
HTTPS 默认使用端口号443进行通信。
证书：
HTTPS 使用数字证书来验证服务器的身份，客户端与服务器建立连接时会进行证书交换和验证，确保通信双方的身份和通信的安全性。
HTTP 不需要证书，通信的双方直接进行数据传输。
URL：
HTTP 的 URL 以 "http://" 开头。
HTTPS 的 URL 以 "https://" 开头。
总的来说，HTTPS 是在 HTTP 的基础上加入了 SSL/TLS 加密机制，提供了更高的安全性，适用于对数据传输安全性有要求的场景，如网上支付、用户登录等。而 HTTP 则适用于对安全性要求不高的场景，如浏览网页、传输公共信息等。

3.5.3 深入理解HTTPS

HTTP协议是超文本传输协议，而HTTPS只是多了一个SSL/TLS认证。

所以理解了SSL/TLS认证，就理解了HTTPS。

SSL/TLS认证，是通过一个Ca证书完成的，这个证书可以在本地生成，也可以申请。

但是在本地生成的证书，需要客户端信任才能访问。

申请证书： 向权威机构申请证书

密码基础概念：

明文：直接能看懂的文本内容

密文：经过加密后的内容

加密：对数据进行处理，使数据被外部人员无法理解，但是自己人能够理解。

加密算法：对数据加密时，所采用的相关计算方法。

密钥：加密数据和解密数据时，使用的一个字符串，这个字符串起到钥匙的作用。

签名：使用一种超运算加密算法，如MD5,SHA256算法对数据进行运算，形成摘要，然后截取一部分摘要得到的字符串就是签名。签名的作用是确保数据没有被篡改。

对称加密：

加密和解密过程完全对称的加密。

典型的对称加密算法是：DES，3DES加密算法。

特点：

- 加密解密完全对称，加密时使用的密钥和解密时使用的密钥是同一个密钥
- 性能好，速度快

要保证对称加密的安全，需要确保对称加密的密钥不被泄露，可以通过线下的手段来传递密钥，这样就不会被泄露了。

银行：U盾，加密狗。

在互联网中，尤其是电商，由于不可能通过线下的手段来传递密钥，所以只能通过线上来传输密钥。

非对称加密：

加密和解密过程不对称的加密。

非对称加密算法有：RSA加密算法

非对称加密算法中，把密钥分成了公钥和私钥。其中公钥是指公开的密钥，私钥是指不公开的密钥。

使用非对称加密算法加密数据的过程中：

- 公钥加密的数据，只能用私钥解密。

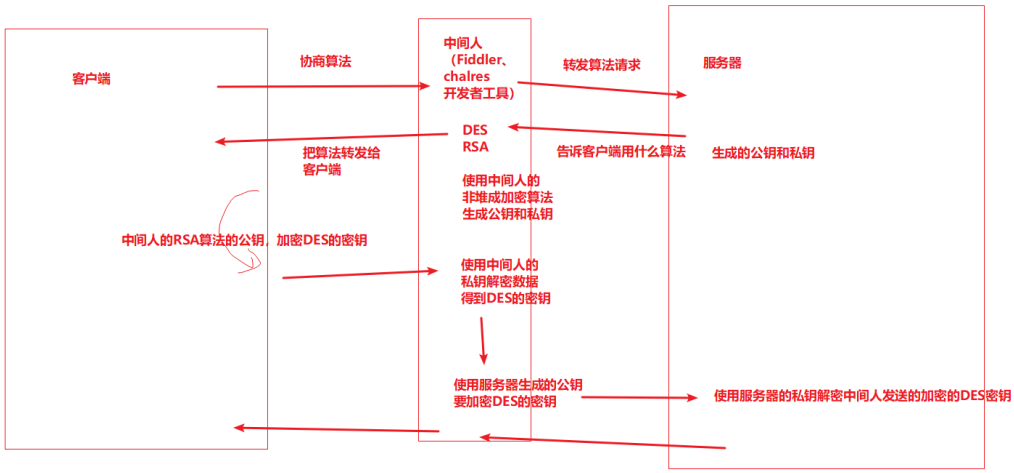
- 私钥加密的数据，只能用公钥解密。

特点：

- 因为加解密不完全对称，所以恶意攻击者不能获取全部信息，更能保证数据的安全性
- 性能比对称加密差

作用：一般用公钥加密客户端生成要用的对称加密算法中的密钥，保证对称加密的密钥不被公开。

非对称加密虽然在一定程度上保证了数据的安全，但是由于黑客可以模拟服务器，进行攻击，所以它也并不安全。



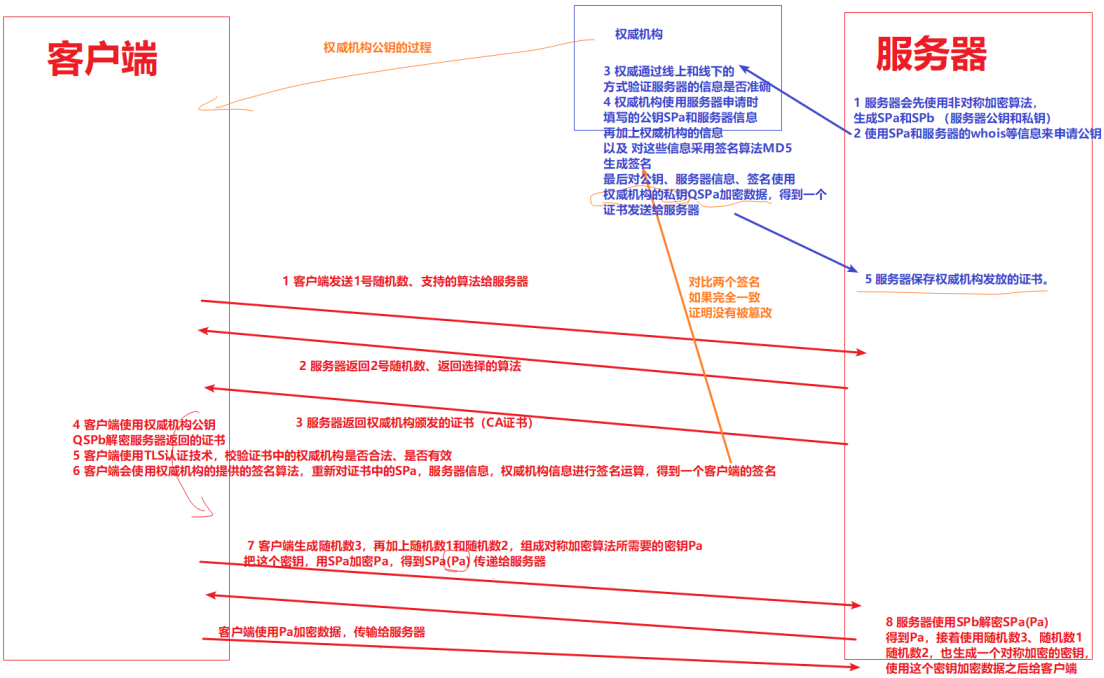
SSL/TLS认证：

SSL认证是一个非常复杂的过程，主要可以认为分成以下几个步骤完成：

- 客户端发送请求，要与服务器建立HTTPS连接
- 服务器返回SSL证书和加密算法给客户端
- 浏览器根据内置的SSL证书验证权威结构来验证SSL证书是否合法、有效
- 浏览器内部使用对称加密算法生成一个密钥，用于加密传输数据
- 浏览器使用SSL证书中的公钥即为Pa，加密对称加密算法生成的密钥记为Ra，得到一个Pa(Ra)加密过后的字符串，然后把它发送给服务器
- 服务器使用私钥解密Pa(Ra)加密过后的字符串，得到Ra
- 后期，客户端和服务器都使用Ra加密数据，保证数据的安全性。

