



# Python\_异常处理



# Python进阶课程

0

异常

117

6

数据库：MySQL

132

6

Python与MySQL的交互

171

7



## 5、异常



# 5.1 异常

- <1>异常简介

看如下示例:

```
print '-----test--1---'  
open('123.txt','r')  
print '-----test--2---'
```

运行结果:

```
code@ubuntu:~/python-test$ python test.py  
-----test--1---  
Traceback (most recent call last):  
  File "test.py", line 2, in <module>  
    open('123.txt','r')  
IOError: [Errno 2] No such file or directory: '123.txt'
```

说明:

打开一个不存在的文件123.txt，当找不到123.txt文件时，就会抛出给我们一个IOError类型的错误，No such file or directory: 123.txt（没有123.txt这样的文件或目录）

异常:

当Python检测到一个错误时，解释器就无法继续执行了，反而出现了一些错误的提示，这就是所谓的“异常”

## 5.1 异常

- <1>捕获异常 try...except...

```
try:
    print('-----test--1---')
    open('123.txt','r')
    print('-----test--2---')
except IOError:
    pass
```

运行结果:

```
MacBook-Pro 01-python基础班-资料$ python test.py
-----test--1---
MacBook-Pro 01-python基础班-资料$
```

说明:

- 此程序看不到任何错误，因为用except 捕获到了IOError异常，并添加了处理的方法
- pass 表示实现了相应的实现，但什么也不做；如果把pass改为print语句，那么就会输出其他信息

小总结:

```
try:
    print '-----test--1---'
    open('123.txt','r')
    print '-----test--2---'
except IOError:
    pass
```

可能产生异常的代码，放在try中

如果产生错误时，处理的方法

- 把可能出现问题的代码，放在try中
- 把处理异常的代码，放在except中

## 5.1 异常

- <2> except捕获多个异常

看如下示例:

```
try:
    print num
except IOError:
    print('产生错误了')
```

运行结果如下:

```
MacBook-Pro 01-python基础班-资料$ python test.py
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print num
NameError: name 'num' is not defined
```

- 想一想:
- 上例程序, 已经使用except来捕获异常了, 为什么还会看到错误的信息提示?
- 答:
- except捕获的错误类型是IOError, 而此时程序产生的异常为 NameError , 所以except没有生效

## 5.1 异常

- 实际开发中，捕获多个异常的方式，

```
#coding=utf-8
try:
    print('-----test--1---')
    open('123.txt','r') # 如果123.txt文件不存在，那么会产生 IOError 异常
    print('-----test--2---')
    print(num)# 如果num变量没有定义，那么会产生 NameError 异常

except (IOError,NameError):
    #如果想通过一次except捕获到多个异常可以用一个元组的方式

    # errorMsg里会保存捕获到的错误信息
    print(errorMsg)
```



A terminal window titled '桌面 - -bash - 68x9' showing the execution of a Python script. The prompt is 'dongGe@dongGe-Mac Desktop\$'. The command 'python filetest.py' is entered. The output shows '-----test--1---', '-----test--2---', and then the error message 'name 'num' is not defined'. The prompt returns to 'dongGe@dongGe-Mac Desktop\$'.

- 如下：注意：
- 当捕获多个异常时，可以把要捕获的异常的名字，放到except后，并使用元组的方式仅进行存储

## 5.1 异常

- <3>获取异常的信息描述

```
In [8]: try:
...:     print(a)
...: except NameError as result:
...:     print(result)
...:
name 'a' is not defined
```

存储异常的  
基本信息

要捕获的异常

```
In [13]: try:
...:     open("a.txt")
...: except (NameError, IOError) as result:
...:     print("哈哈，捕获到了异常")
...:     print("异常的基本信息是:", result)
...:
哈哈，捕获到了异常
异常的基本信息是: [Errno 2] No such file or directory: 'a.txt'
```

捕获多个异常，并且存储异常的基本信息



## 5.1 异常

- <4>捕获所有异常

```
In [14]: try:
...:     open("a.txt")
...: except:
...:     print("产生了一个异常")
...:
产生了一个异常
```

