06 Navicat工具、pymysql模块

Navicat**工具、**pymysql**模块**

阅读目录

一 IDE工具介绍(Navicat)

生产环境还是推荐使用mysql命令行，但为了方便我们测试，可以使用IDE工具，我们使用Navicat工具，这个工具本质上就是一个socket客户端，可视化的连接mysql服务端的一个工具，并且他是图形界面版的。我们使用它和直接使用命令行的区别就类似linux和windows系统操作起来的一个区别。

下载链接：https://pan.baidu.com/s/1bpo5mqj

Navicat的安装教程看这篇博客：https://www.cnblogs.com/clschao/articles/10022040.html

```
掌握：
#1. 测试+链接数据库
#2. 新建库
#3. 新建表，新增字段+类型+约束
#4. 设计表：外键
#5. 新建查询
#6. 备份库/表

#注意：
批量加注释：ctrl+？键
批量去注释：ctrl+shift+？键
```

二 pymysql模块

我们要学的pymysql就是用来在python程序中如何操作mysql，它和mysql自带的那个客户端还有navicat是一样的，本质上就是一个套接字客户端，只不过这个套接字客户端是在python程序中用的，既然是客户端套接字，应该怎么用，是不是要连接服务端，并且和服务端进行通信啊，让我们来学习一下pymysql这个模块

```
#安装
pip3 install pymysql
```

一 链接、执行sql**、关闭（游标）**

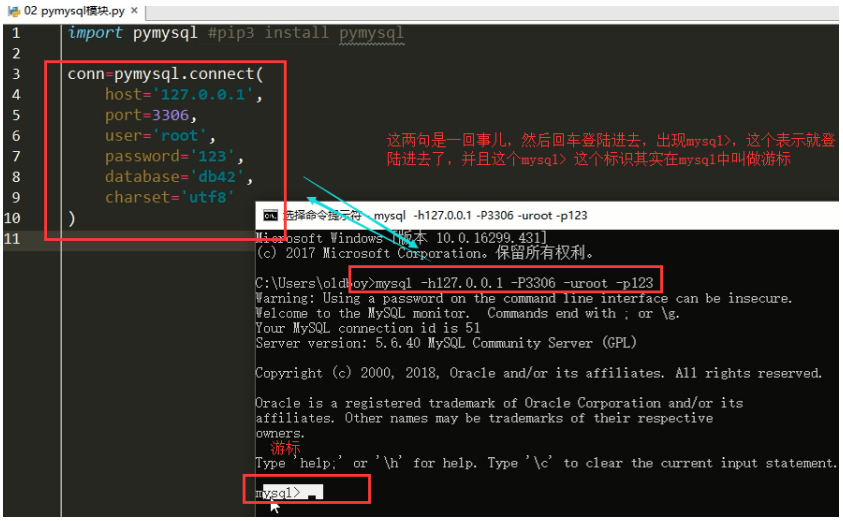

```
import pymysql
user=input('用户名: ').strip()
pwd=input('密码: ').strip()

#链接，指定ip地址和端口，本机上测试时ip地址可以写localhost或者自己的ip地址或者127.0.0.1，然后你操作数据库的时候的用户名
conn=pymysql.connect(host='localhost',port=3306,user='root',password='123',database='student',charset='utf8') #指定编
#游标
cursor=conn.cursor() #这就想到于mysql自带的那个客户端的游标mysql> 在这后面输入指令，回车执行
#cursor=conn.cursor(cursor=pymysql.cursors.DictCursor) #获取字典数据类型表示的结果：{'sid': 1, 'gender': '男', 'class_id': 1



#然后给游标输入sql语句并执行sql语句execute
sql='select * from userinfo where name="%s" and password="%s"' %(user,pwd) #注意%s需要加引号，执行这句sql的前提是医
print(sql)
res=cursor.execute(sql) #执行sql语句，返回sql查询成功的记录数目，是个数字，是受sql语句影响到的记录行数，其实除了受影响的
```

```
fetchone：(1, '男', 1, '理解')
fetchone：(2, '女', 1, '钢蛋')
fetchall：((3, '男', 1, '张三'), (4, '男', 1, '张一'))
```

```
#上面fetch的结果都是元祖格式的，没法看出哪个数据是对应的哪个字段，这样是不是不太好看，想一想，我们可以通过python的哪一
```

```
#我们可以再创建游标的时候，在cursor里面加上一个参数：cursor=conn.cursor(cursor=pymysql.cursors.DictCursor)获取的结果
```

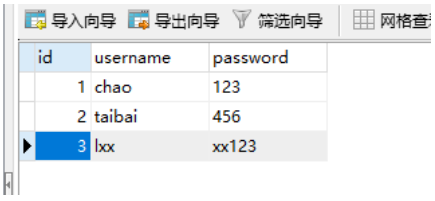
```
上面我们说，我们的数据取一次是不是就没有了啊，实际上不是的，这个取数据的操作就像读取文件内容一样，每次read之后，光标就
print(res) #一个数字

cursor.close() #关闭游标
conn.close()  #关闭连接

if res:
    print('登录成功')
else:
    print('登录失败')
```



