HTTP请求的常用方法

GET

请求URI所对应的资源

POST

传输内容实体

PUT

通过内容实体部分传输文件内容，将文件上传到指定的位置

DELETE

和PUT相反，删除指定位置的文件

PATCH

默认是以x-www-form-urlencoded的contentType来发送信息，并且信息内容是放在request的body里。

HEAD

HEAD方法和GET方法一致，不返回报文的内容实体
用于确定URI资源的有效性以及资源更新的日期时间

OPTIONS

用来查询针对URI资源所支持的方法

TRACE

追踪路径

CONNECT

要求用隧道协议连接代理，将内容加密后进行传输

数据传输方式:Content-Type

```
Content-Type: application/json
{"name":"test0107","password":"123456"}

Content-Type: application/x-www-form-urlencoded
name=test0107&password=123456
```

面试题： HTTP协议当中GET方法和POST方法的区别？

1.get方法一般是获取资源,post方法为了传输内容实体

2.get方法的参数是直接拼接到url上进行传输,post的参数主要通过内容实体传输

3.getf方法的参数只能是urlencode方式,post可以是其他Content-Type方式

4.post传递参数方式相对于get方式更加安全,因为get的参数会显示到url上

5.get参数传递的个数可能会因为浏览器的不同而有限制,post没有

6.对于资源来说.get是安全的,post不安全

面试题： HTTP协议当中POST方法和PUT方法的区别

1.post是传输内容实体,put用于传输文件

HTTP响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Frame-Options: DENY
Content-Length: 158
Vary: Cookie, Origin
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: http://101.91.150.147:8008
Set-Cookie: sessionid=80ztl2ghhyxk0uakjewf5u9d037xh7om; expires=Mon, 13 Mar 2023
12:13:17 GMT; HttpOnly; Max-Age=1209600; Path=/; SameSite=Lax