没有存储异常  
的基本信息

```
In [15]: try:
...:     open("a.txt")
...: except Exception as result:
...:     print("捕获到了异常")
...:     print(result)
...:
捕获到了异常
[Errno 2] No such file or directory: 'a.txt'
```

捕获所有异常，  
并且存储异常的基本  
信息

## 5.1 异常

- **<5> else**
- 咱们应该对else并不陌生，在if中，它的作用是当条件不满足时执行的实行；同样在try...except...中也是如此，即如果没有捕获到异常，那么就执行else中的事情

```
try:
    num = 100
    print num
except NameError as errorMsg:
    print('产生错误了:%s'%errorMsg)
else:
    print('没有捕获到异常，真高兴')
```

```
In [17]: try:
.....:     num = 100
.....:     print(num)
.....: except NameError as result:
.....:     print("捕获到了一个异常，信息是:%s"%result)
.....: else:
.....:     print("程序正常运行没有捕获到异常")
.....:
100
程序正常运行没有捕获到异常
```

## 5.1 异常

- **<6> try...finally...**
- try...finally...语句用来表达这样的情况:
- 在程序中, 如果一个段代码必须要执行, 即无论异常是否产生都要执行, 那么此时就需要使用**finally**。比如文件关闭, 释放锁, 把数据库连接返还给连接池等
- demo:

```
import time
try:
    f = open('test.txt')
    try:
        while True:
            content = f.readline()
            if len(content) == 0:
                break
            time.sleep(2)
            print(content)
    except:
        #如果在读取文件的过程中, 产生了异常, 那么就会捕获到
        #比如 按下了 ctrl+c
        pass
    finally:
        f.close()
        print('关闭文件')
except:
    print("没有这个文件")
```

## 5.2 异常的传递

- 1. try嵌套中

```
import time
try:
    f = open('test.txt')
    try:
        while True:
            content = f.readline()
            if len(content) == 0:
                break
            time.sleep(2)
            print(content)
    finally:
        f.close()
        print('关闭文件')
except:
    print("没有这个文件")
```

```
In [26]: import time
...: try:
...:     f = open('test.txt')
...:     try:
...:         while True:
...:             content = f.readline()
...:             if len(content) == 0:
...:                 break
...:             time.sleep(2)
...:             print(content)
...:     finally:
...:         f.close()
...:         print('关闭文件')
...: except:
...:     print("没有这个文件")
...: finally:
...:     print("最后的finally")
...:
```

```
xxxxxxx--->这是test.txt文件中读取到信息
^C关闭文件
没有这个文件
最后的finally
```

## 5.2 异常的传递

- 2. 函数嵌套调用中

```
def test1():  
    print("----test1-1----")  
    print(num)  
    print("----test1-2----")  
  
def test2():  
    print("----test2-1----")  
    test1()  
    print("----test2-2----")  
  
def test3():  
    try:  
        print("----test3-1----")  
        test1()  
        print("----test3-2----")  
    except Exception as result:  
        print("捕获到了异常，信息是:%s"%result)  
  
    print("----test3-2----")
```

```
python@ubuntu:~/Desktop/test$ python3 04-tye-except.py  
----test3-1----  
----test1-1----  
捕获到了异常，信息是:name 'num' is not defined  
----test3-3----  
-----华丽的分割线-----  
----test2-1----  
----test1-1----  
Traceback (most recent call last):  
  File "04-tye-except.py", line 26, in <module>  
    test2()  
  File "04-tye-except.py", line 8, in test2  
    test1()  
  File "04-tye-except.py", line 3, in test1  
    print(num)  
NameError: name 'num' is not defined
```

## 5.2 异常的传递

- 总结:
- 如果try嵌套，那么如果里面的try没有捕获到这个异常，那么外面的try会接收到这个异常，然后进行处理，如果外边的try依然没有捕获到，那么再进行传递。。。
- 如果一个异常是在一个函数中产生的，例如函数A---->函数B---->函数C,而异常是在函数C中产生的，那么如果函数C中没有对这个异常进行处理，那么这个异常会传递到函数B中，如果函数B有异常处理那么就会按照函数B的处理方式进行执行；如果函数B也没有异常处理，那么这个异常会继续传递，以此类推。。。如果所有的函数都没有处理，那么此时就会进行异常的默认处理，即通常见到的那样
- 注意观察上图中，当调用test3函数时，在test1函数内部产生了异常，此异常被传递到test3函数中完成了异常处理，而当异常处理完后，并没有返回到函数test1中进行执行，而是在函数test3中继续执行