二 execute()之sql注入

之前我们进行用户名密码认证是先将用户名和密码保存到一个文件中，然后通过读文件里面的内容，来和客户端发送过来的用户名密码进行匹配，现在我们学了数据库，我们可以将这些用户数据保存到数据库中，然后通过数据库里面的数据来对客户端进行用户名和密码的认证。

自行创建一个用户信息表userinfo，里面包含两个字段，username和password，然后里面写两条记录




```
#我们来使用数据来进行一下用户名和密码的认证操作import pymysql

conn = pymysql.connect(
    host='127.0.0.1',
    port=3306,
    user='root',
    password='666',
    database='crm',
    charset='utf8'
)

cursor = conn.cursor(pymysql.cursors.DictCursor)
uname = input('请输入用户名：')
pword = input('请输入密码：')

sql = "select * from userinfo where username='%s' and password='%s';"%(uname,pword)

res = cursor.execute(sql) #res我们说是得到的行数，如果这个行数不为零，说明用户输入的用户名和密码存在，如果为0说名存在，

print(res) #如果输入的用户名和密码错误，这个结果为0，如果正确，这个结果为1
if res:
    print('登陆成功')
else:
    print('用户名和密码错误！')
```

```
#通过上面的验证方式，比我们使用文件来保存用户名和密码信息的来进行验证操作要方便很多。
```

但是我们来看下面的操作，如果将在输入用户名的地方输入一个 chao'空格然后--空格然后加上任意的字符串，就能够登陆成功，也就是只知道用户名的情况下，他就能登陆成功的情况：

```
uname = input('请输入用户名：')
pword = input('请输入密码：')

sql = "select * from userinfo where username='%s' and password='%s';"%(uname,pword)
print(sql)
res = cursor.execute(sql) #res我们说是得到的行数，如果这个行数不为零，说明用户输入的用户名和密码存在，如果为0说名存在，

print(res) #如果输入的用户名和密码错误，这个结果为0，如果正确，这个结果为1
if res:
    print('登陆成功')
else:
    print('用户名和密码错误！')
#运行看结果：居然登陆成功
请输入用户名：chao' -- xxx
请输入密码：
select * from userinfo where username='chao' -- xxx' and password='';
1
登陆成功

我们来分析一下：
此时uname这个变量等于什么，等于chao' -- xxx,然后我们来看我们的sql语句被这个字符串替换之后是个什么样子：
select * from userinfo where username='chao' -- xxx' and password=''; 其中chao后面的这个'，在进行字符串替换的时候，我
```

然后我们再来看一个例子，直接连用户名和密码都不知道，但是依然能够登陆成功的情况：

```
请输入用户名：xxx' or 1=1 -- xxxxxx
请输入密码：
select * from userinfo where username='xxx' or 1=1 -- xxxxxx' and password='';
3
登陆成功

我们只输入了一个xxx' 加or 加 1=1 加 -- 加任意字符串
看上面被执行的sql语句你就发现了，or 后面跟了一个永远为真的条件，那么即便是username对不上，但是or后面的条件是成立的，t
```

上面两个例子就是两个sql注入的问题，看完上面这两个例子，有没有感觉后背发凉啊同志们，别急，我们来解决一下这个问题，怎么解决呢?

有些网站直接在你输入内容的时候，是不是就给你限定了，你不能输入一些特殊的符号，因为有些特殊符号可以改变sql的执行逻辑，其实不光是--，还有一些其他的符号也能改变sql语句的执行逻辑，这个方案我们是在客户端给用户输入的地方进行限制，但是别人可不可以模拟你的客户端来发送请求，是可以的，他模拟一个客户端，不按照你的客户端的要求来，就发一些特殊字符，你的客户端是限制不了的。所以单纯的在客户端进行这个特殊字符的过滤是不能解决根本问题的，那怎么办？我们服务端也需要进行验证，可以通过正则来将客户端发送过来的内容进行特殊字符的匹配，如果有这些特殊字符，我们就让它登陆失败。

在服务端来解决sql注入的问题：不要自己来进行sql字符串的拼接了，pymysql能帮我们拼接，他能够防止sql

注入，所以以后我们再写sql语句的时候按下面的方式写：

```
之前我们的sql语句是这样写的：
sql = "select * from userinfo where username='%s' and password='%s';"%(uname,pword)

以后再写的时候，sql语句里面的%s左右的引号去掉，并且语句后面的%(uname,pword)这些内容也不要自己写了，按照下面的方式写
sql = "select * from userinfo where username=%s and password=%s;"
难道我们不传值了吗，不是的，我们通过下面的形式，在excute里面写参数：
#cursor.execute(sql,[uname,pword])，其实它本质也是帮你进行了字符串的替换，只不过它会将uname和pword里面的特殊字符给

看下面的例子：
uname = input('请输入用户名：') #输入的内容是：chao' -- xxx或者xxx' or 1=1 -- xxxxx
pword = input('请输入密码：')

sql = "select * from userinfo where username=%s and password=%s;"
print(sql)
res = cursor.execute(sql,[uname,pword]) #res我们说是得到的行数，如果这个行数不为零，说明用户输入的用户名和密码存在，如

print(res) #如果输入的用户名和密码错误，这个结果为0，如果正确，这个结果为1
if res:
    print('登陆成功')
else:
    print('用户名和密码错误！')
#看结果：
请输入用户名：xxx' or 1=1 -- xxxxx
请输入密码：
select * from userinfo where username=%s and password=%s;
0
用户名和密码错误！
```