{"code": "200", "msg": "Success Create", "data": {"name": "test0107", "openid":
"290fc49f49dbe28661bb27f6ed785d31", "user_id": 251921}, "ip": "113.91.43.228"}
```

报文格式

1. 响应行

   HTTP/1.1 200 OK 协议版本： HTTP/1.1
   状态码： 200
   描述信息：OK

2. 响应头域

   Content-Type: application/json#响应数据类型 X-Frame-Options: DENY#表示该页面不能再iframe中展示 Content-Length: 158#响应内容的长度 Vary: Cookie, Origin#返回的内容添加了服务器的头部信息和Cookie X-Content-Type-Options: nosniff#有助于防御MIME型攻击 Referrer-Policy: same-origin#对应同源请求会发送引用地址 Access-Control-Allow-Credentials: true#表示是否允许发送Cookie Access-Control-Allow-Origin: http://101.91.150.147:8008/#指定服务器可以跨域源 Set-Cookie: sessionid=80ztl2ghhyxk0uakjewf5u9d037xh7om; expires=Mon, 13 Mar 2023 12:13:17 GMT; HttpOnly; Max-Age=1209600; Path=/; SameSite=Lax #设置Cookie

3. 空行

4. 内容实体

   {"code": "200", "msg": "Success Create", "data": {"name": "test0107", "openid": "290fc49f49dbe28661bb27f6ed785d31", "user_id": 251921}, "ip": "113.91.43.228"}

HTTP响应状态码

- 1XX: 提示信息，服务器端已经收到了请求，但是还需要进一步的处理
- 2XX: 操作成功
- 3XX: 重定向
- 4XX: 客户端错误
- 5XX: 服务器端错误

100 Continue

200 OK

操作处理成功

204 No Content

请求处理成功，但是没有内容返回

206 Partial Content

客户端进行了范围请求，服务器按照客户端的要求返回了部分内容

301 Moned Permanently

永久重定向

302 Found

临时重定向

303 Other

临时重定向,明确表示客户端应该采用GET方法来获取资源

304 Not Modified

根据客户端请求的条件，资源并没有发生改变

400 Bad Request

HTTP请求的语法错误

403 Forbidden

对于资源的被服务器拒绝，服务器没有必要给出拒绝的理由

404 Not Found

无法找到请求的资源

500 Internal Server Error

服务器端错误

503 Service Unavailable

服务器临时错误

HTTP协议的特点

无连接

一次HTTP的请求和响应完成之后，会关闭掉TCP的连接
由于现在的网页的内容越来越丰富，浏览一个页面，不再是简单的发起一个HTTP请求，完成业务操作需要频繁的发送HTTP请求。这就要求TCP的连接可以复用。
通过： Connection: keep-alive (长链接) 来解决该问题

无状态

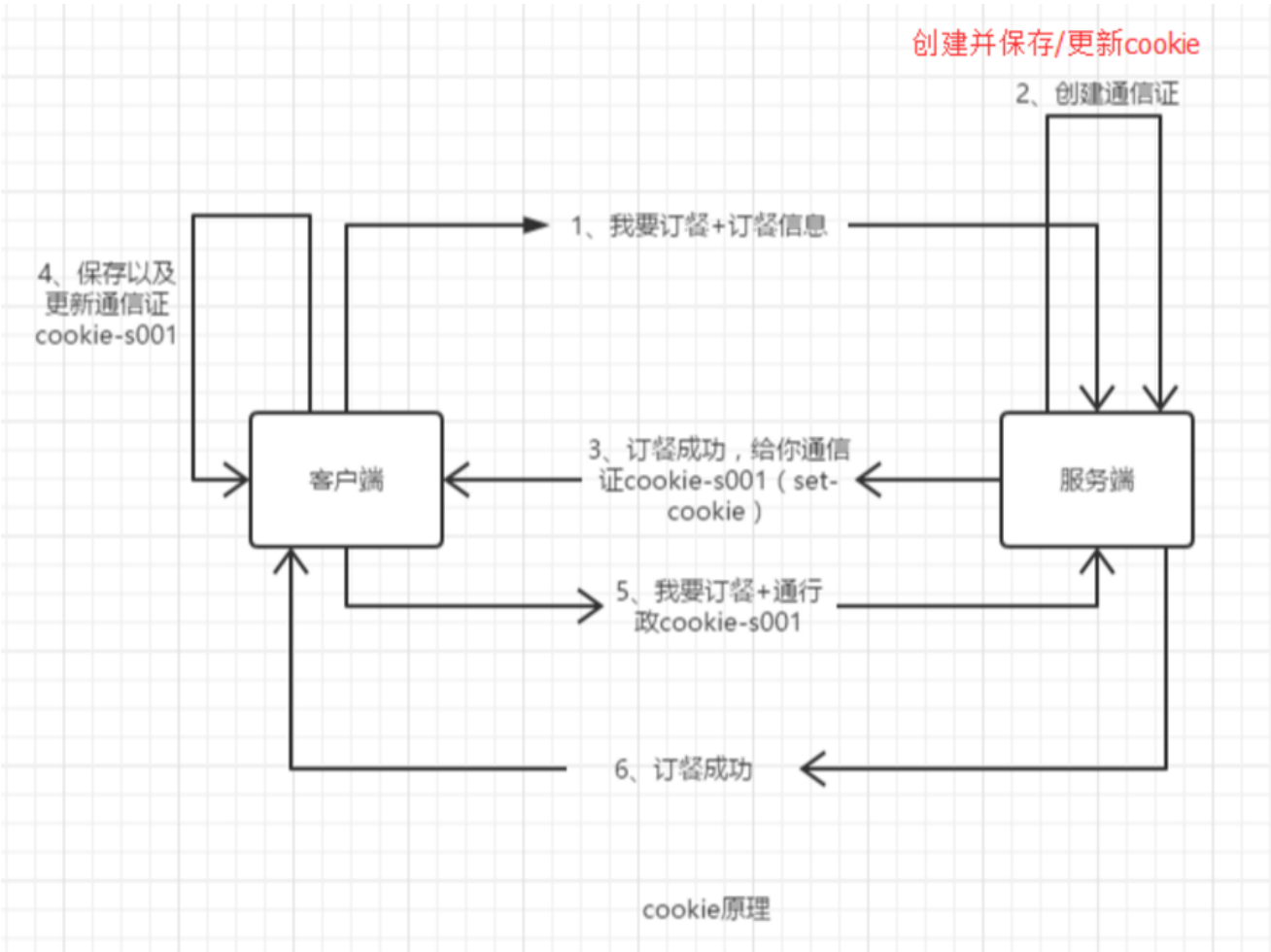
HTTP协议对交互的场景没有记忆能力

解决HTTP无状态的问题

cookie

是一段文本内容

