Python 3 进阶 —— 使用 PyMySQL 操作 MySQL

[PyMySQL](https://github.com/PyMySQL/PyMySQL) 是一个纯 Python 实现的 MySQL 客户端操作库，支持事务、存储过程、批量执行等。

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

安装

|  |  |
| --- | --- |
| 1 | pip install PyMySQL |

创建数据库连接

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **import** pymysql  connection = pymysql.connect(host='localhost',  port=**3306**,  user='root',  password='root',  db='demo',  charset='utf8') |

参数列表：

| **参数** | **描述** |
| --- | --- |
| host | 数据库服务器地址，默认 localhost |
| user | 用户名，默认为当前程序运行用户 |
| password | 登录密码，默认为空字符串 |
| database | 默认操作的数据库 |
| port | 数据库端口，默认为 3306 |
| bind\_address | 当客户端有多个网络接口时，指定连接到主机的接口。参数可以是主机名或IP地址。 |
| unix\_socket | unix 套接字地址，区别于 host 连接 |
| read\_timeout | 读取数据超时时间，单位秒，默认无限制 |
| write\_timeout | 写入数据超时时间，单位秒，默认无限制 |
| charset | 数据库编码 |
| sql\_mode | 指定默认的 SQL\_MODE |
| read\_default\_file | Specifies my.cnf file to read these parameters from under the [client] section. |
| conv | Conversion dictionary to use instead of the default one. This is used to provide custom marshalling and unmarshaling of types. |
| use\_unicode | Whether or not to default to unicode strings. This option defaults to true for Py3k. |
| client\_flag | Custom flags to send to MySQL. Find potential values in constants.CLIENT. |
| cursorclass | 设置默认的游标类型 |
| init\_command | 当连接建立完成之后执行的初始化 SQL 语句 |
| connect\_timeout | 连接超时时间，默认 10，最小 1，最大 31536000 |
| ssl | A dict of arguments similar to mysql\_ssl\_set()’s parameters. For now the capath and cipher arguments are not supported. |
| read\_default\_group | Group to read from in the configuration file. |
| compress | Not supported |
| named\_pipe | Not supported |
| autocommit | 是否自动提交，默认不自动提交，参数值为 None 表示以服务器为准 |
| local\_infile | Boolean to enable the use of LOAD DATA LOCAL command. (default: False) |
| max\_allowed\_packet | 发送给服务器的最大数据量，默认为 16MB |
| defer\_connect | 是否惰性连接，默认为立即连接 |
| auth\_plugin\_map | A dict of plugin names to a class that processes that plugin. The class will take the Connection object as the argument to the constructor. The class needs an authenticate method taking an authentication packet as an argument. For the dialog plugin, a prompt(echo, prompt) method can be used (if no authenticate method) for returning a string from the user. (experimental) |
| server\_public\_key | SHA256 authenticaiton plugin public key value. (default: None) |
| db | 参数 database 的别名 |
| passwd | 参数 password 的别名 |
| binary\_prefix | Add \_binary prefix on bytes and bytearray. (default: False) |

执行 SQL

* cursor.execute(sql, args) 执行单条 SQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | *# 获取游标*  cursor = connection.cursor()    *# 创建数据表*  effect\_row = cursor.execute('''  CREATE TABLE `users` (  `name` varchar(32) NOT NULL,  `age` int(10) unsigned NOT NULL DEFAULT '0',  PRIMARY KEY (`name`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8  ''')    *# 插入数据(元组或列表)*  effect\_row = cursor.execute('INSERT INTO `users` (`name`, `age`) VALUES (%s, %s)', ('mary', **18**))    *# 插入数据(字典)*  info = {'name': 'fake', 'age': **15**}  effect\_row = cursor.execute('INSERT INTO `users` (`name`, `age`) VALUES (%(name)s, %(age)s)', info)    connection.commit() |

* executemany(sql, args) 批量执行 SQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | *# 获取游标*  cursor = connection.cursor()    *# 批量插入*  effect\_row = cursor.executemany(  'INSERT INTO `users` (`name`, `age`) VALUES (%s, %s) ON DUPLICATE KEY UPDATE age=VALUES(age)', [  ('hello', **13**),  ('fake', **28**),  ])    connection.commit() |

注意：INSERT、UPDATE、DELETE 等修改数据的语句需手动执行connection.commit()完成对数据修改的提交。

获取自增 ID

|  |  |
| --- | --- |
| 1 | cursor.lastrowid |

查询数据

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | *# 执行查询 SQL*  cursor.execute('SELECT \* FROM `users`')  *# 获取单条数据*  cursor.fetchone()  *# 获取前N条数据*  cursor.fetchmany(**3**)  *# 获取所有数据*  cursor.fetchall() |

游标控制

所有的数据查询操作均基于游标，我们可以通过cursor.scroll(num, mode)控制游标的位置。

|  |  |
| --- | --- |
| 1  2 | cursor.scroll(**1**, mode='relative') *# 相对当前位置移动*  cursor.scroll(**2**, mode='absolute') *# 相对绝对位置移动* |

设置游标类型

查询时，默认返回的数据类型为元组，可以自定义设置返回类型。支持5种游标类型：

* Cursor: 默认，元组类型
* DictCursor: 字典类型
* DictCursorMixin: 支持自定义的游标类型，需先自定义才可使用
* SSCursor: 无缓冲元组类型
* SSDictCursor: 无缓冲字典类型

无缓冲游标类型，适用于数据量很大，一次性返回太慢，或者服务端带宽较小时。源码注释：

Unbuffered Cursor, mainly useful for queries that return a lot of data, or for connections to remote servers over a slow network.

Instead of copying every row of data into a buffer, this will fetch rows as needed. The upside of this is the client uses much less memory, and rows are returned much faster when traveling over a slow network or if the result set is very big.

There are limitations, though. The MySQL protocol doesn’t support returning the total number of rows, so the only way to tell how many rows there are is to iterate over every row returned. Also, it currently isn’t possible to scroll backwards, as only the current row is held in memory.

创建连接时，通过 cursorclass 参数指定类型：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | connection = pymysql.connect(host='localhost',  user='root',  password='root',  db='demo',  charset='utf8',  cursorclass=pymysql.cursors.DictCursor) |

也可以在创建游标时指定类型：

|  |  |
| --- | --- |
| 1 | cursor = connection.cursor(cursor=pymysql.cursors.DictCursor) |

事务处理

* 开启事务 connection.begin()
* 提交修改 connection.commit()
* 回滚事务 connection.rollback()

防 SQL 注入

* 转义特殊字符 connection.escape\_string(str)
* 参数化语句 支持传入参数进行自动转义、格式化 SQL 语句，以避免 SQL 注入等安全问题。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | *# 插入数据(元组或列表)*  effect\_row = cursor.execute('INSERT INTO `users` (`name`, `age`) VALUES (%s, %s)', ('mary', **18**))  *# 插入数据(字典)*  info = {'name': 'fake', 'age': **15**}  effect\_row = cursor.execute('INSERT INTO `users` (`name`, `age`) VALUES (%(name)s, %(age)s)', info)  *# 批量插入*  effect\_row = cursor.executemany(  'INSERT INTO `users` (`name`, `age`) VALUES (%s, %s) ON DUPLICATE KEY UPDATE age=VALUES(age)', [  ('hello', **13**),  ('fake', **28**),  ]) |