

PyMySQL的基本使用

什么是 PyMySQL?

PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个库，Python2中则使用mysqldb

PyMySQL 安装

在使用 PyMySQL 之前，我们需要确保 PyMySQL 已安装。

PyMySQL 下载地址: <https://github.com/PyMySQL/PyMySQL>。

如果还未安装，我们可以使用以下命令安装最新版的 PyMySQL：

```
1 | $ pip3 install PyMySQL
```

创建链接的基本使用

```
1  # 导入pymysql模块
2  import pymysql
3
4  # 连接database
5  conn = pymysql.connect(
6      host="你的数据库地址", # 本地为localhost
7      user="用户名",
8      password="密码",
9      database="数据库名",
10     charset="utf8")
11
12 # 得到一个可以执行SQL语句的光标对象
13 cursor = conn.cursor() # 执行完毕返回的结果集默认以元组显示
14 # 得到一个可以执行SQL语句并且将结果作为字典返回的游标
15 # cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)
16
17 # 定义要执行的SQL语句
18 sql = "SELECT VERSION()"
19
20 # 执行SQL语句
21 cursor.execute(sql)
22
23 # 使用 fetchone() 方法获取单条数据。
24 data = cursor.fetchone()
25
26 # 打印返回结果
27 print(data)
28 # 关闭游标对象
29 cursor.close()
30
31 # 关闭数据库连接
32 conn.close()
```

将数据库连接信息修改为自己的数据

```

1 conn = pymysql.connect(
2     host="localhost", # 本地为localhost
3     user="root", # 账号
4     port=3306, # 端口号
5     password="123456", # 密码
6     database="gm03") # 数据库名

```

执行以上脚本输出结果如下：

```

1 ('5.7.3-m13',)

```

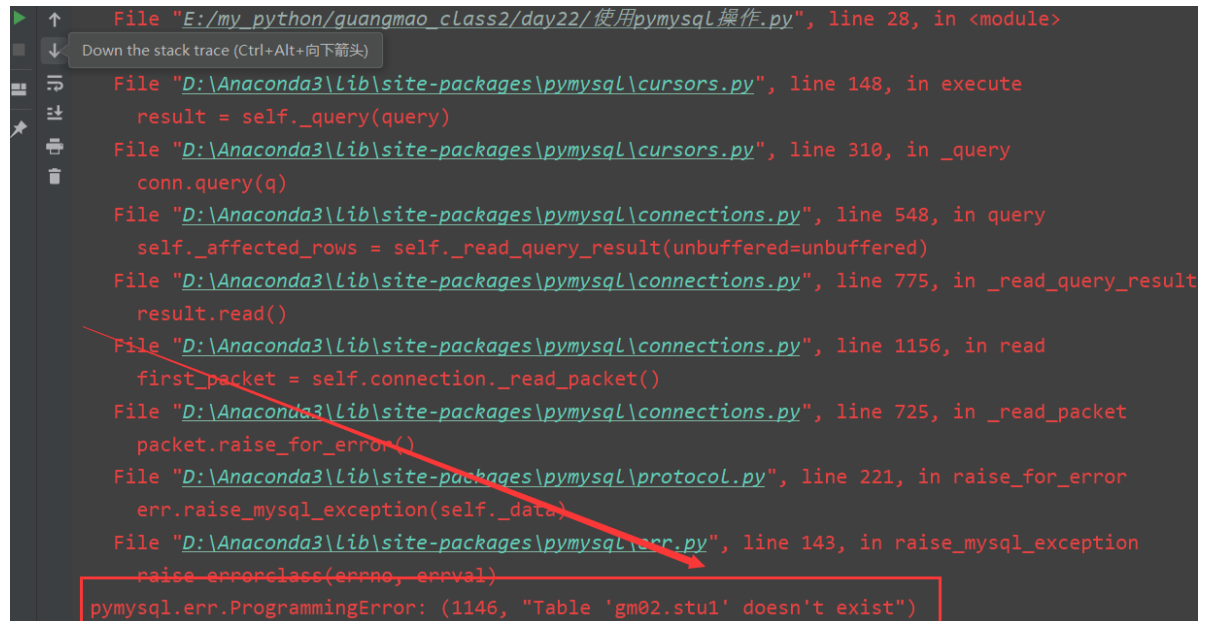
运行提示：检查数据库连接配置信息

```

1     raise errorclass(errno, errval)
2 pymysql.err.OperationalError: (1045, "Access denied for user
   'root'@'localhost' (using password: YES)")