## 5.3 抛出自定义的异常

- 你可以用`raise`语句来引发一个异常。异常/错误对象必须有一个名字，且它们应是`Error`或`Exception`类的子类
- 下面是一个引发异常的例子:

```
class ShortInputException(Exception):
    '''自定义的异常类'''
    def __init__(self, length, atleast):
        #super().__init__()
        self.length = length
        self.atleast = atleast

def main():
    try:
        s = input('请输入 --> ')
        if len(s) < 3:
            # raise引发一个你定义的异常
            raise ShortInputException(len(s), 3)
    except ShortInputException as result: #x这个变量被绑定到了错误的实例
        print('ShortInputException: 输入的长度是 %d, 长度至少应是 %d' % (result.length, result.atleast))
    else:
        print('没有异常发生。')
```

运行结果如下:

```
python@ubuntu:~/Desktop/test$ subl 05-try-except.py
python@ubuntu:~/Desktop/test$ python3 05-try-except.py
请输入 --> hello
没有异常发生.
python@ubuntu:~/Desktop/test$ python3 05-try-except.py
请输入 --> abc
没有异常发生.
python@ubuntu:~/Desktop/test$ python3 05-try-except.py
请输入 --> ab
ShortInputException: 输入的长度是 2, 长度至少应是 3
python@ubuntu:~/Desktop/test$ python3 05-try-except.py
请输入 --> a
ShortInputException: 输入的长度是 1, 长度至少应是 3
python@ubuntu:~/Desktop/test$ python3 05-try-except.py
请输入 --> 
ShortInputException: 输入的长度是 0, 长度至少应是 3
python@ubuntu:~/Desktop/test$
```

只输入回车

## 5.3 抛出自定义的异常

- 注意
- 以上程序中，关于代码`#super().__init__()`的说明这一行代码，可以调用也可以不调用，建议调用，因为`__init__`方法往往是用来对创建完的对象进行初始化工作，如果在子类中重写了父类的`__init__`方法，即意味着父类中的很多初始化工作没有做，这样就不保证程序的稳定了，所以在以后的开发中，如果重写了父类的`__init__`方法，最好是先调用父类的这个方法，然后再添加自己的功能



## 5.4 异常处理中抛出异常

- 抛出异常

```
class Test(object):
    def __init__(self, switch):
        self.switch = switch #开关
    def calc(self, a, b):
        try:
            return a/b
        except Exception as result:
            if self.switch:
                print("捕获开启, 已经捕获到了异常, 信息如下:")
                print(result)
            else:
                #重新抛出这个异常, 此时就不会被这个异常处理给捕获到, 从而触发默认异常处理
                raise

a = Test(True)
a.calc(11,0)

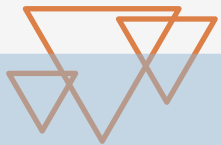
print("-----华丽的分割线-----")

a.switch = False
a.calc(11,0)
```



## 6、数据库：MySQL





# CONTENTS

1

MySQL的简介

2

MySQL的图形操作

3

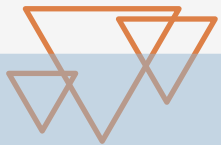
MySQL的命令操作

3.1 数据库的操作

3.2 表的操作

3.3 数据的操作

3.4 备份和还原



# CONTENTS

## 4

### MySQL查询

- 4.1 比较运算符
- 4.2 逻辑运算符
- 4.3 模糊查询
- 4.4 范围查询
- 4.5 空判断查询
- 4.6 聚合
- 4.7 分组

## 6.1 MySQL简介

## 6.1 简介

### 简介

- mysql数据库，是当前应用非常广泛的一款关系型数据库
- 查看[官方网站](#)
- 查看[数据库排名](#)
- 主要知识点包括：
  - 数据库与表的创建、删除
  - 字段的类型、约束
  - 关系的存储
  - 数据行的增加、修改、删除
  - 数据行的查询
  - 视图、事务、索引
  - 与python交互