SSL证书是服务器提前申请的

- 服务器先使用非对称加密算法，生成公钥Pa和私钥Pb
- 服务器使用公钥Pa像权威结构申请SSL证书，权威结构验证后，返回SSL证书
    - SSL证书内容包括：服务器相关信息、服务器公钥、以及这些信息的签名



3.6 扩展：TCP/UDP协议

TCP是一个传输层协议，英文全称是：Transmission Control Protocol

UDP也是一个传输层协议，英文全称是：User Datagram Protocol

这两个协议都属于传输层协议。

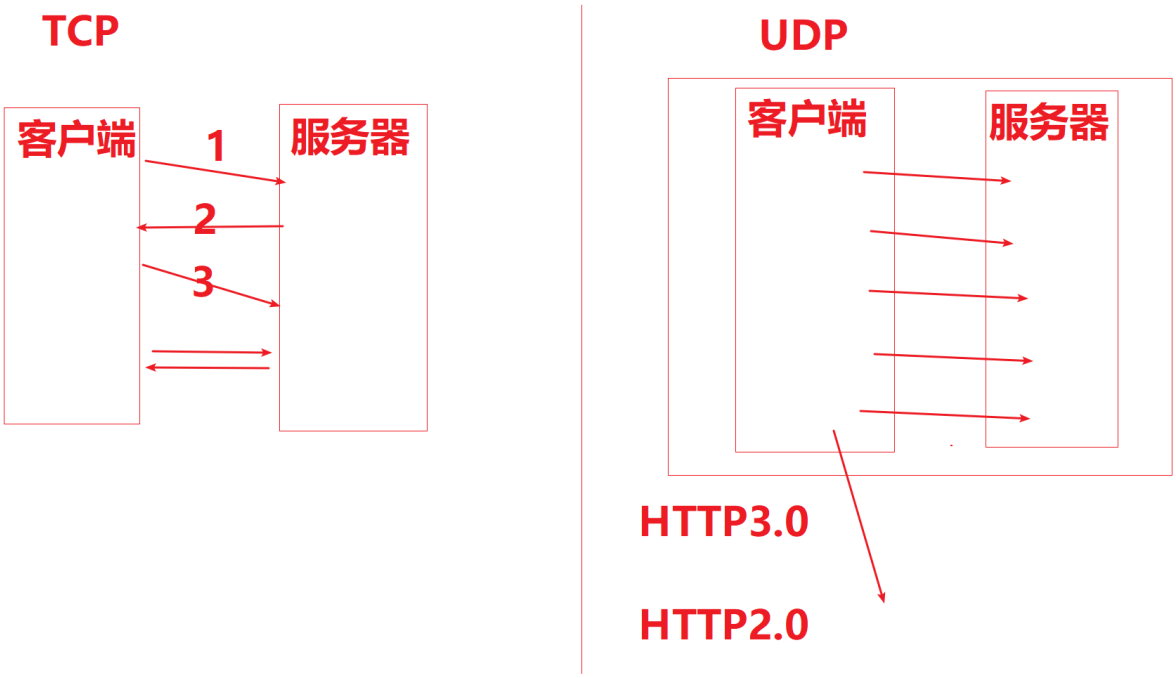
其中，TCP的作用是，通过内部运行机制，保证传输数据的安全性、可靠性。

而UDP不保证安全性和可靠性，只确保传输效率。

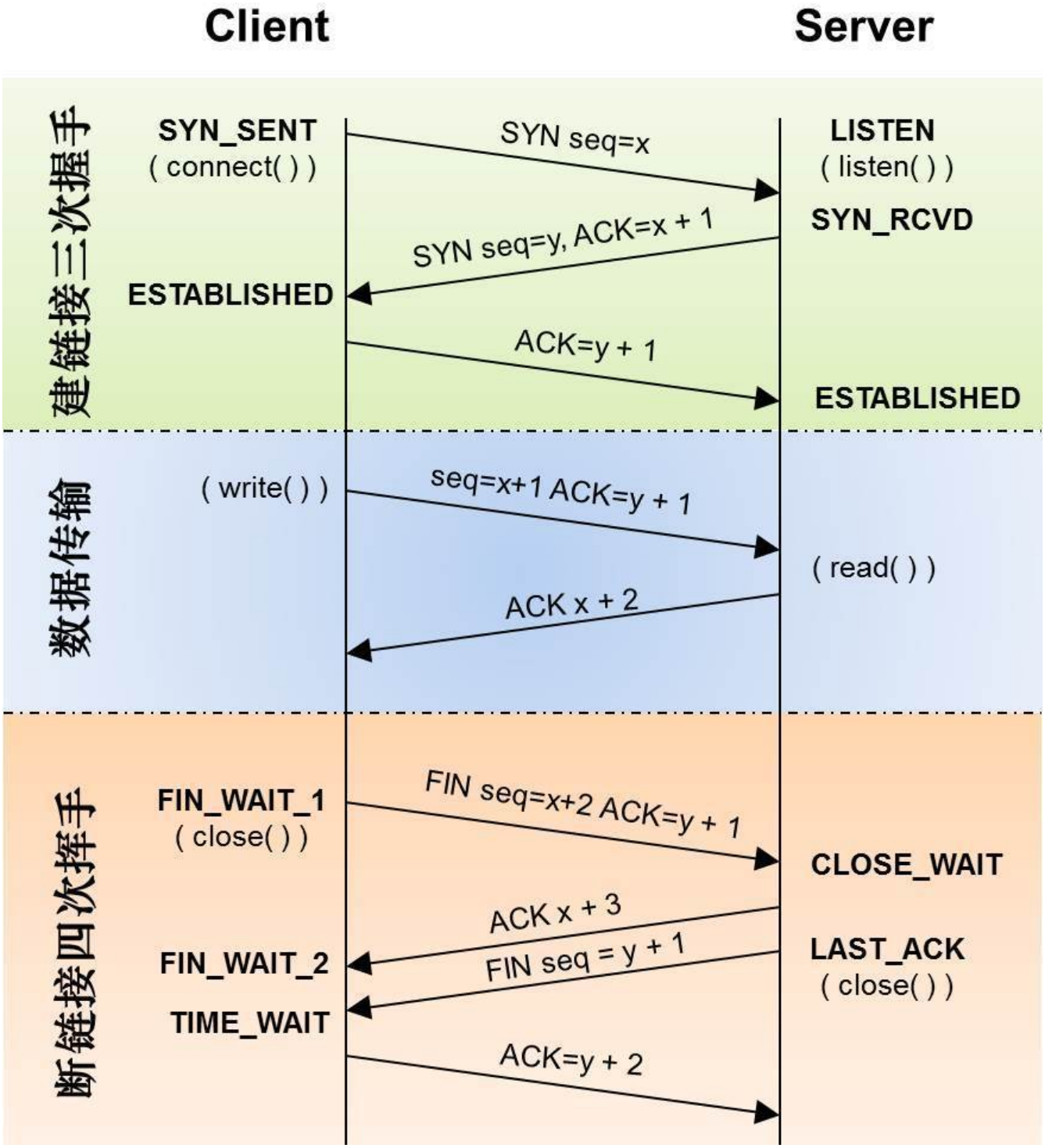
TCP安全可靠的原因：

TCP通过三次握手建立连接，四次挥手断开连接。

UDP是直接发送数据，而不会关注客户端是否收到数据内容。



3.7 扩展：TCP三次握手和四次挥手



三次握手：

- 客户端主动与服务器建立连接
- 客户端设置SYN序号为1，并发送一个seq序号，seq序号随机产生
- 服务器将SYN和ACK位置都设置为1，并把客户端的序号+1，表示应答，然后生成自己的随机序号seq
- 客户端恢复一个ACK，ACK的值是服务器发送的SYN+1