```

在通过游标执行sql时报错，提示范例：



```

File "E:/my_python/quangmao_class2/day22/使用pymysql操作.py", line 28, in <module>
  Down the stack trace (Ctrl+Alt+向下箭头)
File "D:\Anaconda3\Lib\site-packages\pymysql\cursors.py", line 148, in execute
  result = self._query(query)
File "D:\Anaconda3\Lib\site-packages\pymysql\cursors.py", line 310, in _query
  conn.query(q)
File "D:\Anaconda3\Lib\site-packages\pymysql\connections.py", line 548, in query
  self._affected_rows = self._read_query_result(unbuffered=unbuffered)
File "D:\Anaconda3\Lib\site-packages\pymysql\connections.py", line 775, in _read_query_result
  result.read()
File "D:\Anaconda3\Lib\site-packages\pymysql\connections.py", line 1156, in read
  first_packet = self.connection._read_packet()
File "D:\Anaconda3\Lib\site-packages\pymysql\connections.py", line 725, in _read_packet
  packet.raise_for_error()
File "D:\Anaconda3\Lib\site-packages\pymysql\protocol.py", line 221, in raise_for_error
  err.raise_mysql_exception(self._data)
File "D:\Anaconda3\Lib\site-packages\pymysql\err.py", line 143, in raise_mysql_exception
  raise errorclass(errno, errval)
pymysql.err.ProgrammingError: (1146, "Table 'gm02.stu1' doesn't exist")

```

创建数据库表

如果数据库连接存在我们可以使用execute()方法来为数据库创建表，如下所示创建表user

```

1 import pymysql
2
3 # 打开数据库连接
4 conn = pymysql.connect(
5     host="localhost", # 本地为localhost
6     user="root", # 账号
7     port=3306, # 端口号
8     password="123456", # 密码
9     database="gm03") # 数据库名
10
11 # 使用 cursor() 方法创建一个游标对象 cursor
12 cursor = conn.cursor()
13

```

```

14 # 使用 execute() 方法执行 SQL，如果表存在则删除
15 cursor.execute("DROP TABLE IF EXISTS USER")
16
17 # 使用预处理语句创建表
18 sql = """CREATE TABLE user (
19     id INT auto_increment PRIMARY KEY,
20     user_name VARCHAR(20) NOT NULL,
21     pwd VARCHAR(20) NOT NULL,
22     reg_datetime DATETIME,
23     login_datetime DATETIME)"""
24
25 cursor.execute(sql)
26
27 # 关闭数据库连接
28 conn.close()

```

数据库插入操作

以下实例使用执行 SQL INSERT 语句向表 user 插入记录：

```

1  import pymysql
2
3  # 打开数据库连接
4  conn = pymysql.connect(
5      host="localhost", # 本地为localhost
6      user="root", # 账号
7      port=3306, # 端口号
8      password="123456", # 密码
9      database="gm03") # 数据库名
10
11 # 使用cursor()方法获取操作游标
12 cursor = conn.cursor()
13
14 # SQL 插入语句
15 sql = """INSERT INTO user(user_name,
16     pwd, reg_datetime)
17     VALUES ('jack', '123456', now())"""
18
19 # SQL 插入语句也可以写成如下形式：
20 # sql = """INSERT INTO user(user_name,
21 #     pwd, reg_datetime)
22 #     VALUES ('%s', '%s', now())""" % ('root', '123456')
23 try:
24     # 执行sql语句
25     cursor.execute(sql)
26     # 提交到数据库执行
27     conn.commit()
28 except Exception as e:
29     # 如果发生错误则回滚
30     conn.rollback()
31     print('异常', e)
32
33 # 关闭数据库连接
34 conn.close()

```

以下代码使用变量向SQL语句中传递参数：

```

1 .....
2 user_name = "test123"
3 password = "password"
4
5 con.execute("INSERT INTO user(user_name,
6             pwd, reg_datetime, login_datetime)
7             VALUES ('%s', '%s', now())" % (user_name, password))
8 .....