## 6.1 简介

### 简介

- 主要知识点包括：能够与mysql建立连接，创建数据库、表，分别从图形界面与脚本界面两个方面讲解
- 相关的知识点包括：E-R关系模型，数据库的3范式，mysql中数据字段的类型，字段约束
- 主要操作包括：
  - 数据库的操作，包括创建、删除
  - 表的操作，包括创建、修改、删除
  - 数据的操作，包括增加、修改、删除、查询，简称crud

## 6.1 简介

- 本节主要完成数据库的操作、表的操作、数据的增加、修改、删除操作，分别从图形窗口、命令两个方面掌握
- 学生表结构：
  - id
  - 姓名
  - 性别
  - 地址
  - 生日
- 科目表结构：
  - id
  - 名称



## 6.1 简介

### 数据完整性

- 一个数据库就是一个完整的业务单元，可以包含多张表，数据被存储在表中
- 在表中为了更加准确的存储数据，保证数据的正确有效，可以在创建表的时候，为表添加一些强制性的验证，包括数据字段的类型、约束



## 6.1 简介

### 字段类型

- 在mysql中包含的数据类型很多，这里主要列出来常用的几种
- 数字：int,decimal
- 字符串：char,varchar,text
- 日期：datetime
- 布尔：bit

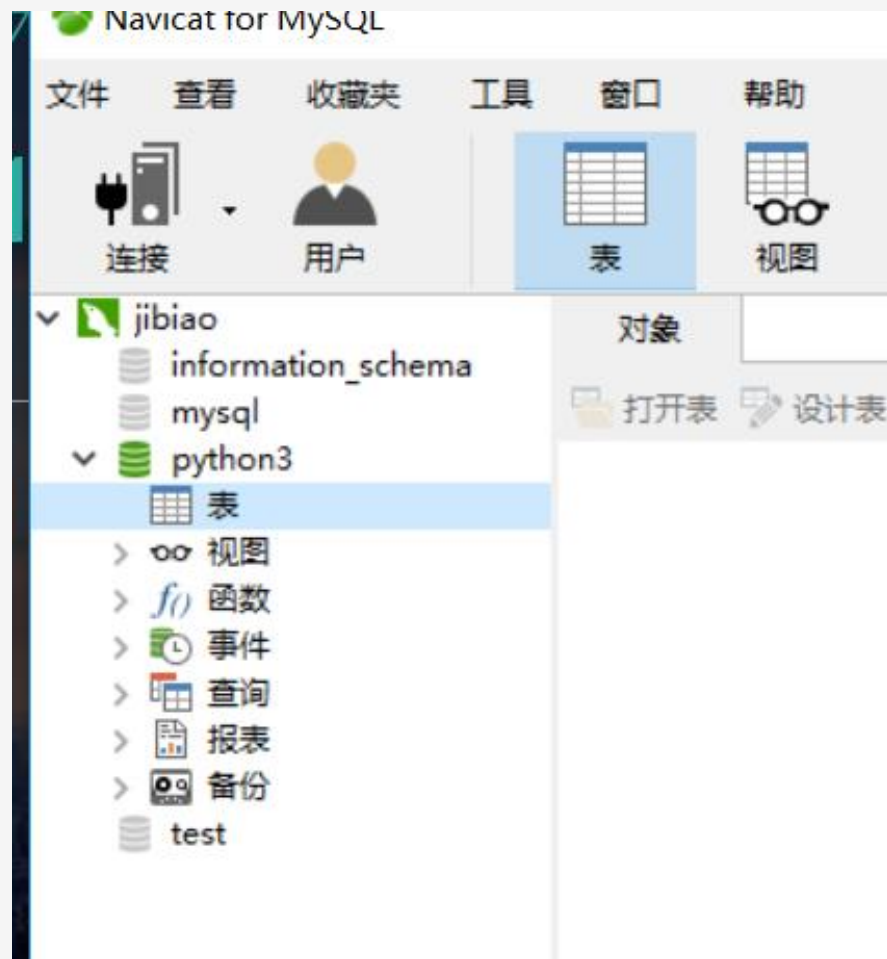
## 6.1 简介

### 约束


- 主键primary key
- 非空not null
- 惟一unique
- 默认default
- 外键foreign key

## 6.2 MySql的图形操作


## 6.2 Navicat for MySQL



## 6.2 Navicat for MySQL

栏位	索引	外键	触发器	选项	注释	SQL 预览
名	类型	长度	小数点	不是 null		
▶ id	int	11	0	<input checked="" type="checkbox"/>	 1	
name	varchar	255	0	<input checked="" type="checkbox"/>		
birthday	datetime	0	0	<input checked="" type="checkbox"/>		
gender	bit	1	0	<input checked="" type="checkbox"/>		