注意：SYN=1，ACK=0是一个连接请求报文的意思。SYN=1,ACK=1是响应报文的意思

四次挥手：

- 客户端发送关闭连接请求FIN=1 和 seq

- 服务器返回ACK，ACK的值是客户端的seq的值加1
- 服务器返回FIN 和 seq
- 客户端返回ACK，ACK的值是服务器seq的值+1

3.8 其他协议的测试

测试的本质：输入和输出

其他协议的接口测试，也是针对输入和输出进行测试。

4 会话管理

4.1 会话的概念

由于HTTP协议无状态的特点，所以客户端和服务器都不会保存客户端请求的相关信息。

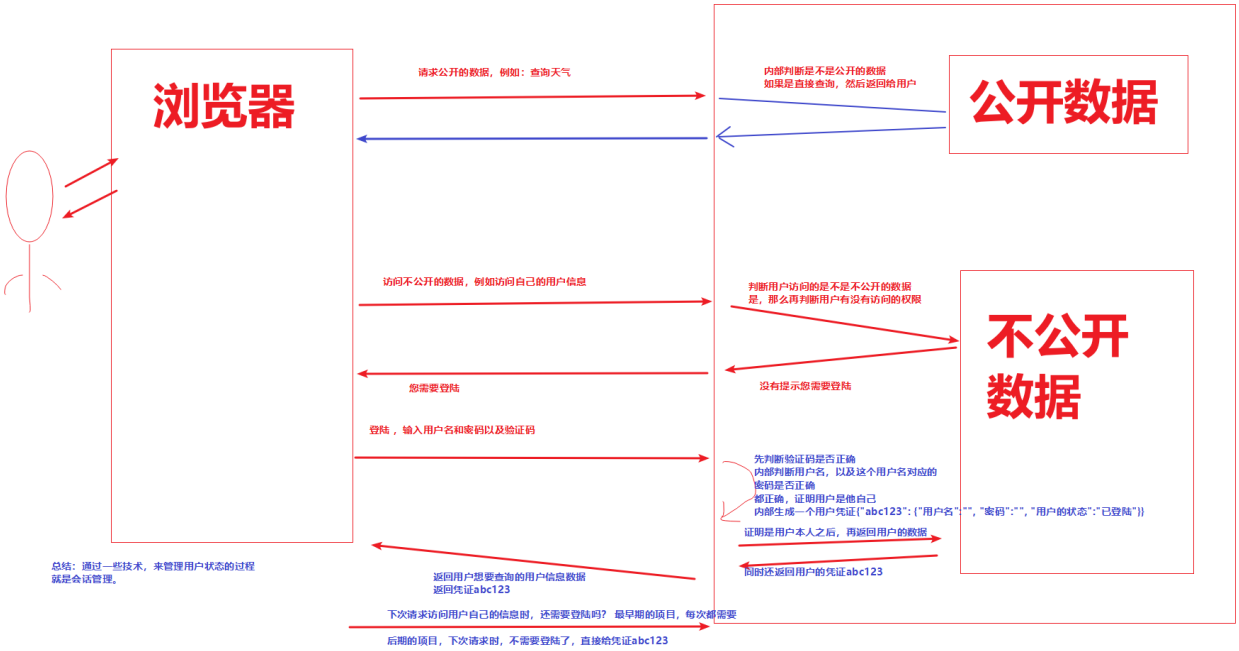
所以服务器无法知道客户端的请求状态是不是已登录的状态，

为了解决这个问题，我们需要通过一个字符串表示用户的未登录/已登陆的状态。然后控制用户对资源的访问。

在现在的Web项目中，管理用户会话主要有3种实现方式：

- 基于Cookie
- 基于Session
- token

典型的会话管理接口也叫做登陆接口或授权接口



4.2 基于Cookie实现会话管理

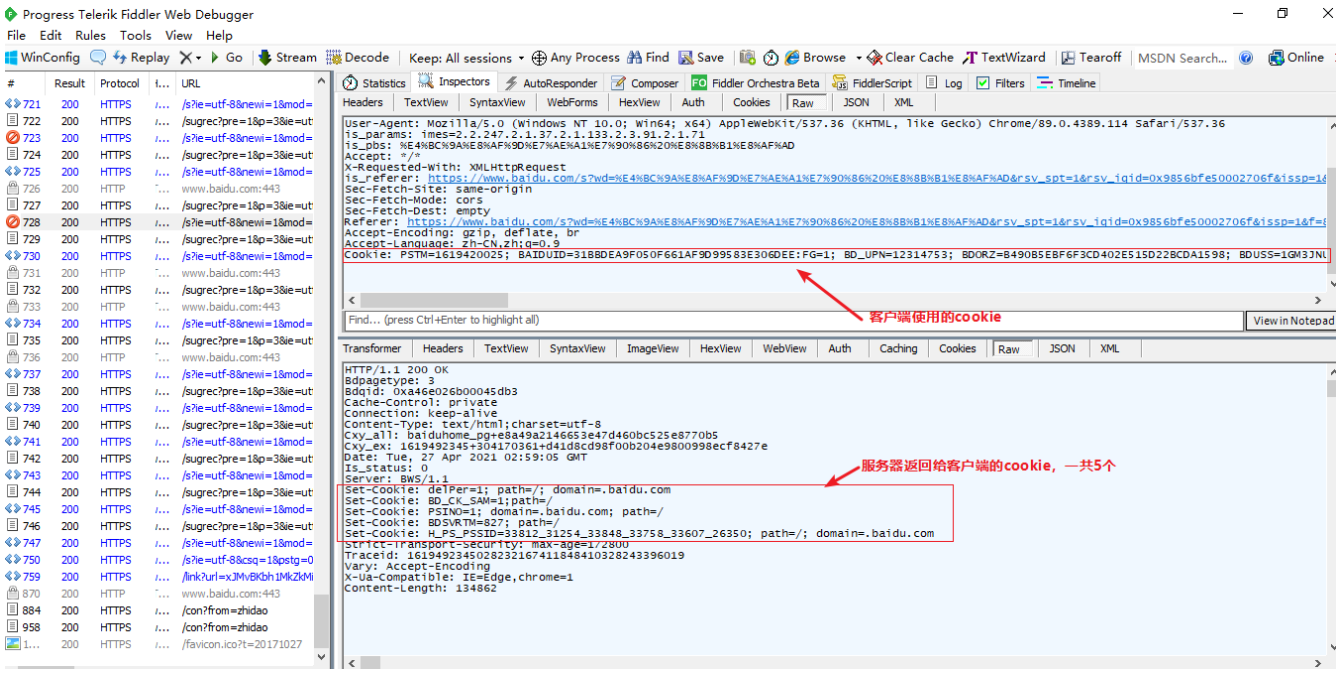
Cookie是一个容器，默认大小4KB，主要用于存放数据大小比较小的临时数据。

Cookie的特点：

- 不能跨域：我们访问服务器时，都会通过域名来访问。例如：www.lagou.com，这个www.lagou.com是域名。如果是我们www.lagou.com域名指向的服务器发送的cookie，那么这个cookie就不能在www.baidu.com中使用。
- 存在有效期

  Cookie存在有效时间，超过规定的时间后就会失效。可以设置Cookie的配置，来让cookie永久不失效。

  不设置有效期时，cookie默认是一个session cookie，浏览器不关闭，cookie不失效
- 大小：默认4KB大小
- 有数量限制：不同浏览器限制的cookie数量不一样，为了保证兼容性，做最低兼容，cookie数量不要超过二十个。
- 对字符编码有限制