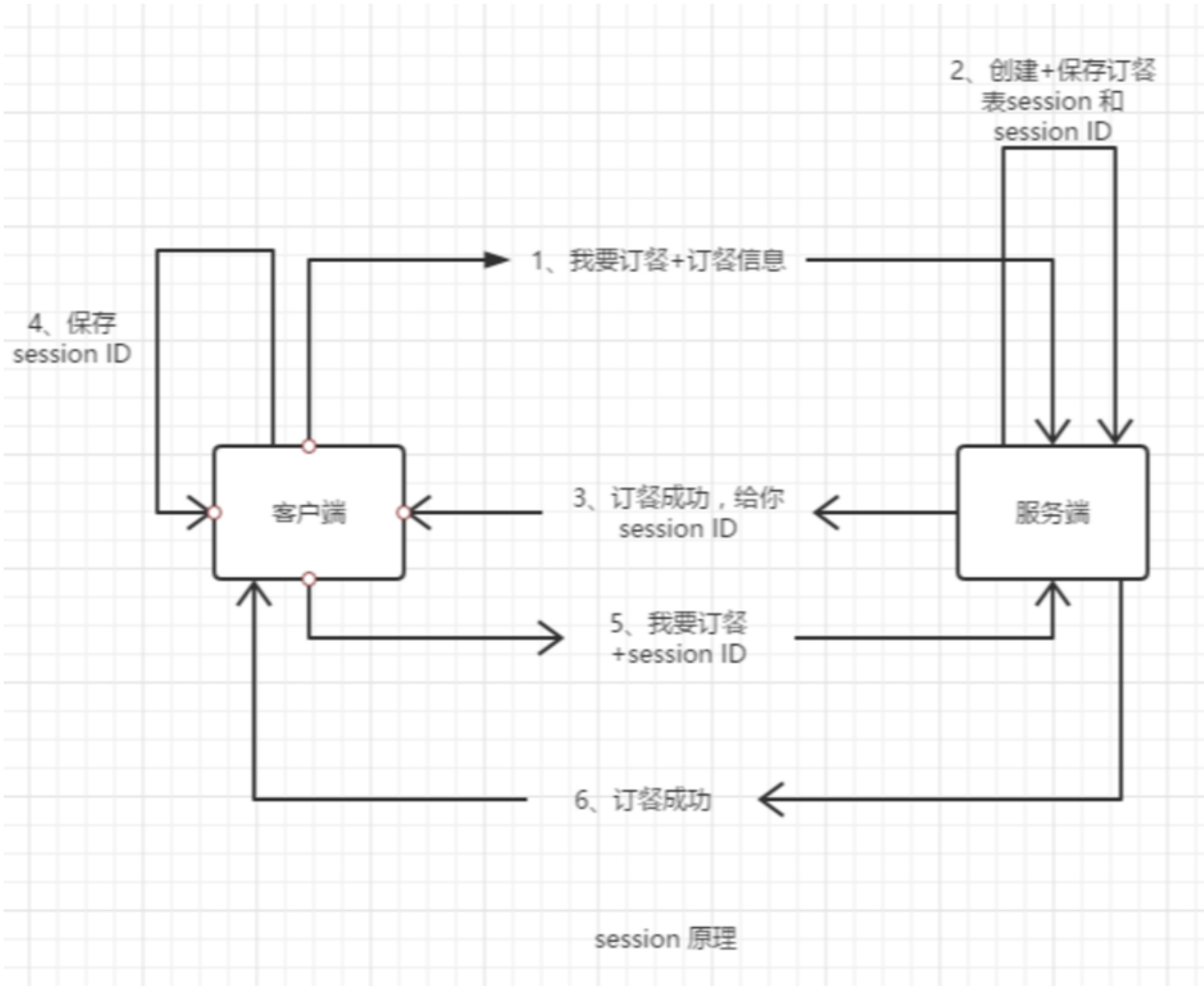
1. 服务器生成cookies信息，在HTTP响应报文的头域当中通过Set-Cookie头域来告知客户端应当保存cookie内容
2. 浏览器接收响应，将cookie内容在本地保存
3. 浏览器针对相同域名发起HTTP请求，会附带上保存的cookies，通过请求报文的cookies头域



cookie原理

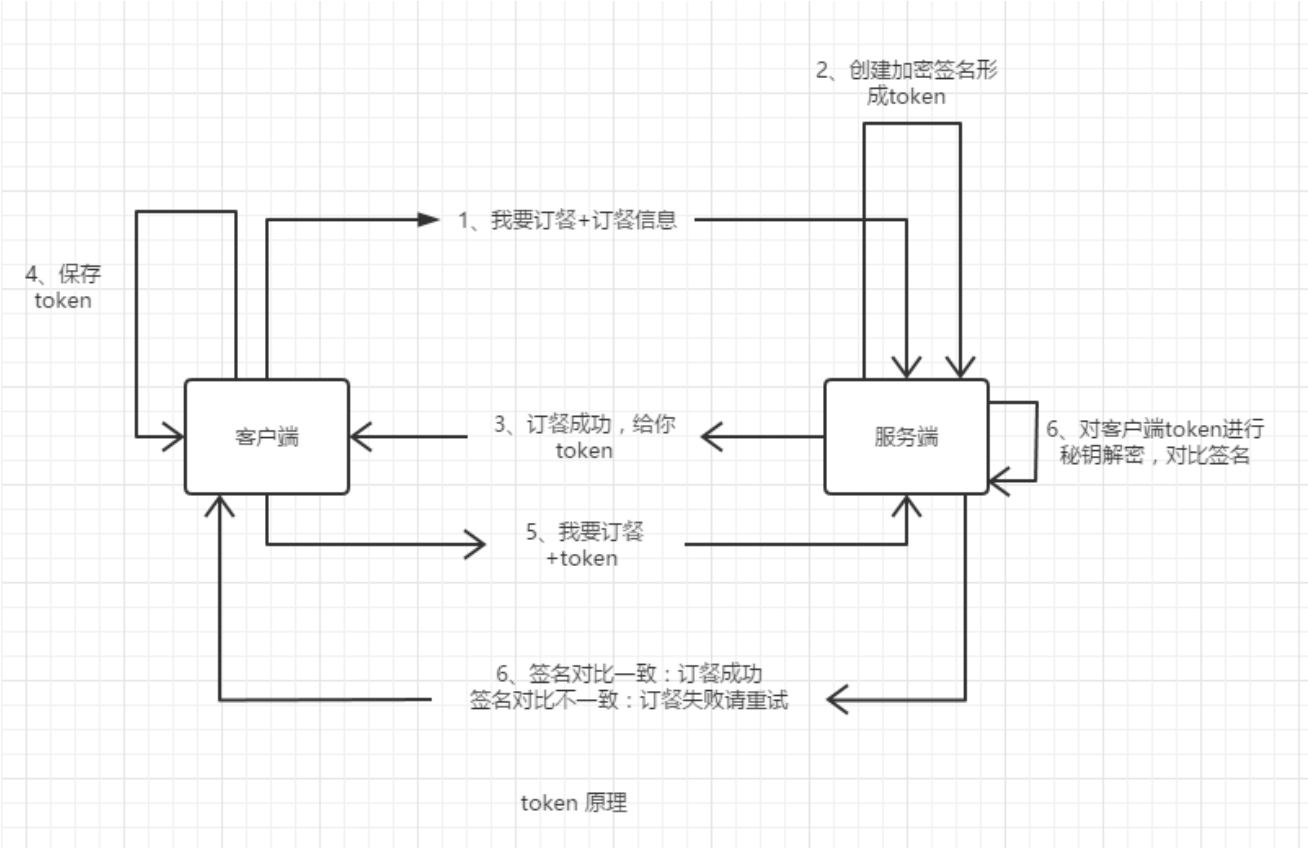
session

session是一种记录客户状态的机制，不同的是，cookie是保存在客户端的浏览器中，session是保存在服务器上。当浏览器访问服务器的时候，服务器把用户信息记录以某种形式记录在服务器上，这就是seesion



token

标识用户身份的一串字符串，服务器加密生成token,保存在客户端
客户端的请求带上token,服务器对token解密对比即可验证身份



面试题： cookie和session的区别

1.cookie是保存在客户端,session是保存到服务端

2. cookie保存在客户端,相对于session就不是那么安全
3. cookie的数量和数据有限制,数量一般不超过20个,数据不能超过4K
4. cookie是HTTP协议中的规范,session是一种机制.通常会使用cookie来存储session_id

加密算法

1. 对称加密

加密和解密采用相同的秘钥
优点： 加密速度快
缺点： 秘钥的传递和保存是一个问题，参与加密和解密的双方使用的秘钥一样，这样秘钥就很容易泄露

2. 非对称加密

加密和解密采用不同的秘钥(公钥和私钥)

优点： 加密和解密的秘钥不一样，公钥是可以公开的，只保证私钥不被泄露即可，这样秘钥的传递就变得简单很多，从而降低了被破解的几率

缺点： 加密速度慢

3. 线性散列算法

单向的不可逆的(加密之后不能够被解密) 比如，密码存储,md5加密