```

数据库查询操作

Python查询Mysql使用 fetchone() 方法获取单条数据, 使用fetchall() 方法获取多条数据。

- **fetchone():** 该方法获取下一个查询结果集。结果集是一个对象
- **fetchall():** 接收全部的返回结果行。
- **rowcount:** 这是一个只读属性，并返回执行execute()方法后影响的行数。

fetchall()

接收全部的返回结果行。

查询user表中的所有数据：

```

1 # 使用cursor()方法获取操作游标
2 cursor = conn.cursor()
3
4 # SQL 查询语句
5 sql = "SELECT * FROM user "
6 try:
7     # 执行SQL语句
8     cursor.execute(sql)
9     # 获取所有记录列表
10    results = cursor.fetchall()
11    print("使用fetchall返回的结果", results)
12    # ((1, 'jack', '123456', datetime.datetime(2021, 10, 21, 11, 22, 53),
13    None), (2, 'root', '123456', datetime.datetime(2021, 10, 21, 11, 25, 18),
14    None))
15    for row in results:
16        # 打印结果
17        print("id=%s,user_name=%s,pwd=%s,reg_datetime=%s,login_datetime=%s"
18              % \
19              (row[0], row[1], row[2], row[3], row[4]))
20 except:
21    print("Error: unable to fetch data")
22

```

结果为:

```

1 使用fetchall返回的结果: ((1, 'jack', '123456', datetime.datetime(2021, 10, 21,
2 11, 22, 53), None), (2, 'root', '123456', datetime.datetime(2021, 10, 21, 11,
3 25, 18), None))
4 id=1,user_name=jack,pwd=123456,reg_datetime=2021-10-21
5 11:22:53,login_datetime=None
6 id=2,user_name=root,pwd=123456,reg_datetime=2021-10-21
7 11:25:18,login_datetime=None

```

如果在创建游标是设置参数pymysql.cursors.DictCursor, 则返回为字典:

```
1 # 使用cursor()方法获取操作游标
2 cursor = conn.cursor(pymysql.cursors.DictCursor)
3
4 # SQL 查询语句
5 sql = "SELECT * FROM user "
6 try:
7     # 执行SQL语句
8     cursor.execute(sql)
9     # 获取所有记录列表
10    results = cursor.fetchall()
11    print('使用fetchall返回的结果: ',results)
12    for row in results:
13        # 打印结果
14        print("id=%s,user_name=%s,pwd=%s,reg_datetime=%s,login_datetime=%s"
15              % \
16                (row['id'], row['user_name'], row['pwd'], row['reg_datetime'],
17                 row['login_datetime']))
18 except:
19     print("Error: unable to fetch data")
```

结果为:

```
1 使用fetchall返回的结果: [{'id': 1, 'user_name': 'jack', 'pwd': '123456',
2    'reg_datetime': datetime.datetime(2021, 10, 21, 11, 22, 53),
3    'login_datetime': None}, {'id': 2, 'user_name': 'root', 'pwd': '123456',
4    'reg_datetime': datetime.datetime(2021, 10, 21, 11, 25, 18),
5    'login_datetime': None}]
6 id=1,user_name=jack,pwd=123456,reg_datetime=2021-10-21
7 11:22:53,login_datetime=None
8 id=2,user_name=root,pwd=123456,reg_datetime=2021-10-21
9 11:25:18,login_datetime=None
```

fetchone()

该方法获取下一个查询结果集。结果集是一个对象

```
1 cursor = conn.cursor()
2 # SQL 查询语句
3 sql = "SELECT * FROM user "
4 try:
5     # 执行SQL语句
6     cursor.execute(sql)
7     # 获取单条记录
8     results = cursor.fetchone()
9     print('使用fetone返回的结果: ', results)
10    print("id=%s,user_name=%s,pwd=%s,reg_datetime=%s,login_datetime=%s" % \
11          (results[0], results[1], results[2], results[3], results[4]))
12 except:
13     print("Error: unable to fetch data")
```