Cookie实现会话管理流程

- 用户发起登陆请求，服务器根据用户输入的用户名密码判断是否满足登陆的条件，满足则创建一个用户凭证（字符串）

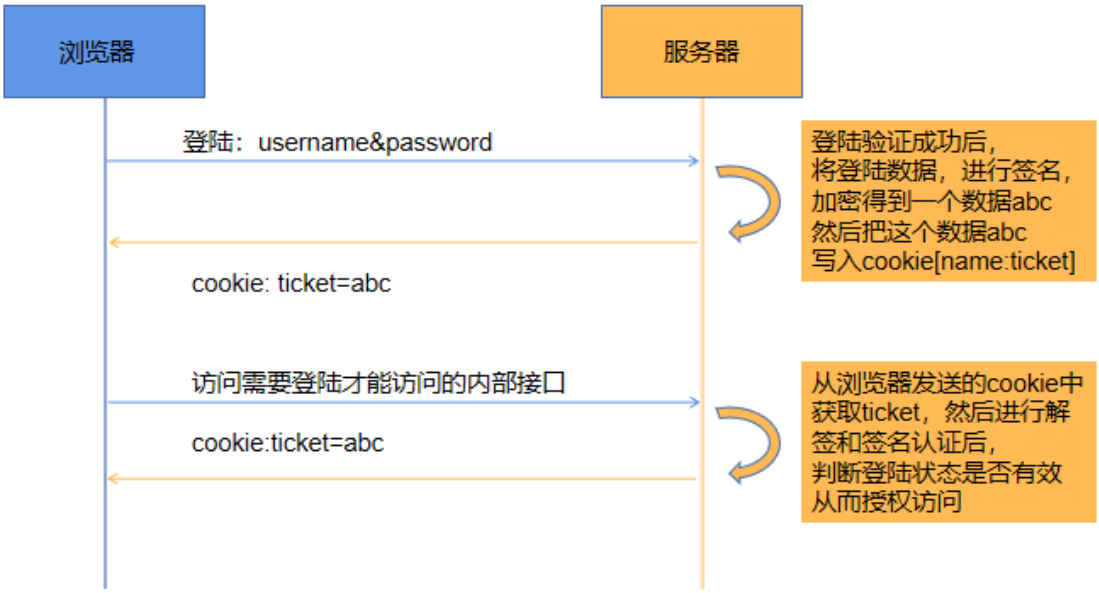
```
md5 ({"username":"taitan", "password":"123456"})  # md5用户数据，生成凭证
```

- 服务器对该用户凭证进行签名，加密后，写在cookie中

```
s = des(md5_data) # 加密md5后的用户数据

cookie[("t":s)] # 保存到cookie
```

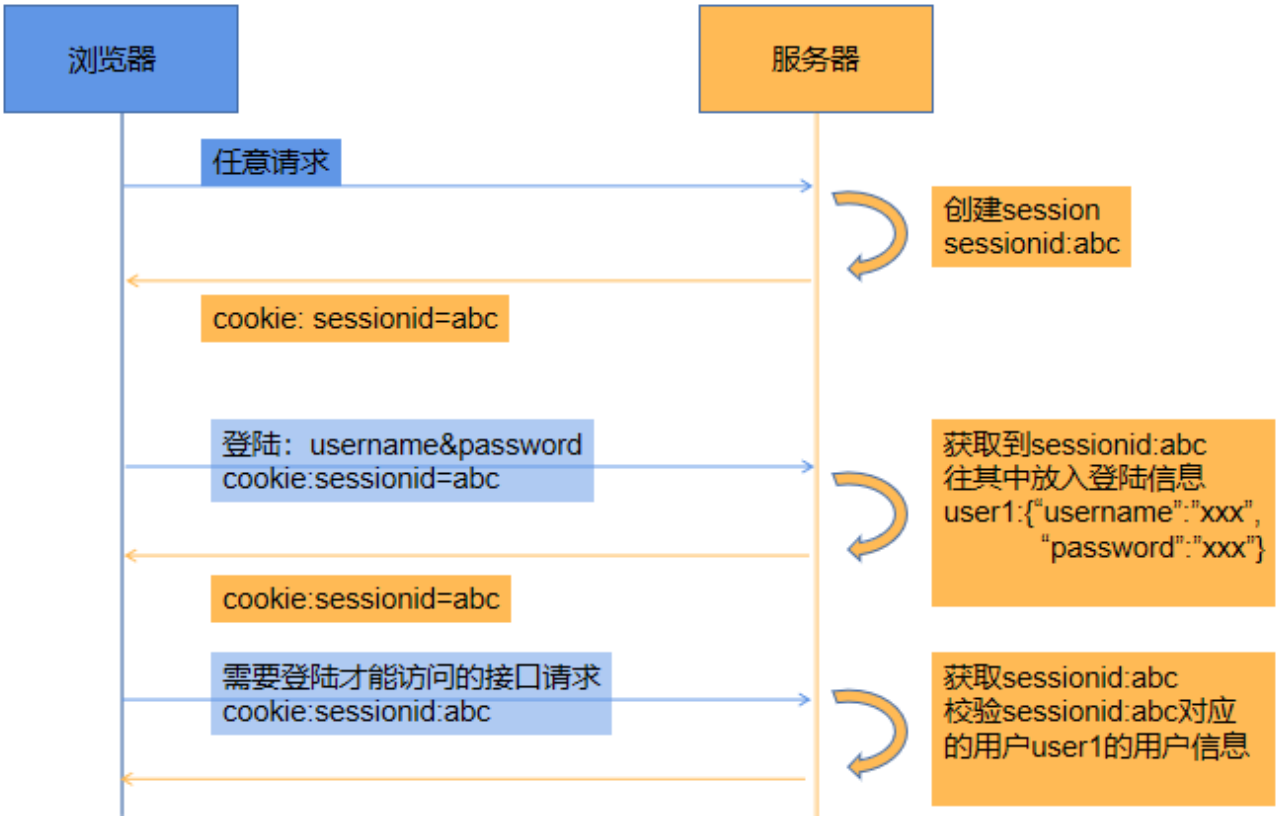
- 服务器返回该cookie给客户端
- 客户端保存cookie，并且下次请求时，会默认发送cookie给服务器，服务器做解密和签名认证后拿到其中的登陆凭证，判断其有效性。判断通过则允许用户访问登陆后的页面，不通过则提示用户登陆



4.3 基于Session实现会话管理

session是把用户数据，放在服务端。

但是客户端依然需要通过sessionid来访问存放在服务端的session，这样才能实现会话管理。



cookie和session的区别

- 大小不同，cookie默认4kb，session无上限。
- 有效期不同，cookie可以设置为永久不失效，而session中，浏览器一旦关闭sessoin就会自动close失效
- 位置不同，cookie在客户端，session在服务器
- 性能不同，cookie在客户端，不会占用太多服务器资源，而session占用服务器资源，对服务器压力较大
- 安全性不同：cookie在客户端不安全。session在服务端更安全

4.4 Token实现会话管理

Token是服务端生成的一串字符串，用来当作客户端请求的令牌，当第一次登录后，服务器根据用户数据的信息，生成token，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。

作用：Token的目的是为了减轻服务器的压力，减少频繁的查询数据库，使服务器更加健壮。

token示例(JWT Token)：

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ3YW5nIiwiY3JlYXRlZCI6MTQ4OTA3OTk4MTM5MywiZXhwIjoxNDg5Njg0NzgxfQ.RC-BYCe_UZ2URtWddUpWXIp4NMsoeq2O6UF-8tVplqXY1-CI9u1-a-9DAAJGfNWkHE81mpnR3gXzfrBAB3WUAg
```

token的结构：

token分成三个部分，头、请求体和签名。每个部分用“.”做分隔符

- 头(header)

  服务器从头信息中，获取相关的加密算法来对请求体进行解密