```
注册:
1.从终端输入用户名,密码,电话号码
2.把以上信息存储到数据库,并且对密码进行md5加密
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
import hashlib
import pymysql
# #把字符串转换为ascii码
# mystr="zb123"
# n=mystr[0]
# print(ord(n))


#获取用户信息方法
def getUser():
    while True:
        print("用户名的长度必须是6-18个字符,字母,数字,下划线等其他字符组成,字母开头")
        name=input("请输入用户名:")
        if len(name)>=6 and len(name)<=18:
            pass
        else:
            print("你输入的字符个数不够或超出了字符个数限制")
            continue

        if (ord(name[0])>=65 and ord(name[0])<=90) or (ord(name[0])>=97 and
ord(name[0])<=122):
            pass
        else:
            print("你输入的首字符不是字母")
            continue

        print("密码长度必须是6-18个字符,字母,数字,下划线等其他字符组成")
        passwd=input("请输入密码:")
        if len(passwd)>=6 and len(passwd)<=18:
            pass
        else:
            print("你输入的字符个数不够或超出了字符个数限制")
            continue

        print("只能是中国内陆的手机号码")
        iph=input("请输入你的手机号码:")
        #首先要11位,要是纯数字,首字符必须是1
        if len(iph)==11 and iph.isdigit() and iph[0]=="1":
            pass
        else:
            print("输入的手机号码不符合规定")
            continue

```

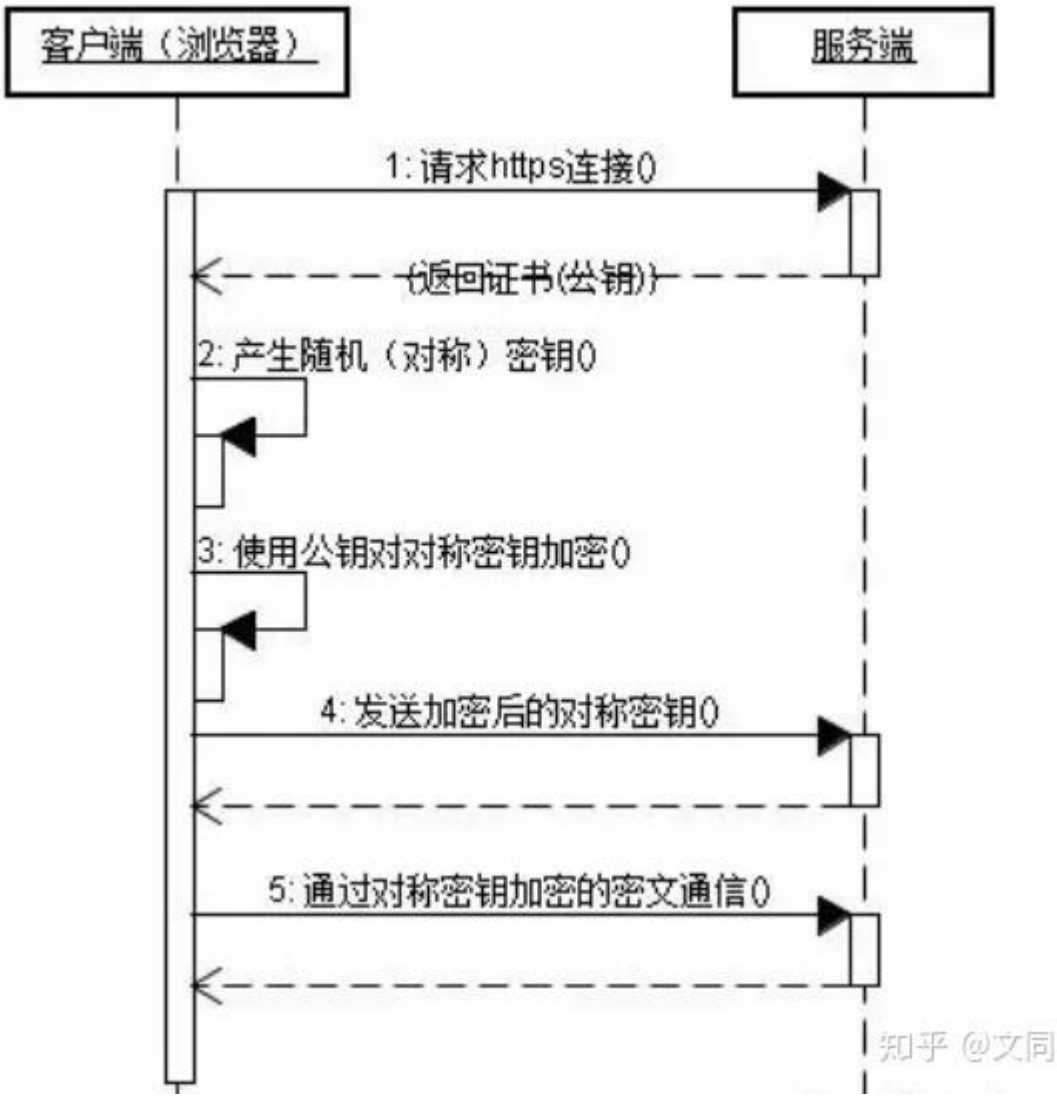
```
        print("输入的信息都符合")
        break