返回结果：

```
1 使用fetone返回的结果: (1, 'jack', '123456', datetime.datetime(2021, 10, 21, 11, 22, 53), None)
2  id=1,user_name=jack,pwd=123456,reg_datetime=2021-10-21 11:22:53,login_datetime=None
```

rowcount

这是一个只读属性，并返回执行execute()方法后影响的行数。

```
1  # 使用cursor()方法获取操作游标
2  cursor = conn.cursor()
3  # SQL 查询语句
4  sql = "SELECT * FROM user "
5  try:
6      # 执行SQL语句
7      cursor.execute(sql)
8      # 获取单条记录
9      row_count = cursor.rowcount
10     print('使用row_count返回的结果: ', row_count)
11 except:
12     print("Error: unable to fetch data")
```

返回的结果

```
1 使用row_count返回的结果:  2
```

执行事务

事务机制可以确保数据一致性。

事务应该具有4个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为ACID特性。

- 原子性 (atomicity) 。一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
- 一致性 (consistency) 。事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
- 隔离性 (isolation) 。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
- 持久性 (durability) 。持续性也称永久性 (permanence) ，指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

Python DB API 2.0 的事务提供了两个方法 commit 或 rollback。

数据库更新操作

更新操作用于更新数据表的数据，以下实例将 user表中 jack 的登录时间修改为当前时间

```
1  conn = pymysql.connect(
2      host="localhost", # 本地为localhost
3      user="root", # 账号
4      port=3306, # 端口号
5      password="123456", # 密码
6      database="gm03") # 数据库名
```

```

7  # 使用cursor()方法获取操作游标
8  cursor = conn.cursor()
9  # SQL 查询语句
10 sql = "update user set login_datetime=now() where user_name='jack'"
11 try:
12     # 执行SQL语句
13     cursor.execute(sql)
14     # 提交到数据库执行
15     conn.commit()
16 except Exception as e:
17     print("update error :", e)
18     #发生错误时回滚
19     conn.rollback()

```

原始及记录信息:

```
mysql> select * from user;
```

id	user_name	pwd	reg_datetime	login_datetime
1	jack	123456	2021-10-21 11:22:53	2021-10-21 16:33:43
2	root	123456	2021-10-21 11:25:18	NULL

程序执行后的结果:

```
mysql> select * from user;
```

id	user_name	pwd	reg_datetime	login_datetime
1	jack	123456	2021-10-21 11:22:53	2021-10-21 16:38:14

删除操作

删除操作用于删除数据表中的数据，以下实例演示了删除数据表 user表中 root账号：

```

1  # 使用cursor()方法获取操作游标
2  cursor = conn.cursor()
3  # SQL 查询语句
4  sql = "delete from user where user_name='root'"
5  try:
6     # 执行SQL语句
7     cursor.execute(sql)
8     # 提交到数据库执行
9     conn.commit()
10 except Exception as e:
11     print("update error :", e)
12     # conn.rollback()

```

原始及记录信息:

```
mysql> select * from user;
```

id	user_name	pwd	reg_datetime	login_datetime
1	jack	123456	2021-10-21 11:22:53	2021-10-21 16:33:43
2	root	123456	2021-10-21 11:25:18	NULL

操作后的结果：

```
mysql> select * from user;
```

id	user_name	pwd	reg_datetime	login_datetime
1	jack	123456	2021-10-21 11:22:53	2021-10-21 16:38:14

1 row in set (0.00 sec)