  ```
  eyJhbGciOiJIUzUxMiJ9 # 它是base64编码得出（base64不是加密，而是编码）
  ```

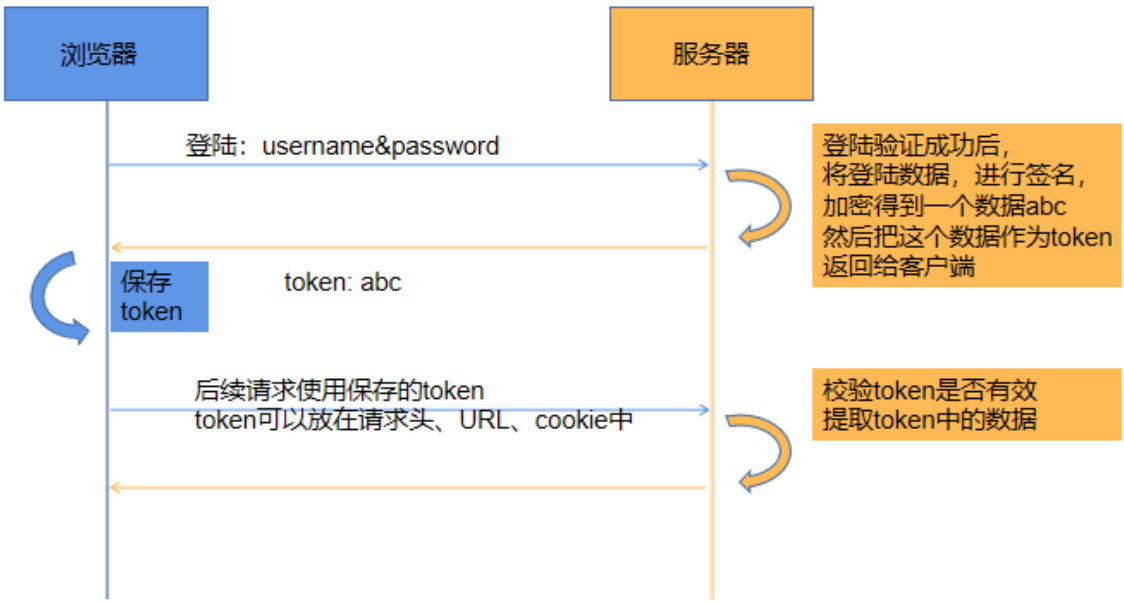
- 请求体(payload)

  服务器使用头信息中的算法解密payload部分后，得到用户信息，然后继续管理用户的状态。

- 签名(signature)

  服务器通过使用头信息中记录的签名算法，来对头和请求体进行签名，再与token中的签名进行比较。

  如果一致，证明token没有被篡改。



4.5 会话的有效时间控制

- cookie实现的会话，可以通过cookie的有效时间配置

客户端无法控制cookie的有效时间，怎么通过服务端代码进行控制。

- session实现的会话，可以通过cookie和session双向的有效时间进行控制
- token的有效时间，可以通过token、cookie和session三个技术的有效时间进行控制
- 此外，实际应用中，还可以通过给接口的参数指定时间戳，然后通过内部代码逻辑校验时间戳。

  应用：12306未支付订单的有效时间

4.6 接口测试难点：登陆接口处理方法

登陆接口是一个带有安全功能的接口，其中包括了用户名、密码、验证码这三块。

其中，密码必须使用内部算法加密；验证码必须让黑客难以识别。这也就带来了无法通过接口对登陆接口进行全面测试的问题。

所以接口测试中，不会全面测试登陆接口，改为通过手工进行登陆功能的测试。

绕过登陆的方法：拿到令牌

- 通过前端页面登陆，拿到令牌
- Mock登陆接口：

  Mock验证码为万能验证码，Mock登陆接口密码为固定密码从而让测试能够进行登陆，拿到令牌
- 白名单控制

  在代码中增加白名单功能，白名单账号无需复杂的校验就可以通过接口登陆，拿到令牌

以上解决方法，除了第一种可以测试人员独立解决以外，其他两种，都需要开发协助才能解决。

```
没有加密的用户数据：{"username":"taitan", "password":"123456", "verify_code":"123456"}
已加密的用户数据：{"username":"taitan", "password":"1c2VybmFtZSI6InRhaXRhb",
"verify_code":"123456"}
```

4.7 扩展：接口签名

接口签名非常好理解，就是把接口的请求数据，进行签名后，再放入该接口的sign中。

主要的目的，就是为了防止接口信息被非法篡改，保证每次接口请求数据的一致性和完整性。

但是它并不能阻止黑客查看接口的数据。

```
如：
接口请求数据为：{"username":"13800000001", "password":"123456", "create_time":"1519516631"}
那么md5({"username":"13800000001", "password":"123456", "create_time":"1519516631"}) => 得到a465b652eb7707ffaf7c0da21f7f8a6a
然后重新构造请求：
{"username":"13800000001", "password":"123456", "create_time":"1519516631", "sign":"得到a465b652eb7707ffaf7c0da21f7f8a6a"}