## 6.2 Navicat for MySQL

  开始事务  备注  筛选 			
id	name	birthday	gender
▶ 1	路飞	2000-01-01	0 1
2	索隆	1995-02-01	0 1
3	香吉士	1996-03-01	0 1
4	娜美	2000-04-01	0 0
5	罗宾	1989-05-01	0 0
6	乔巴	2005-06-01	0 1
7	乌索普	2001-07-01	0 1
8	弗兰奇	1988-09-01	0 1
9	布鲁克	1950-10-01	0 1

## 6.2 Navicat for MySQL

### 逻辑删除

- 对于重要数据，并不希望物理删除，一旦删除，数据无法找回
- 一般对于重要数据，会设置一个isDelete的列，类型为bit，表示逻辑删除
- 大于大量增长的非重要数据，可以进行物理删除
- 数据的重要性，要根据实际开发决定



## 6.3 MySql的命令操作

## 6.3 MySQL的命令操作

通过终端进入数据库：

```
mysql -uroot -p:进入MySQL里面
```

## 6.3 MySQL的命令操作

- 创建数据库

```
create database 数据库名 charset=utf8;
```

- 删除数据库

```
drop database 数据库名;
```

- 切换数据库

```
use 数据库名;
```

- 查看当前选择的数据库

```
select database();
```

## 6.3 MySQL的命令操作

- 查看当前数据库中所有表

```
show tables;
```

- 创建表
- auto\_increment表示自动增长

```
create table 表名(列及类型);  
如：  
create table students(  
id int auto_increment primary key,  
sname varchar(10) not null  
);
```

## 6.3 MySql的命令操作

注意：change表，只能修改类型。

- 修改表

```
alter table 表名 add|change|drop 列名 类型;
```

如：

```
alter table students add birthday datetime;
```

## 6.3 MySQL的命令操作

- 删除表

```
drop table 表名;
```

- 查看表结构

```
desc 表名;
```

- 更改表名称

```
rename table 原表名 to 新表名;
```

- 查看表的创建语句

```
show create table '表名';
```

## 6.3 MySQL的命令操作

- 查询

```
select * from 表名;
```

- 增加

全列插入: `insert into 表名 values(...)`

缺省插入: `insert into 表名(列1,...) values(值1,...)`

同时插入多条数据: `insert into 表名 values(...),(...)...;`

或 `insert into 表名(列1,...) values(值1,...),(值1,...)...;`

## 6.3 MySQL的命令操作

- 修改

```
update 表名 set 列1=值1,... where 条件
```

- 删除

```
delete from 表名 where 条件
```

- 逻辑删除，本质就是修改操作update

```
alter table students add isdelete bit default 0;
```

- 逻辑删除，本质就是修改操作update

```
alter table students add isdelete bit default 0;
```

如果需要删除则

```
update students isdelete=1 where ...;
```



## 6.3 MySql的命令操作

### 备份与恢复



#### 数据备份

- 进入超级管理员

```
sudo -s
```

- 进入mysql库目录

```
cd /var/lib/mysql
```

- 运行mysqldump命令

```
mysqldump -uroot -p 数据库名 > ~/Desktop/备份文件.sql;
```

按提示输入mysql的密码

## 6.3 MySql的命令操作

### 数据恢复

- 连接mysql, 创建数据库
- 退出连接, 执行如下命令

```
mysql -uroot -p 数据库名 < ~/Desktop/备份文件.sql
```

