Python3 CGI编程 Python3 网络编程

Python3 MySQL 数据库连接 - PyMySQL 驱动

本文我们为大家介绍 Python3 使用 PyMySQL 连接数据库，并实现简单的增删改查。

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 遵循 Python 数据库 API v2.0 规范，并包含了 pure-Python MySQL 客户端库。

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：

```
$ pip3 install PyMySQL
```

如果你的系统不支持 pip 命令，可以使用以下方式安装：

1、使用 git 命令下载安装包安装(你也可以手动下载)：

```
$ git clone https://github.com/PyMySQL/PyMySQL
$ cd PyMySQL/
$ python3 setup.py install
```

2、如果需要制定版本号，可以使用 curl 命令来安装：

```
$ # X.X 为 PyMySQL 的版本号
$ curl -L https://github.com/PyMySQL/PyMySQL/tarball/pymysql-X.X | tar xz
$ cd PyMySQL*
$ python3 setup.py install
$ # 现在你可以删除 PyMySQL* 目录
```

注意：请确保您有root权限来安装上述模块。

> *安装的过程中可能会出现"ImportError: No module named setuptools"的错误提示，意思是你没有安装setuptools，你可以访问https://pypi.python.org/pypi/setuptools 找到各个系统的安装方法。*
>
> *Linux 系统安装实例：*
>
> ```
> $ wget https://bootstrap.pypa.io/ez_setup.py
> $ python3 ez_setup.py
> ```

数据库连接

连接数据库前，请先确认以下事项：

- 您已经创建了数据库 TESTDB.
- 在TESTDB数据库中您已经创建了表 EMPLOYEE
- EMPLOYEE表字段为 FIRST_NAME, LAST_NAME, AGE, SEX 和 INCOME。
- 连接数据库TESTDB使用的用户名为 "testuser" ，密码为 "test123",你可以可以自己设定或者直接使用root用户名及其密码，Mysql数据库用户授权请使用Grant命令。
- 在你的机子上已经安装了 Python MySQLdb 模块。
- 如果您对sql语句不熟悉，可以访问我们的 SQL基础教程

实例：

以下实例链接 Mysql 的 TESTDB 数据库：

实例(Python 3.0+)

```
#!/usr/bin/python3
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()
# 使用 execute() 方法执行 SQL 查询
cursor.execute("SELECT VERSION()")
# 使用 fetchone() 方法获取单条数据.
data = cursor.fetchone()
print ("Database version : %s " % data)
# 关闭数据库连接
db.close()
```

执行以上脚本输出结果如下：

```
Database version : 5.5.20-log
```

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EMPLOYEE：

实例(Python 3.0+)

```
#!/usr/bin/python3
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()
```

```
# 使用 execute() 方法执行 SQL，如果表存在则删除
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
# 使用预处理语句创建表
sql = """CREATE TABLE EMPLOYEE (
FIRST_NAME CHAR(20) NOT NULL,
LAST_NAME CHAR(20),
AGE INT,
SEX CHAR(1),
INCOME FLOAT )"""
cursor.execute(sql)
# 关闭数据库连接
db.close()
```

数据库插入操作

以下实例使用执行 SQL INSERT 语句向表 EMPLOYEE 插入记录：

实例(Python 3.0+)

```
#!/usr/bin/python3
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
# 使用cursor()方法获取操作游标
cursor = db.cursor()
# SQL 插入语句
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
LAST_NAME, AGE, SEX, INCOME)
VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
# 执行sql语句
cursor.execute(sql)
# 提交到数据库执行
db.commit()
except:
# 如果发生错误则回滚
db.rollback()
# 关闭数据库连接
db.close()
```

以上例子也可以写成如下形式：

实例(Python 3.0+)

```
#!/usr/bin/python3
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
# 使用cursor()方法获取操作游标
cursor = db.cursor()
# SQL 插入语句
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
LAST_NAME, AGE, SEX, INCOME) \
VALUES ('%s', '%s', %s, '%s', %s)" % \
```

```
('Mac', 'Mohan', 20, 'M', 2000)
try:
# 执行sql语句
cursor.execute(sql)
# 执行sql语句
db.commit()
except:
# 发生错误时回滚
db.rollback()
# 关闭数据库连接
db.close()
```