服务器拿到这组数据后
取出{"username":"13800000001", "password":"123456", "create_time":"1519516631"}
然后也进行md5({"username":"13800000001", "password":"123456", "create_time":"1519516631"})
得出=> x
查看x是否与 sing中的值一模一样。
如果一样证明接口信息没有被篡改
```

应用：保证数据的一致性、完整性

签名后的接口测试方法：

- 开发提供签名算法，我们使用工具也构造出同样要求的请求
- 开发提供签名的工具或插件，我们使用它们，结合接口测试工具，构造出同样要求的请求
- 开发关闭签名，不校验sign（这种属于不全面的测试，没有校验的sign，需要开启sign之后，放在手工环节进行）

5 接口文档介绍

接口文档，又称为API文档 (API：Application Program Interface)

描述接口的相关信息，包括接口的请求数据，也就是入参，和响应数据，也就是出参

作用：

- 让前端工程师和后端工程师沟通协作提高工作效率
- 让项目版本更迭/人力更迭时，可以通过文档交接工作，提高效率
- 让测试人员进行测试时，有一个可以参考的文档，从而更好的进行测试

5.1 接口文档组成部分

一个标准的接口文档，无论是它是什么形式，一定包括以下内容：

基本信息：

- 请求方法

- 资源路径
- 描述
- 接口的名字

请求内容：

- 请求头
  - 请求头包括字段名、字段值、是否必须、示例、备注
- 查询参数

  URL中的查询参数，包括字段名，字段值类型，是否必须，示例、备注
- 请求体

  包括字段名，字段值类型，是否必须，示例、备注

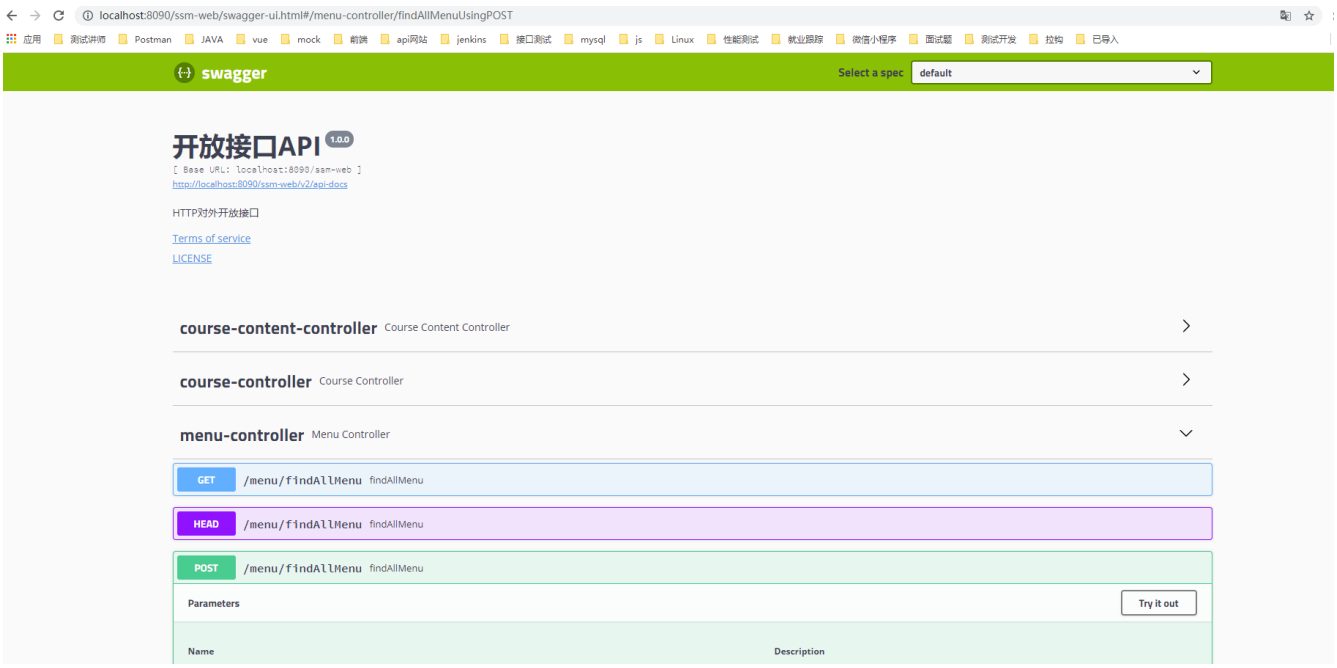
响应内容：

- 响应数据（参数名、参数类型、是否必填、示例、备注）
- 响应数据中，各种状态码含义的描述

5.2 接口文档的分类

- Web形式接口文档

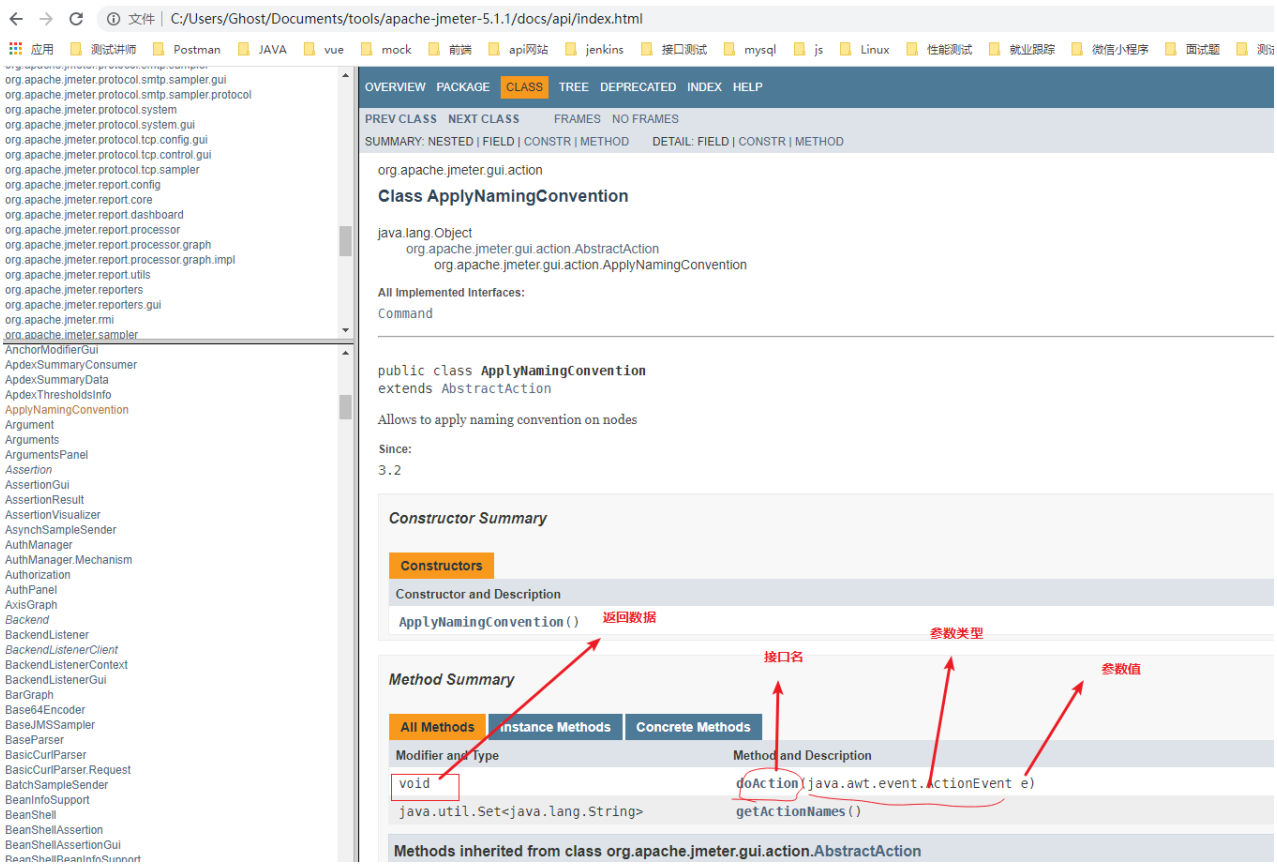
  swagger工具，就能够生成web浏览器形式的接口文档



- PDF、Word、Excel形式接口文档


拉勾教育接口文档.md - Typora
文件(F) 编辑(E) 段落(P) 格式(O) 视图(V) 主题(T) 帮助(H)
文件
大纲
查找
拉勾教育接口文档
用户管理
查询用户列表
搜索用户
查询用户权限
登陆
冻结/解冻用户
修改用户角色权限
课程管理
查询所有课程
新增课程
修改课程
更新课程状态
查询课程章节
新增课程章节
修改课程章节
更新课程章节状态
权限管理
角色列表
查询角色
添加角色
修改角色
删除角色
拉勾教育接口文档
拉勾教育域名：www.edu2.com
端口：8090
用户管理
查询用户列表
基本信息
请求方法：POST
资源路径：/user/findAllUserByPage
接口描述：根据分页查询多个用户组成的列表信息
请求参数
请求头
参数名 参数类型 是否必填 示例 备注
Authorization TOKEN 是 用户令牌
Content-Type application/json 是

- 代码形式接口文档


文件 | C:/Users/Ghost/Documents/tools/apache-jmeter-5.1.1/docs/api/index.html
OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP
org.apache.jmeter.gui.action
Class ApplyNamingConvention
java.lang.Object
org.apache.jmeter.gui.action.AbstractAction
org.apache.jmeter.gui.action.ApplyNamingConvention
All Implemented Interfaces:
Command
public class ApplyNamingConvention
extends AbstractAction
Allows to apply naming convention on nodes
Since:
3.2
Constructor Summary
Constructors
Constructor and Description
ApplyNamingConvention()
返回数据
参数类型
接口名
参数值
Method Summary
All Methods Instance Methods Concrete Methods
Modifier and Type
Method and Description
void
doAction(java.awt.event.ActionEvent e)
java.util.Set<java.lang.String>
getActionNames()
Methods inherited from class org.apache.jmeter.gui.action.AbstractAction

5.3 接口规范

5.3.1 接口规范示例

5.3.2 传统风格的接口

订单管理	请求方法	URL	响应状态码
查询订单列表	GET/POST	http://www.xxx.com/api/v1/order_list http://www.xxx.com/api/v1/queryAllOrder	200
查询订单详情	GET/POST	http://www.xxx.com/api/v1/queryOrderDetail?order_id=1 http://www.xxx.com/api/v1/queryOrderByID	200