根据提示输入mysql密码

## 6.4 MySql查询

## 6.4 MySQL查询

### 简介

- 查询的基本语法

```
select * from 表名;
```

- from关键字后面写表名，表示数据来源于这张表
- select后面写表中的列名，如果是\*表示在结果中显示表中所有列
- 在select后面的列名部分，可以使用as为列起别名，这个别名出现在结果集中
- 如果要查询多个列，之间使用逗号分隔

### 消除重复行

- 在select后面列前使用distinct可以消除重复的行

```
select distinct gender from students;
```

## 6.4 MySQL查询

### 条件

- 使用where子句对表中的数据筛选，结果为true的行会出现在结果集中
- 语法如下：

```
select * from 表名 where 条件;
```

### 比较运算符

- 等于=
- 大于>
- 大于等于>=
- 小于<
- 小于等于<=
- 不等于!=或<>
- 查询编号大于3的学生

```
select * from students where id>3;
```

## 6.4 MySQL查询

### 逻辑运算符

- **and**
- or
- not
- 查询编号大于3的女同学

```
select * from students where id>3 and gender=0;
```

- 查询编号小于4或没被删除的学生

```
select * from students where id<4 or isdelete=0;
```

## 6.4 MySQL查询

### 模糊查询

- like
- %表示任意多个任意字符
- \_表示一个任意字符
- 查询姓黄的学生

```
select * from students where sname like '黄%';
```

- 查询姓黄并且名字是一个字的学生

```
select * from students where sname like '黄_';
```

- 查询姓黄或叫靖的学生

```
select * from students where sname like '黄%' or sname like '%靖%';
```

## 6.4 MySQL查询

### 范围查询

- in表示在一个非连续的范围
- 查询编号是1或3或8的学生

```
select * from students where id in(1,3,8);
```

- between ... and ...表示在一个连续的范围
- 查询学生是3至8的学生

```
select * from students where id between 3 and 8;
```

- 查询学生是3至8的男生

```
select * from students where id between 3 and 8 and gender=1;
```



## 6.4 MySQL查询

### 空判断

- 注意：null与"是不同的
- 判空is null
- 查询没有填写地址的学生

```
select * from students where hometown is null;
```

- 判非空is not null
- 查询填写了地址的学生

```
select * from students where hometown is not null;
```

- 查询填写了地址的女生

```
select * from students where hometown is not null and gender=0;
```

## 6.4 MySql查询

### 优先级

- 小括号，not，比较运算符，逻辑运算符
- and比or先运算，如果同时出现并希望先算or，需要结合()使用

## 6.4 MySql查询

### 聚合

- 为了快速得到统计数据，提供了5个聚合函数
- `count(*)`表示计算总行数，括号中写星与列名，结果是相同的
- 查询学生总数

```
select count(*) from students;
```

- `max(列)`表示求此列的最大值
- 查询女生的编号最大值

```
select max(id) from students where gender=0;
```

- `min(列)`表示求此列的最小值
- 查询未删除的学生最小编号

```
select min(id) from students where isdelete=0;
```

## 6.4 MySQL查询

- `sum(列)`表示求此列的和
- 查询男生的编号之后

```
select sum(id) from students where gender=1;
```

- `avg(列)`表示求此列的平均值
- 查询未删除女生的编号平均值

```
select avg(id) from students where isdelete=0 and gender=0;
```

## 6.4 MySQL查询

### 分组

- 按照字段分组，表示此字段相同的数据会被放到一个组中
- 分组后，只能查询出相同的数据列，对于有差异的数据列无法出现在结果集中
- 可以对分组后的数据进行统计，做聚合运算
- 语法：

```
select 列1,列2,聚合... from 表名 group by 列1,列2,列3...
```

- 查询男女生总数

```
select gender as 性别,count(*)  
from students  
group by gender;
```

## 6.4 MySql查询

### 对比where与having

- where是对from后面指定的表进行数据筛选，属于对原始数据的筛选
- having是对group by的结果进行筛选

## 6.4 MySQL查询

- 分组1:
- `select count(*) from students group by gender;`
- 分组2:
- `select gender, count(*) from students group by gender;`
- 分组3:
- `select gender, count(*) as res from students group by gender;`