    return name,passwd,iph

#加密方法
def myMD5(mystr):
    md5str=hashlib.md5(mystr.encode(encoding='utf=8')).hexdigest()
    return md5str

#存储信息到数据库
def mysaveMysql(msg):

db=pymysql.connect(host="127.0.0.1",port=3306,user="root",passwd="123456",db="myte
st202305")
    cur=db.cursor()
    sql="insert into user values('%s','%s','%s')"%(msg[0],msg[1],msg[2])
    cur.execute(sql)
    db.commit()
    db.close()



def mytest():
    #获取用户输入的信息
    msg=getUser()

    #对密码进行加密
    MD5passwd=myMD5(msg[1])
    mylist=[msg[0],MD5passwd,msg[2]]

    #把信息存储到数据库中
    mysaveMysql(mylist)

mytest()

登录:
1.从终端输入用户名和密码
2.从数据库获取用户名和密码,然后和用户输入的用户名和密码进行对比
3.如果正确那么显示"登录成功,并打印用户名,加密后的密码,电话号码",如果失败,显示"用户名或密码错误"

#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
import pymysql
import hashlib
#通过name来查询数据库中的数据
def myselect(name):
```

```
    db = pymysql.connect(host="127.0.0.1", port=3306, user="root", passwd="123456", db="mytest202305")
    cur = db.cursor()
    sql="select *from user where name='%s'"%name
    cur.execute(sql)
    data = cur.fetchall()
    # print(data[0][1])
    return data



#加密方法
def myMD5(mystr):
    md5str=hashlib.md5(mystr.encode(encoding='utf=8')).hexdigest()
    return md5str

def getUser():
    # 从终端获取用户的信息
    name = input("请输入用户名:")
    if len(name) >= 6 and len(name) <= 18:
        pass
    else:
        print("你输入的字符个数不够或超出了字符个数限制")

    if (ord(name[0]) >= 65 and ord(name[0]) <= 90) or (ord(name[0]) >= 97 and ord(name[0]) <= 122):
        pass
    else:
        print("你输入的首字符不是字母")

    passwd = input("请输入密码:")
    if len(passwd) >= 6 and len(passwd) <= 18:
        pass
    else:
        print("你输入的字符个数不够或超出了字符个数限制")
    return name,passwd

def mytest():
    while True:
        #从终端获取用户信息
        name,passwd=getUser()

        #根据从终端获取的用户信息来从数据库获取用户的信息
        data=myselect(name)
        if len(data)==0:
            print("你没有注册,请注册")
            continue

        #比对
        #用户输入的密码(加密后的)
        inputpasswd=myMD5(passwd)
        #从数据库获取的密码
        mysqlpasswd=data[0][1]
```

```
        if inputpasswd==mysqlpasswd:
            print("登录成功")
            break
        else:
            print("你输入的密码不对,请重新输入")
            continue


mytest()
```

HTTPS

1. 首先客户端通过URL访问服务器建立SSL连接。
2. 服务端收到客户端请求后，会将网站支持的证书信息（证书中包含公钥）传送一份给客户端。
3. 客户端的服务器开始协商SSL连接的安全等级，也就是信息加密的等级。
4. 客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。
5. 服务器利用自己的私钥解密出会话密钥。
6. 服务器利用会话密钥加密与客户端之间的通信。



面试题： HTTP和HTTPS协议的区别

1.https是http的安全版本,http的明文的数据方式传输,HTTPS是用了SSL/TLS协议进行了加密传输

2. http的默认端口号是80,https的默认端口号是443
3. https需要证书,http不需要