创建订单	POST	http://www.xxx.com/api/v1/CreateOrder	200
修改订单	POST	http://www.xxx.com/api/v1/modifyOrderById	200
删除订单	POST	http://www.xxx.com/api/v1/deleteOrderById	200

特点

- URL没有固定的写法，一个公司中，操作同一组数据时，可能会出现多个URL的情况。

  > 查询订单详情和修改订单详情的URL不一样

- 响应状态码统一都是200
- 请求方法：GET、POST完成接口请求

传统风格的出现是因为早期HTTP协议只支持GET和POST

5.3.3 Restful风格的接口

如果一个架构设计满足REST设计原则，就称之为RESTful。

一个满足RESTful的接口设计，必须遵循REST的很多标准，其中主要包括5点：

- 每一个URL都是一个资源
- 客户端只能操作资源的表现层。表现层：是指数据的表现形式(json,html,xml)

  客户端通过HTTP动词（GET、POST、PUT、DELETE等），操作资源，实现表现层状态转化
- 客户端通过Get查询资源

  通过POST新增资源

  通过PUT修改资源

  通过DELETE删除资源

  操作完成之后，资源中对应的数据会发生改变，所以叫做状态转化。

  无状态
- 不会保存请求数据

- 自描述性

订单管理	请求方法	URL	响应状态码
查询订单列表	GET	http://www.xxx.com/api/v1/order	200
查询订单详情	GET	http://www.xxx.com/api/v1/order/1	200
创建订单	POST	http://www.xxx.com/api/v1/order	201
修改订单	PUT	http://www.xxx.com/api/v1/order/1	201
删除订单	DELETE	http://www.xxx.com/api/v1/order/1	204

特点

- 在项目中对于同一个资源的操作，URL都是一样的。
- 响应状态码
  - GET 200
  - POST 201
  - PUT 201

    - DELETE 204
- 请求方法：
    - 查询： GET
    - 新增： POST
    - 修改： PUT
    - 删除： DELETE

REST自描述性的体现：

- 通过OPTIONS 请求方法， 能获取到操作的URL资源的所有请求方法和子资源，以及每个资源的调用方法

6 接口测试的工作流程

6.1 理想情况的接口测试

- 需求分析
- 开发提供接口文档
    - 文档形式
    - Web形式
    - 代码形式
- 分析接口文档
- 编写接口测试用例
- 执行接口测试用例
    - Postman执行
    - Jmeter执行
    - Python、Java执行
    - Swagger也可以简单调试执行
    - Fiddler、Charles也可以执行（不好用）
- 输出报告
- 进行自动化接口测试和持续集成

6.2 实际工作的接口测试

接口文档来源：

- 开发不提供接口文档，没有界面能够使用
- 开发不提供接口文档，有UI界面能够使用
- 开发提供的接口文档不全
- 开发提供的是一个Web形式的接口文档

- 开发直接贴的代码形式

接口文档解析：

- 看不懂接口文档

解决办法：

- 开发不提供接口文档，没有界面能够使用
    - 从竞品分析，模糊分析
    - 与开发沟通
- 没有接口文档，有UI界面能够使用：
    - 使用抓包工具，抓取系统使用过程的接口，来进行接口测试；功能测试更深入时，也会这么进行功能测试
    - 与开发沟通
- 开发提供的接口文档不全
    - 抓包工具，进一步获取信息
    - 需要和开发沟通
- 开发直接贴的代码形式
    - 与开发沟通，以为了达成"接口测试"的目标，协商解决
- 看不懂接口文档
    - 与开发沟通，搞清楚每一个参数的来源、作用

7 接口用例设计

设计接口用例的目的是为了提升工作效率，防止遗漏

并且能够通过用例，来实施测试监控。

7.1 接口用例设计思路

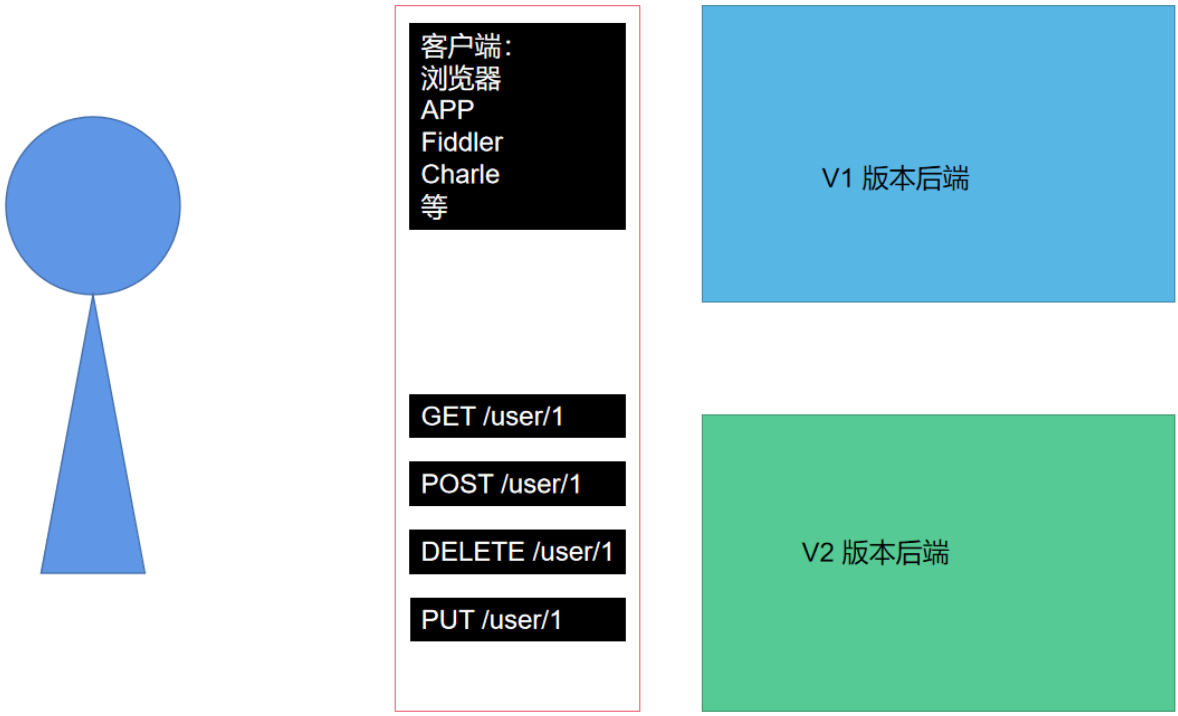
7.1.1 设计用例时的四大维度

- 功能
    - 单个接口测试
    - 接口业务场景测试（多接口组合测试）
- 性能
    - 响应时间
    - 吞吐量
    - 服务器资源使用率（CPU、内存、磁盘、网络、IO）
    - 错误率
    - 并发数

      PS:幂等测试
- 安全

    - 敏感信息是否泄露
    - XSS注入
    - SQL注入
    - 其他
- 兼容性

  接口的兼容性，主要是测试不同接口版本的兼容。



7.2 接口功能测试用例设计方法

简单的说，接口测试用例设计，也可以使用黑盒测试用例设计方法来设计。

例如：采用等价类、边界值来对接口中的请求数据中的每个参数设计测试用例；也需要对响应的数据进行用例的设计。

然后根据需求规定的业务场景，使用场景法来设计业务场景测试用例。

第一步：分析接口文档中，接口的入参和出参，明确每个参数类型和作用

- **参数类型：** 字符串、整型数字、小数、日期、列表对象
    - 字符串：测试数字、字母、特殊字符、字符串长度、为空、为null、重复字符串
    - 整型数字：测试数字大小、位数、小数、负数、0、为空、为null、重复数字，精度，科学计算法
    - 小数：测试小数大小、位数、整数、负数、0、为空、为null、重复数字，精度
    - 日期：测试闰年润月、大小月、时分秒
    - 列表对象：重点关注有没有“列表”的功能；对于请求数据，通过构造单个列表数据、多个列表数据进行测试