## 6.4 MySql查询

- having后面的条件运算符与where的相同
- 查询男生总人数

方案一

```
select count(*)  
from students  
where gender=1;
```

-----

方案二：

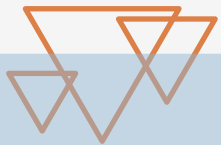
```
select gender as 性别,count(*)  
from students  
group by gender  
having gender=1;
```





## 7、python与MySQL的交互





# CONTENTS


- 1 python中mysql模块的安装
- 2 pymysql模块的使用
- 3 pymysql模块中cursor的使用
- 4 MySql封装
- 5 案例： 用户登录



## 7.1 python中mysql模块的安装

## 7.1 pymysql模块安装

### 与python交互

- 在熟练使用sql语句的基础上，开始使用python语言提供的模块与mysql进行交互
- 这是我们在工作中大量要做的事
- 先学会sql是基础，定要熟练编写sql语句

## 7.1 pymysql模块安装

### 安装引入模块

- 安装mysql模块

```
sudo apt-get install python-mysqldb
```

- 在文件中引入模块

```
import MySQLdb
```

## 7.1 pymysql模块安装

- 1、为什么要安装MySQL-python?
- 要想使python可以操作mysql 就需要MySQL-python驱动，它是python 操作mysql必不可少的模块。
- 2、注意点：
- 安装mysql-python报错，发现mysql-python只更新支持到了python3.4，可以用pymysql替代mysql-python。
- `pip install pymysql`

## 7.2 pymysql模块的使用

## 7.2 pymysql模块的使用

### Connection对象

- 用于建立与数据库的连接
- 创建对象：调用connect()方法

```
conn=connect(参数列表)
```

- 参数host：连接的mysql主机，如果本机是'localhost'
- 参数port：连接的mysql主机的端口，默认是3306
- 参数db：数据库的名称
- 参数user：连接的用户名
- 参数passwd：连接的密码
- 参数charset：通信采用的编码方式，推荐使用utf8



## 7.2 pymysql模块的使用

### 对象的方法

- close() 关闭连接
- commit() 事务，所以需要提交才会生效
- rollback() 事务，放弃之前的操作
- cursor() 返回Cursor对象，用于执行sql语句并获得结果

## 7.3 pymysql模块中cursor的使用

## 7.3 cursor的使用

### Cursor对象

- 执行sql语句
- 创建对象：调用Connection对象的cursor()方法

```
cursor1=conn.cursor()
```

### 对象的方法

- close() 关闭
- execute(operation [, parameters ]) 执行语句，返回受影响的行数
- fetchone() 执行查询语句时，获取查询结果集的第一个行数据，返回一个元组
- next() 执行查询语句时，获取当前行的下一行
- fetchall() 执行查询时，获取结果集的所有行，一行构成一个元组，再将这些元组装入一个元组返回
- scroll(value[,mode]) 将行指针移动到某个位置
  - mode表示移动的方式
  - mode的默认值为relative，表示基于当前行移动到value，value为正则向下移动，value为负则向上移动
  - mode的值为absolute，表示基于第一条数据的位置，第一条数据的位置为0

## 7.3 cursor的使用

### 增加

- 创建testInsert.py文件，向学生表中插入一条数据

```
#encoding=utf-8
import MySQLdb
try:
    conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',passwd='mysql')
    cs1=conn.cursor()
    count=cs1.execute("insert into students(sname) values('张良')")
    print count
    conn.commit()
    cs1.close()
    conn.close()
except Exception,e:
    print e.message
```