以下代码使用变量向SQL语句中传递参数:

```
.................................
user_id = "test123"
password = "password"

con.execute('insert into Login values( %s,  %s)' % \
            (user_id, password))
.................................
```

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。

实例：

查询EMPLOYEE表中salary（工资）字段大于1000的所有数据：

实例(Python 3.0+)

```
#!/usr/bin/python3
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
# 使用cursor()方法获取操作游标
cursor = db.cursor()
# SQL 查询语句
sql = "SELECT * FROM EMPLOYEE \
WHERE INCOME > %s" % (1000)
try:
# 执行SQL语句
cursor.execute(sql)
# 获取所有记录列表
results = cursor.fetchall()
for row in results:
```

```
fname = row[0]
lname = row[1]
age = row[2]
sex = row[3]
income = row[4]
# 打印结果
print ("fname=%s,lname=%s,age=%s,sex=%s,income=%s" % \
(fname, lname, age, sex, income ))
except:
print ("Error: unable to fetch data")
# 关闭数据库连接
db.close()
```

以上脚本执行结果如下：

```
fname=Mac, lname=Mohan, age=20, sex=M, income=2000
```

数据库更新操作

更新操作用于更新数据表的的数据，以下实例将 TESTDB 表中 SEX 为 'M' 的 AGE 字段递增 1：

实例(Python 3.0+)

```
#!/usr/bin/python3
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
# 使用cursor()方法获取操作游标
cursor = db.cursor()
# SQL 更新语句
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M')
try:
# 执行SQL语句
cursor.execute(sql)
# 提交到数据库执行
db.commit()
except:
# 发生错误时回滚
db.rollback()
# 关闭数据库连接
db.close()
```

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 EMPLOYEE 中 AGE 大于 20 的所有数据：

实例(Python 3.0+)

```
#!/usr/bin/python3
import pymysql
# 打开数据库连接
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
```

```
# 使用cursor()方法获取操作游标
cursor = db.cursor()
# SQL 删除语句
sql = "DELETE FROM EMPLOYEE WHERE AGE > %s" % (20)
try:
# 执行SQL语句
cursor.execute(sql)
# 提交修改
db.commit()
except:
# 发生错误时回滚
db.rollback()
# 关闭连接
db.close()
```

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。

实例

实例(Python 3.0+)

```
# SQL删除记录语句
sql = "DELETE FROM EMPLOYEE WHERE AGE > %s" % (20)
try:
# 执行SQL语句
cursor.execute(sql)
# 向数据库提交
db.commit()
except:
# 发生错误时回滚
db.rollback()
```

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。

错误处理

DB API中定义了一些数据库操作的错误及异常，下表列出了这些错误和异常:

异常	描述
Warning	当有严重警告时触发，例如插入数据是被截断等等。必须是 StandardError 的子类。
Error	警告以外所有其他错误类。必须是 StandardError 的子类。
InterfaceError	当有数据库接口模块本身的错误（而不是数据库的错误）发生时触发。 必须是Error的子类。
DatabaseError	和数据库有关的错误发生时触发。 必须是Error的子类。
DataError	当有数据处理时的错误发生时触发，例如：除零错误，数据超范围等等。 必须是DatabaseError的子类。
OperationalError	指非用户控制的，而是操作数据库时发生的错误。例如：连接意外断开、 数据库名未找到、事务处理失败、内存分配错误等等操作数据库是发生的错误。 必须是DatabaseError的子类。
IntegrityError	完整性相关的错误，例如外键检查失败等。必须是DatabaseError子类。
InternalError	数据库的内部错误，例如游标（cursor）失效了、事务同步失败等等。 必须是DatabaseError子类。
ProgrammingError	程序错误，例如数据表（table）没找到或已存在、SQL语句语法错误、 参数数量错误等等。必须是DatabaseError的子类。
NotSupportedError	不支持错误，指使用了数据库不支持的函数或API等。例如在连接对象上 使用.rollback()函数，然而数据库并不支持事务或者事务已关闭。 必须是DatabaseError的子类。

← Python3 CGI编程 Python3 网络编程 →