```
post http://xxx.xxx/api/v1/user
content-type:application/json

{"tag":"add", "user_info_list":[{"mobile":"xx", "password":"x"}, {"mobile":"",
"password":"x"}]}
```

优先覆盖列表数据都正确的场景，次级覆盖列表数据既有正确数据，又有错误数据的场景

- **参数作用**：结合接口文档、需求文档分析

  每个参数的作用不是固定的，必须结合接口的作用，结合上下文场景分析。

  例如：登陆接口

  ```
  post /api/v1/login
  content-type:application/json

  {"mobile":"", "password":"", "verify_code":""}

  # 其中，可以思考到登陆接口中的mobile、password、verify_code用来传递用户名、密码和验证码
  ```

  有些参数我们也不知道做什么用，只能通过接口文档查看，如果接口文档没有写，那么最终还是需要和开发沟通。

  ```
  post /api/v1/login
  content-type:application/json

  {"mobile":"", "password":"", "verify_code":"", "sign":"xxx", "channel":"1"}

  channel的作用是未知的。
  ```

  接口测试中，原则上来讲，我们需要覆盖接口中的所有参数，但是并不是所有参数都是需要测试的。

  有的接口参数特别多，但是在当前业务中，并没有使用全部的参数，这个时候，我们可以重点关注当前业务所使用的参数。

  例如：订单列表的接口参数解析。

responseCode				String	是	响应代号
responseMessage				String	是	响应信息
responseData	dataSize			Int	是	总记录条数
	totalPage			Int	是	总页数
	currPageNum			Long	是	当前页
	currPageSize			Long	是	当前页大小
	mainOrderList			List	是	订单集合
		mainOrderNo			是	主订单号
		mainorderCreateTime			是	主订单生成时间
		multiAccountCode			是	组合号
		multiType			是	业务类型
		subOrderList		List	是	子订单集合
			orderNo	String	是	订单编号
			productId	String	是	产品Id
			productCode	String	是	产品代码
			productName	String	是	商品名称
			productSerial	String	是	商品系列
			lv2CatogoryCode	String	是	商品二级分类
			productCatogoryCode	String	是	商品类别（最小分类）
			product_icon_url	String	否	列表图片
			sku_key_list	String	否	订单商品SKU
			orderType	String	否	订单类型
			fundShare	String	否	基金份额
			orderAmount	BigDecimal	是	订单金额
			orderStatus	String	是	订单状态
			orderCreateTime	Timestamp	是	订单生成时间
			renew	String	否	续期交易指令
			shareRegisterDate	String	否	份额注册日期
			allowRedemptDate	String	否	可赎回日期
			withdrawStatus	String	否	最新赎回状态
			possessionAsset	String	否	持仓总金额
			totalIncome	String	否	累计收益
			remark	String	否	备注
			fundCode	String	否	基金代码
			bankName	String	否	银行名称
			bankCardNo	String	否	银行卡号
			payCompletedTime	Timestamp	否	支付完成时间
			partnerOrderId	String	否	共联方订单ID

如果该订单不是基金，
那么这些字段预期应该不发挥作用

如果发挥了作用，就是设计的上的bug

第二步：分析接口的使用场景

在接口测试中，单个接口参数，已经涵盖了测试场景，所以我们需要分析出每个参数的可能使用场景来进行用例的设计。

例如：

```
支付接口中：
post /api/v1/pay
content-type:application/json
toekn:xxxx

{"price":"1", "create_time":"", "order_status":"1", "goods_name":"华为mate20",
"goods_id":"1"}

# price是价格，那么价格的使用场景有哪些？
# create_time，是创建时间，根据创建时间，使用场景有哪些？
# order_status，是订单状态，根据订单状态，使用场景有哪些？
# goods_name,商品名称，使用场景有哪些？
# goods_id，商品id，使用场景有哪些？

如果没有做过相关业务功能测试，也没有做过接口测试，都很难知道这些参数的使用场景。
所以需要结合接口文档、需求文档，并和设计该接口的开发、产品经理讨论之后，才能确定。
```

```
如：结合之前积累的经验，可以知道：price就是支付之后，要扣除的金额；
create_time可以用来验证订单是否失效
order_status可以控制订单状态，只有未支付的订单，才能进行支付；已支付的订单无法重复支付
goods_name用来描述支付的商品
goods_id 具有唯一性，防止支付的商品错误
```

在进行接口的业务功能测试后，我们还需要考虑接口的参数测试（优先级低）

- 必填参数
- 组合参数
- 全部参数
- 多参
- 少参
- 无参
- 错误参数

总结：

第一步：熟悉接口的参数（入参和出参）

第二步：针对参数使用等价类、边界值设计测试用例

- 所有的入参，针对每个参数使用等价类、边界值等黑盒测试用例设计方法进行测试
- 所有的出参，针对重要参数，使用等价类、边界值等黑盒测试用例设计方法进行测试

第三步：分析参数的使用场景，设计该接口每个参数的使用场景

第四步：接口参数测试

7.3 单接口用例设计

案例：拉勾教育后台登陆

编号	标题	模块	接口名称	请求方法	URL	请求头	请求数据	请求数据类型	预期数据	实际数据
1	登陆成功	拉勾教育	登陆	POST	/ssm_web/user/login		查询参数: phone=15321919666 password=123456	x-www-form-urlencoded	{ "success": true, "state": 1, "message": "响应成功",	

2	密码错误	拉勾教育	登陆	POST	/ssm_web/user/login		查询参数:
							phone=15321919666
							password=error
								x-www-form-urlencoded	{
									"success": true,
									"state": 206,
									"message": "用户名密码错误",
									"content": null
									}	

3	账号不存在	拉勾教育	登陆	POST	/ssm_web/user/login		查询参数:
							phone=15121919666
							password=123456
								x-www-form-urlencoded	{
									"success": true,
									"state": 206,
									"message": "用户名密码错误",
									"content": null
									}	

4	账号为空	拉勾教育	登陆	POST	/ssm_web/user/login		查询参数:
							phone=
							password=123456
								x-www-form-urlencoded	{
									"success": true,
									"state": 206,
									"message": "用户名密码错误",
									"content": null
									}	

5	密码为空	拉勾教育	登陆	POST	/ssm_web/user/login		查询参数: phone=15121919666 password=	x-www-form-urlencoded	{ "success": true, "state": 206, "message": "用户名密码错误",	

第一步：熟悉接口入参和出参

第二步：针对每一个参数设计测试用例

第三步：针对业务功能设计测试用例

第四步：设计接口的参数测试用例

熟悉接口的设计思路之后，我们实际编写测试用例时，为了提升工作效率会先写xind的测试用例

再根据实际情况是否编写excel的详细有步骤地测试用例。

- 实际情况：如果这个测试用例，需要发给其他不懂的同事来使用，那么就需要编写EXCEL的测试用例。

7.4 业务场景接口用例设计

业务场景：用户实际使用业务的流程场景，叫做业务场景。

案例：拉勾教育登陆和课程管理

编号	标题	模块	接口名称	请求方法	URL	请求头	请求数据
1	测试登陆后管理课程	拉勾教育	登陆	POST	/ssm_web/user/login		查询参数: phone=15321919666 password=123456
			查询课程列表	POST	/ssm_web/course/findAllCourse	Cookie:{"JSESSIONID":"FA36D8F677B83200C94D8B46495E697A"}, Content-Type:application/json	{}
			添加课程	POST	/ssm_web/course/saveOrUpdateCourse	Cookie:{"JSESSIONID":"FA36D8F677B83200C94D8B46495E697A"}, Content-Type:application/json	{"brief":"简介 ","courseDescriptionMarkDown "","courseName":"测试添加课程" ","id":"","previewFirstField":"这是 ","previewSecondField":"这是— ","price":"1","sales":"1","sortNur ","position":"讲师","description" ","teacherDTO":{},"activityCours
			修改课程	POST	/ssm_web/course/saveOrUpdateCourse	Cookie:{"JSESSIONID":"FA36D8F677B83200C94D8B46495E697A"}, Content-	{"brief":"简介 ","courseDescriptionMarkDown "","courseName":"测试添加课程 坦","id":33,"previewFirstField":" ","previewSecondField":"这是— ","price":1,"sales":1,"sortNum":0

进行接口的业务场景测试时：必须只测试正常的功能

登陆必须登陆成功

查询课程列表成功

添加课程成功

修改课程成功

删除课程成功

优先级高