对于支持事务的数据库，在Python数据库编程中，当游标建立之时，就自动开始了一个隐形的数据库事务。

commit()方法游标的所有更新操作，rollback () 方法回滚当前游标的所有操作。每一个方法都开始了一个新的事务。

实例：

简单验证功能

用户输入账号和密码确定用户是否登录成功

```
1  #!/usr/bin/python3
2  import pymysql
3  # 隐藏密码模块
4  import getpass
5
6
7  class ActionMysql:
8      def __init__(self, host, user, password, database, port=3306,
9          charset='utf8'):
10         # 打开数据库连接
11         self.conn = pymysql.connect(
12             host=host, # 本地为localhost
13             port=port, # 端口号
14             user=user, # 账号
15             password=password, # 密码
16             database=database,
17             charset=charset) # 数据库名
18         self.cursor = self.conn.cursor(pymysql.cursors.DictCursor)
19
20     def fetch_one(self, sql):
21         """
22         执行查询，返回一条数据
23         :param sql:
24         :return:
25         """
26         try:
27             self.cursor.execute(sql)
28             res = self.cursor.fetchone()
29             return res
30         except Exception as e:
31             print("fetch_one error:", e)
32             return None
33
34     def find_exists(self, sql):
35         """
```



```

35         执行查询返回结果是否存在
36         :param sql:
37         :return:
38         """
39         try:
40             self.cursor.execute(sql)
41             row_count = self.cursor.rowcount
42             # 判断查询结果是否大于0
43             if row_count:
44                 return True
45         except Exception as e:
46             print("fetch_one error:", e)
47         return False
48
49     def __del__(self):
50         """
51         执行数据库连接删除操作
52         :return:
53         """
54         # 关闭游标
55         self.cursor.close()
56         # 关闭数据库连接
57         self.conn.close()
58
59
60 if __name__ == '__main__':
61     # 实例化mysql操作对象
62     my_mysql = ActionMysql(host="localhost", # 本地为localhost
63                             user="root", # 账号
64                             password="123456", # 密码
65                             database="gm03")
66     while True:
67         user_name = input("账号>>>")
68         #pwd = input("密码>>>")
69         # 调用隐藏密码模块 ,在pycharm的IDE中无效
70         pwd = getpass.getpass("pwd>>>")
71         sql = 'select * from user where user_name="%s" and pwd="%s"' %
72         (user_name, pwd)
73         if my_mysql.find_exists(sql):
74             print('登录成功, 欢迎', user_name)
75             break
76         else:
77             print('登录失败重新登录')

```

PyMySQL防止SQL注入

一、SQL注入简介

SQL注入是比较常见的网络攻击方式之一，它不是利用**操作系统**的BUG来实现攻击，而是针对程序员编程时的疏忽，通过SQL语句，实现无帐号登录，甚至篡改**数据库**。

二、SQL注入攻击的总体思路

- 1.寻找到SQL注入的位置
- 2.判断服务器类型和后台数据库类型

3.针对不通的服务器和数据库特点进行SQL注入攻击

三、SQL注入攻击实例

1、字符串拼接查询，造成注入

注意mysql的单行注释为:

- #
- --空格

比如: 上个实例中如果用户输入的账号为: "or 1 = 1 or"1=1

```
1 user_name = ' " or 1 = 1 -- '
2 pwd = ""
3 sql = 'select * from user where user_name="%s" and pwd = "%s" ' % (user_name,
4   print(sql)
5   print(my_mysql.find_exists(sql))
```

打印sql结果为:

```
1 select * from user where user_name="" or 1 = 1 -- " and pwd = ""
2 True
```

当用户账号输入: " or 1 = 1 -- 时, 就可以登录成功。

因为条件后面user_name="or 1=1 用户名等于 " 或1=1 那么这个条件一定会成功; 然后后面加两个-, 这意味着注释, 它将后面的语句注释, 让他们不起作用, 这样语句永远都能正确执行, 用户轻易骗过系统, 获取合法身份。

解决方法:

1、使用pymysql提供的参数化语句防止注入

使用

```
1 execute() 函数本身就有接受SQL语句变量的参数位, 只要正确的使用就可以对传入的值进行correctly
   转义, 从而避免SQL注入的发生
2 使用execute()进行sql语句拼接
```

如: 注意execute 使用2个参数时, 第一个参数中需要替换的占位符不需要引号

```
1 cursor.execute('select * from user where user_name=%s and pwd = %s ',
   (user_name,pwd))
```

```
1 conn = pymysql.connect(
2     host="localhost", # 本地为localhost
3     user="root", # 账号
4     port=3306, # 端口号
5     password="123456", # 密码
6     database="gm03") # 数据库名
7
8 # 使用cursor()方法获取操作游标
9 cursor = conn.cursor()
```

```

10 # SQL 查询语句
11 sql = 'select * from user where user_name="%s" and pwd = "%s" '
12 try:
13     # " or 1 = 1 --
14     user_name = input("账号>>>")
15     pwd = input("密码>>>")
16     # 使用
17     cursor.execute(sql, (user_name, pwd))
18     # 获取单条记录
19     row_count = cursor.rowcount
20     print('使用row_count返回的结果: ', row_count)
21 except:
22     print("Error: unable to fetch data")
23
24 # 关闭数据库连接
25 conn.close()