## 7.3 cursor的使用

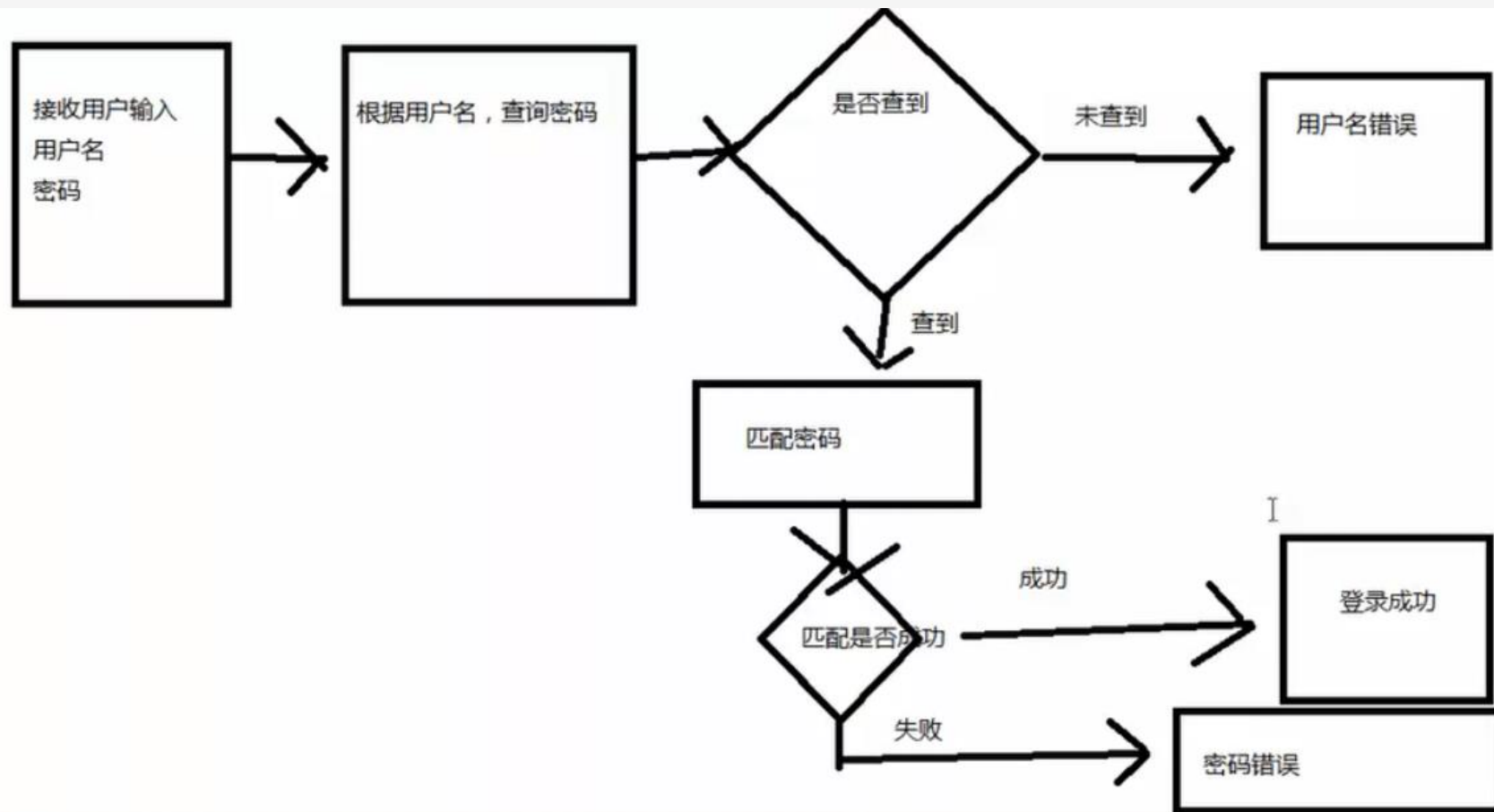
### sql语句参数化

- 创建testInsertParam.py文件，向学生表中插入一条数据

```
#encoding=utf-8
import MySQLdb
try:
    conn=MySQLdb.connect(host='localhost',port=3306,db='test1',user='root',passwd='mysql')
    cs1=conn.cursor()
    sname=raw_input("请输入学生姓名:")
    params=[sname]
    count=cs1.execute('insert into students(sname) values(%s)',params)
    print count
    conn.commit()
    cs1.close()
    conn.close()
except Exception,e:
    print e.message
```

## 案例：用户登录

# 案例：用户登录



# 案例：用户登录

## 示例：用户登录

### 创建用户表userinfos

- 表结构如下
  - id
  - uname
  - upwd
  - isdelete
- 注意：需要对密码进行加密
- 如果使用md5加密，则密码包含32个字符
- 如果使用sha1加密，则密码包含40个字符，推荐使用这种方式

```
create table userinfos(  
  id int primary key auto_increment,  
  uname varchar(20),  
  upwd char(40),  
  isdelete bit default 0
```