通过pymysql提供的excute完美的解决了问题。

```
总结咱们刚才说的两种sql注入的语句#1、sql注入之：用户存在，绕过密码
chao' -- 任意字符

#2、sql注入之：用户不存在，绕过用户与密码
xxx' or 1=1 -- 任意字符
```

解决方法总结：

```
# 原来是我们对sql进行字符串拼接
# sql="select * from userinfo where name='%s' and password='%s'" %(user,pwd)
# print(sql)
# res=cursor.execute(sql)

#改写为（execute帮我们做字符串拼接，我们无需且一定不能再为%s加引号了）
sql="select * from userinfo where name=%s and password=%s" #！！！注意%s需要去掉引号，因为pymysql会自动为我们加上
res=cursor.execute(sql,[user,pwd]) #pymysql模块自动帮我们解决sql注入的问题，只要我们按照pymysql的规矩来。
```

三 增、删、改：conn.commit()

查操作在上面已经说完了，我们来看一下增删改，也要注意，sql语句不要自己拼接，交给excute来拼接

```
import pymysql
#链接
conn=pymysql.connect(host='localhost',port='3306',user='root',password='123',database='crm',charset='utf8')
#游标
cursor=conn.cursor()

#执行sql语句
#part1
# sql='insert into userinfo(name,password) values("root","123456");'
# res=cursor.execute(sql) #执行sql语句，返回sql影响成功的行数
# print(res)# print(cursor.lastrowid) #返回的是你插入的这条记录是到了第几条了

#part2
# sql='insert into userinfo(name,password) values(%s,%s);'
# res=cursor.execute(sql,("root","123456")) #执行sql语句，返回sql影响成功的行数
# print(res)
#还可以进行更改操作：#res=cursor.excute("update userinfo set username='taibaisb' where id=2")#print(res) #结果为1
#part3
sql='insert into userinfo(name,password) values(%s,%s);'
res=cursor.executemany(sql,[("root","123456"),("lhf","12356"),("eee","156")]) #执行sql语句，返回sql影响成功的行数，一次插多
print(res)
#上面的几步，虽然都有返回结果，也就是那个受影响的函数res，但是你去数据库里面一看，并没有保存到数据库里面，
conn.commit() #必须执行conn.commit,注意是conn，不是cursor，执行这句提交后才发现表中插入记录成功，没有这句，上面的这
cursor.close()
conn.close()
```

四 查：fetchone，fetchmany，fetchall

```
mysql> select * from userinfo;
+----+------+----------+
| id | name | password |
+----+------+----------+
|  1 | root | 123456   |
|  2 | root | 123456   |
|  3 | root | 123456   |
|  4 | root | 123456   |
|  5 | root | 123456   |
|  6 | root | 123456   |
|  7 | lhf  | 12356    |
|  8 | eee  | 156      |
+----+------+----------+
8 rows in set (0.00 sec)
```

```
import pymysql
#链接
conn=pymysql.connect(host='localhost',user='root',password='123',database='egon')
#游标
cursor=conn.cursor()

#执行sql语句
sql='select * from userinfo;'
rows=cursor.execute(sql) #执行sql语句，返回sql影响成功的行数rows,将结果放入一个集合，等待被查询

# cursor.scroll(3,mode='absolute') # 相对绝对位置移动
# cursor.scroll(3,mode='relative') # 相对当前位置移动
res1=cursor.fetchone()
res2=cursor.fetchone()
```

```
res3=cursor.fetchone()
res4=cursor.fetchmany(2)
res5=cursor.fetchall()
print(res1)
print(res2)
print(res3)
print(res4)
print(res5)
print('%s rows in set (0.00 sec)' %rows)



conn.commit() #提交后才发现表中插入记录成功
cursor.close()
conn.close()

'''
(1, 'root', '123456')
(2, 'root', '123456')
(3, 'root', '123456')
((4, 'root', '123456'), (5, 'root', '123456'))
((6, 'root', '123456'), (7, 'lhf', '12356'), (8, 'eee', '156'))
rows in set (0.00 sec)
'''
```

五 获取插入的最后一条数据的自增ID

```
import pymysql
conn=pymysql.connect(host='localhost',user='root',password='123',database='egon')
cursor=conn.cursor()

sql='insert into userinfo(name,password) values("xxx","123");'
rows=cursor.execute(sql)
print(cursor.lastrowid) #在插入语句后查看

conn.commit()

cursor.close()
conn.close()
```