```

返回结果:

```

1  账号>>>" or 1 = 1 --
2  密码>>>
3  使用row_count返回的结果:  0

```

对实例内容进行修改, 新增安全查询方法:

```

1  def safe_find_exists(self, sql, params):
2      """
3      安全操作sql
4      :param sql:
5      :param params:
6      :return:
7      """
8      try:
9          self.cursor.execute(sql, params)
10         row_count = self.cursor.rowcount
11         # 判断查询结果是否大于0
12         if row_count:
13             return True
14     except Exception as e:
15         print("fetch_one error:", e)
16     return False

```

代码中调用安全方法:

```

1  user_name = input("账号>>>")
2  #pwd = input("密码>>>")
3  # 调用隐藏密码模块 ,在pycharm的IDE中无效
4  pwd = getpass.getpass("pwd>>>")
5  sql = 'select * from user where user_name="%s" and pwd="%s"'
6  # 调用安全方法
7  if my_mysql.safe_find_exists(sql, (user_name, pwd)):
8      print('登录成功, 欢迎', user_name)
9      break
10 else:
11     print('登录失败重新登录')

```

全代码:

```
1 #!/usr/bin/python3
2 import pymysql
3 # 隐藏密码模块
4 import getpass
5
6
7 class ActionMysql:
8     def __init__(self, host, user, password, database, port=3306,
9         charset='utf8'):
10         # 打开数据库连接
11         self.conn = pymysql.connect(
12             host=host, # 本地为localhost
13             port=port, # 端口号
14             user=user, # 账号
15             password=password, # 密码
16             database=database,
17             charset=charset) # 数据库名
18         self.cursor = self.conn.cursor(pymysql.cursors.DictCursor)
19
20     def fetch_one(self, sql):
21         """
22         执行查询, 返回一条数据
23         :param sql:
24         :return:
25         """
26         try:
27             self.cursor.execute(sql)
28             res = self.cursor.fetchone()
29             return res
30         except Exception as e:
31             print("fetch_one error:", e)
32             return None
33
34     def find_exists(self, sql):
35         """
36         执行查询返回结果是否存在
37         :param sql:
38         :return:
39         """
40         try:
41             self.cursor.execute(sql)
42             row_count = self.cursor.rowcount
43             # 判断查询结果是否大于0
44             if row_count:
45                 return True
46         except Exception as e:
47             print("fetch_one error:", e)
48             return False
49
50     def safe_find_exists(self, sql, params):
51         """
52         安全操作sql
53         :param sql:
```

```

53         :param params:
54         :return:
55         """
56         try:
57             self.cursor.execute(sql, params)
58             row_count = self.cursor.rowcount
59             # 判断查询结果是否大于0
60             if row_count:
61                 return True
62         except Exception as e:
63             print("fetch_one error:", e)
64         return False
65
66     def __del__(self):
67         """
68         执行数据库连接删除操作
69         :return:
70         """
71         # 关闭游标
72         self.cursor.close()
73         # 关闭数据库连接
74         self.conn.close()
75
76
77 if __name__ == '__main__':
78     # 实例化mysql操作对象
79     my_mysql = ActionMysql(host="localhost", # 本地为localhost
80                             user="root", # 账号
81                             password="123456", # 密码
82                             database="gm03")
83     while True:
84         # 例 # " or 1 = 1 --
85         user_name = input("账号>>>")
86         #pwd = input("密码>>>")
87         # 调用隐藏密码模块 ,在pycharm的IDE中无效
88         pwd = getpass.getpass("pwd>>>")
89         # 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）
90         sql = 'select * from user where user_name=%s and pwd=%s'
91         if my_mysql.safe_find_exists(sql, (user_name, pwd)):
92             print('登录成功，欢迎', user_name)
93             break
94         else:
95             print('登录失败重新登录')
96

```

将上面的语句改为下面这样，注意一定不要再加引号了

```

1 #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）
2 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
3 result=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。
4

```

